Master Thesis Artificial Intelligence

Hidden Markov Model structure analysis and simplification for Gesture Recognition

by

Gideon Maillette de Buy Wenniger

Direct Supervisor : Basilio Noris[a]
Supervisors : Prof. Aude Billard[a]**, Prof. Theo Gevers**[b]**, Dr. Nicu Sebe**[b]

[a] Learning and Systems Laboratory (LASA)
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland

[b] Intelligent Sensory Information System Group (ISIS)
University of Amsterdam (UvA)
1098 VA Amsterdam, the Netherlands

gmaillet@science.uva.nl
October 30, 2008

Abstract

Gesture Recognition is a research area that has received much attention in the last decade. The *Hidden Markov Model(HMM)* has been extensively used for it. As a sidetrack, research in unsupervised gesture recognition and segmentation have received increasingly much interest.

Our research is motivated by the strongly related problems of segmentation, unsupervised structure discovery, incremental gesture learning and HMM distance computation. Those problems have in common that they all benefit from if not rely on some form of simplification of a HMM or set of HMMs. We introduce two new and complementary approaches for the simplification and structural analysis of *Hidden Markov Models* (HMMs). The first approach consists of segmenting HMMs into structural parts. To this end we introduce a new algorithm that uses the information provided by the HMM Transition Matrix to find structural groups of states in the HMM corresponding to *Cycles* and *Trajectories*. The second approach consists of directly compressing and Merging HMMs based on the model parameters. This is effectuated by the implementation of a direct HMM compression and merging algorithm, that exploits the spatial information of a Gaussian Mixture Model HMM (GMM-HMM) to find groups of states suitable to be merged.

Our structural analysis algorithm was tested with a segmentation task on database of nine gestures recorded with the Xsens posture tracking system. A big HMM is trained for a sequence containing multiple gestures, and state groups extracted by the algorithm are used to segment the data. The extracted groups gave significant better performance on the segmentation task than a baseline method using only single states. This shows extracted groups contain relevant information, and can be useful for classification or data segmentation.

We tested our second approach with the Interactplay Dataset, a hand gesture database of 3D hand trajectories. The tests established the success of the compression and merging algorithms, by comparison to the best results achieved with direct training on the original data. We also compared with the existing method for compression and merging, which works by resampling data from the models followed by training of a new model on the sampled data. Our algorithm was superior for HMM compression, but gave

worse results for direct merging than could be achieved by data re-sampling merging.

The contributions of this thesis are threefold: (1) it contributes to segmentation and unsupervised structure discovery research by proposing a new algorithm that performs HMM analysis and HMM segmentation. (2) it contributes to incremental gesture recognition and distributed frameworks by introducing new methods for the direct compression and merging of HMMs. (3) it gives a good overview of current research in unsupervised gesture recognition. The three new algorithms introduced promise to be useful to new and existing applications.

Keywords : Gesture Recognition, Hiddeon Markov Models (HMMs), Human Computer Interaction (HCI) , Model Compression and Merging, Gaussian Mixture Model Clustering (GMM-Clustering), Incremental Gesture Recognition, Unsupervised Learning

Acknowledgments

Studying in Switzerland, working in the great Federal institutes EPFL and ETHZ has been a dream for me for a long time. During my first stay in Switzerland at ETHZ Zürich I had such a wonderful time, and I felt I grew so much as a result of this experience that I couldn't wait to repeat it. Almost a year ago I was here in Lausanne, attracted by the EPFL and the Bio Inspired Robotics Group. It was through this group and my initial talks with the people there that I became aware of the LASA group and the project proposed by Basilio Noris on gesture recognition. This project indeed was what I had been looking for, being a student from UvA with a strong background and interest in Machine Learning, and already having an interest in gesture recognition for a long time. My great gratitude to Aude Billard for welcoming me in the LASA group, as well as to Theo Gevers and Nicu Sebe to make my exchange possible from the side of UvA. Here I am also most indebted to Joke Hendriks, who played another key role in enabling me to go to Switzerland for the second time. My project has been very interesting from the beginning, when I started getting the basic gesture recognition working with the idea of a real-time application in mind. Basilio Noris helped me a lot to get things working with the webcam and the tracker, as I was struggling to build the initial setup based on HTK and Java. In December I had some stimulating discussions with Sylvain Calinon on the topic of hierarchical gesture recognition, and the possibility to directly compress Gaussian Mixture Models. Thanks to those discussion I learned about the different works on Incremental and Hierarchical Gesture recognition, and the existence of a direct GMM compression method. This lead to the implementation of the GMM compression algorithm and some pilot studies on layered HMMs, which I did in January. Eventually this all contributed to give me the idea of HMM sructure analysis. The many discussions I had with Basilio while working on HISAA were a great help in working out the ideas into working algorithms, and stimulated me a lot. I also feel a big gratitude towards Aude Billard for giving me tough but good feedback on my papers and work, while being so tolerant to a perhaps sometimes unusual way of working. I feel I learned a lot about writing papers and reports and representing ideas thanks to her feedback on my writings and our monthly meetings. My project must have been the one that went

through the most of changes I sometimes think, but I feel it has been for the best. Indeed it has been wonderful to have the time and opportunity to try something, then think again and try something else altogether. Thanks Theo and Nicu for your good support from Amsterdam, and the many useful comments on my Thesis and papers.

My gratitude also to all my colleagues in the LASA group and EPFL for their friendship and support during my stay in Lausanne and work on the project. My special gratitude to Sankha Ghosh, for his invaluable friendship and company, without whom being in Switzerland would not be the same. Thanks to my friends and family in Amsterdam for accepting my physical absence, while ensuring the continuity in our contact.

Thanks also to the *Ecole Polytechnique Fédérale de Lausanne* for accepting me as a foreign student and allowing me to work in this great academic and stimulating environment. I look most forward to a next episode here.

Contents

1 Introduction **11**

2 Time Sequence Learning **15**
2.1 Supervised Learning 16
2.2 Unsupervised Learning 16
2.2.1 Semi-supervised Learning 17
2.2.2 Expectation Maximization 17
2.3 Relevance to our Project 18
2.4 Hidden Markov Models 18
2.4.1 Formal Description HMMs 19
2.4.2 The Three basic problems for HMMs 20
2.4.3 HMM Model Merging and Compression by Data resampling 24
2.5 Alternative Solutions 24

3 State of the Art **27**
3.1 Incremental HMM Clustering 27
3.2 Data Segmentation by clustering small HMMs 29
3.3 Training of sparse HMMs with a new *Entropic Prior* Baum-Welch variant 30
3.3.1 Entropic Prior 30
3.3.2 MAP estimator 30
3.3.3 Entropic HMM training 31
3.3.4 Results 31
3.4 Hierarchical Hidden Markov Models 31
3.4.1 Model description 32
3.4.2 Inference and Learning 33
3.4.3 Applications 33
3.5 Hierarchical Hidden Markov Models combined with Bayesian Model Adaptation 34
3.6 Simple Segmental Clustering 34
3.7 Hidden Markov Model Induction by Bayesian Model Merging 35
3.7.1 Model Priors 35

3.7.2 Parameter Priors 36
3.7.3 The Algorithm 37
3.7.4 Evaluation 38

I New Research **41**

4 Segmenting, Compressing and Merging HMMs **43**
4.1 Hidden Markov Model Implicit Structure Analysis 43
4.1.1 Cycle extraction 44
4.1.2 Trajectory extraction 45
4.1.3 Setting the Thresholds 46
4.2 HMM compression and HMM merging 46
4.2.1 HMM Compression 48
4.2.2 HMM Merging 50
4.2.3 Computational Complexity Analysis 53
4.3 Implementation 54
4.3.1 HMM Libraries 54

5 Experiments **55**
5.1 Gesture Databases 55
5.1.1 Dataset for HMM segmentation 55
5.1.2 Dataset for HMM Compression and HMM Merging . . 55
5.2 Experiments 57
5.2.1 Experimental set-up HISAA tests 57
5.2.2 Experimental set-up HMM compression and HMM merging tests 59
5.3 Results 61
5.3.1 Results HISAA tests 61
5.3.2 Results HMM compression and HMM merging tests . 68
5.4 Discussion and further work 73
5.4.1 HISAA 73
5.4.2 HMM compression and HMM merging 77
5.4.3 Comparison of the two approaches 78
5.4.4 Further work 78

6 Conclusion **81**

Bibliography **85**

7 Appendix **89**
7.1 Appendix HISAA 89
7.1.1 Algorithms 89
7.1.2 Results 89
7.2 Appendix HMM Compression and Merging 90

7.2.1 Complexity Analysis 90
7.2.2 Algorithm Pseudocode 94
7.2.3 Remaining Results 96

Chapter 1

Introduction

Gesture recognition has been extensively studied over the last decade in the AI community. Good overviews are given in [32, 37, 46, 51].Gesture Recognition has many applications for the improvement of human computer interaction (HCI) [22, 25], as well as facilitating easier interaction with robots, particularly providing more efficient ways to instruct them about specific movements and actions [10, 9]. Another important application subdomain is sign-language understanding [42, 30], which can be combined with machine translation methods to greatly improve the means of deaf people to communicate in a natural way using their sign language. With the introduction of mouse gestures for internet browsers, gesture recognition has recently also found its application to the mass market, similar to the way this has previously happened with speech recognition.

An important problem for gesture recognition, is the extraction of appropriate features to code the gestures [39, 51]. Ideally one would know the precise posture of the subject performing the gesture, which is unambiguously given by the complete set of joint angles between all limbs. However, such complete posture information is very hard to obtain, and is furthermore not completely necessary for most applications. Therefore researchers often resort to simpler posture models, that try to measure for example only the position of the hands, head and torso. Colored clothing and markers can help to simplify the the visual recognition and tracking task that must be performed to retrieve the posture information. Another alternative is to use mounted sensor systems such as the Xsens system, that measure posture information directly using gyroscopes. Some researchers try to extract full posture information based on only visual information and without the help of markers or colored clothing, a nice example of this is the system that was developed at Boston University by Prof. Sclaroff and colleages [29]. For some forms of gesture recognition, particularly sign language recognition, the detailed information about the hand postures is essential as well [11]. In this cases, positional information is often combined

with features extracted from the sub-frames in the video belonging to the hands. Our research was not particularly concerned with the details of low-level feature extraction, but much more with the learning methods required to do gesture recognition based on the recorded gestures. Initially we experimented with visual feature extraction methods, using simple tracking methods for 2-d and 3-d tracking, based on the *mean shift algorithm*. Later we decided to use the Xsens system, which provides much richer and more robust posture information than can be achieved straightforward by visual methods.

Gesture recognition may be generalized as the recognition of time-spatial sequences, and is in that frame identical to speech recognition, except that the spatial features that make up the signal are different. It is therefore no surprise that this research could profit from well established methods, initially developed for speech recognition. Gesture recognition has to deal with variance over different demonstrations of the same gesture, both spatially and temporally. A gesture, or parts of it, can be executed slightly faster or slower. Similarly, parts of the gesture may be executed differently in the sense of being scaled or moved over different demonstrations. In either case their will always be a fair amount of difference between different demonstrations of the same gesture, even though the essential dynamics of the gesture remain the same. The recognition method must be robust enough to deal with such variation.

The central tool for speech recognition is the *Hidden Markov Model (HMM)* [38]. This tool proved to be very appropriate for gesture recognition and motion analysis as well, and has been extensively used for those tasks over the last decade. One of the main strengths of Hidden Markov Model is its ability to effectively model both the temporal and spatial characteristics of a gesture, as well as the variance that can be observed over different demonstrations. Application of HMMs to clustering and automatic segmentation is more recent [12, 52, 28, 47]. Hierarchical Hidden Markov Models [15] (HHMMs) and HMM distance metrics combined with clustering methods play an important role in this research.

While gesture recognition has been extensively studied, there exist still many problems when building models for multiple users when the gesture data is not completely segmented. When gesture models are build for different users in parallel and we want to integrate those models efficiently, special model merging methods are required. Similarly, when we build a model from gesture examples that actually contain multiple (sub-)gestures, it is helpful if we can segment the model into parts corresponding to the different (sub-)gestures. We thus consider the closely related problems of simplifying complex HMMs and analyzing their structure, as well as compressing and merging HMMs. We take two different approaches. Our first approach focuses on the analysis and simplification of HMMs by

splitting them into different parts. The second approach we take is to compress and merge HMMs as a whole, effectively reducing the amount of states and forming simpler models.

In our first approach we analyze and use the structure within a classical HMM directly to segment its states into meaningful groups. In the second approach we perform compression and merging of HMMs, based on the usage of spatial information to find groups of states that can be combined.

This master thesis is organized as follows. In the first part, spanning chapters 2-4 we introduce the theoretical foundations for our research. In the second part, spanning chapters 5-8 we discuss our new research on Hidden Markov Model structure analyses and compression. We first discuss Machine Learning in general in the following chapter, followed by a discussion of Time Sequence Learning in the third Chapter. In this chapter we also thoroughly cover HMMs, which form the backbone for the domain of our research. In chapter four we discuss the state of the art in research on unsupervised gesture segmentation and clustering. Then in chapter five our first new method for HMM structure analysis is introduced, followed by our second method that tackles HMM compression and merging which is covered in chapter six. In chapter 7 we give a comparison between the two approaches. We end with the conclusion in the last chapter.

Chapter 2

Time Sequence Learning

Gesture recognition, segmentation, compression and merging are all cases of the general problem of Machine Learning. Machine Learning [35, 48] is concerned with the problem of making generalizations and deriving new information from information in the form of data. It is implemented by learning algorithms. There are two general forms of machine learning, one called *Supervised learning* which builds on labeled data, and another called *Unsupervised learning* or clustering which attempts to derive information from unlabeled data. Time Sequence learning is the more specific problem we are dealing with. It is distinguished from general machine learning by the temporal component, which means that training examples consist of an ordered sequence of data points rather than just one data point. As such time sequence learning covers still a broad class of learning problems, most prominently speech recognition, video classification and gesture recognition. Time Sequence learning relies on both supervised learning and unsupervised learning. In its simplest form gesture models are directly trained on labeled and segmented training sets. However in more difficult cases labeling and segmentation may be at least partly part of the problem, thus requiring unsupervised or semi-supervised forms of learning to find the best segmentation points and labels for the unlabeled data. In the last decade time sequence learning has received a great boost, as well established techniques from Natural Language processing (NLP) started being applied to other domains. The Hidden Markov Model (HMM) is one of the most used and successful tools in NLP, and has proven to be particularly powerful in its application to speech recognition. Combined with N-gram Language models it offers a very strong framework for building speech recognition systems. But it is a useful tool for language understanding systems in general, as shown by its equally successful application to handwriting recognition [20] and sign language understanding [42].
We will now discuss supervised learning and unsupervised learning in general, followed by a detailed discussion of HMMs and associated learning

algorithms, which are particularly important for the problem of gesture recognition. We end the chapter by a short discussion of alternative methods that can be used to replace HMMs.

2.1 Supervised Learning

The sub-field of supervised learning has received most research attention in the past and is best understood. In supervised learning, a training set of hand-labeled examples is provided to a learning algorithm, which then produces a classification function that is able to classify new examples based on information from the training data. The classification function gains its strength by generalizing over the observed training data. This requires assuming restrictions on the form of possible relations between attributes of the training examples, and the class of those examples. Those restrictions can be more or less explicit in the learning algorithm, but are always there. The restrictions are characterized by a *Hypothesis Space*, which is the set of possible relations between the data attributes and class type Exactly those restrictions allows the learning algorithm to extract more information from the training set than just the set of examples itself.

2.2 Unsupervised Learning

While supervised learning has been quite successful and knows many applications, in many cases the scheme were a trainer has to generate input in the form of long sets of labeled examples is not feasible or at least not very attractive. Unsupervised learning, were no explicitly labeled data is required is an attractive alternative. Unsupervised learning in its purest form takes a dataset and then uses the information in the data to form a (possibly hierarchical) grouping of the examples. The grouping puts together examples that are similar by some similarity measure. Many clustering schemes are parametric, and thus require some structural form of the grouping, by this restriction providing the way to generalize beyond the data.
Other clustering algorithms have no explicit parameters, but through the distance measure they use and the ways to group the data they allow, still introduce constraints on the possible ways of clustering data. A good clustering function exhibits certain properties. First *scale invariance* implies that a scaling of all distances between all examples in the dataset does not change the clustering of the dataset. Second, *richness* implies that for every possible partition p of the data d, a set of distances $Dist$ between the examples can be constructed so that the clustering function returns p when given d and $Dist$. Finally, *consistency* implies that if the distances within clusters are decreased and the distances between points in different clusters

expanded, the obtained result remains the same. It has been proven by Kleinberg [26] that for more than two examples, only two of those properties can be satisfied simultaneously by a clustering function. Thus the intuitive idea that we have to make some assumptions about the structure of the data to generalize beyond it or extract more information from it, holds for unsupervised learning as much as for supervised learning.

2.2.1 Semi-supervised Learning

While purely supervised learning is one extreme and clustering the other extreme in the domain of machine learning, much work lies between those two extremes. In *reinforcement learning*, training information is still given, but rather than providing labels for states (examples) directly feedback is given much more indirectly in the form of observed state transitions and rewards received by the learner upon the execution of certain actions in certain states. In partially supervised learning, first some labeled data is provided to form an initial classification function. Then using this classification function, unlabeled data is labeled, and if this is done with sufficient confidence the newly labeled data is incorporated into the training set. Then based on the expanded training set, a new classification function is learned. This form of bootstrapping-learning has been shown to often improve the performance over using only the labeled set, despite the fact that automatically labeling examples introduces uncertainty about the correctness of the labels in the training data.

2.2.2 Expectation Maximization

Another common scenario within the domain of machine learning is that some information is available in the examples but also some information is hidden. The assumed form of the hidden information introduces restrictions on the set of allowable *complete data*. Given this restrictions and the observed *incomplete data*, it is then possible to find the most likely *complete data* given the *incomplete data*. This can be formulated more generally using terminology from statistics, as the problem of finding the most likely a posteriori hypothesis given initial information about the domain in the form of the Model Prior in combination with the observed data. Finding such a *MAP (Maximum A Posteriori) hypothesis* is often not directly solvable, but can be tackled by iterative algorithms such as the Expectation-Maximization algorithm (EM-algorithm) [13, 34]. The EM-algorithm iterates over an *E-step* in which the incomplete data is 'completed' by providing the best values for the hidden attributes based on the current model. This is followed by the *M-step* in which the model parameters are re-estimated, effectively maximizing Model likelihood given the new complete data as generated in the preceding *E-step*. The EM-algorithm is widely used in its has many

variants (see [33]). It forms the core of modern speech recognition and statistical machine translation.

2.3 Relevance to our Project

Our original research is concerned with gesture recognition. In the simplest form, gesture recognition requires the learning of gesture models and then the mapping of unlabeled gesture sequences to the best fitting learned models. We started from this most simple model, but gradually became more interested in unsupervised gesture learning, gesture segmentation and gesture model merging. All those problems have in common, that they make weaker assumptions about the nature of the information that is used to learn the gesture models, and as such can be more suitable for real applications, in which most available data will be unlabeled. Merging and compression of learned models are special kinds of Machine Learning, that are central to our research. Structural analysis of learned models is another closely related topic, that is equally important to our work. Our model merging algorithm relies on a higher-order form of the popular K-means algorithm, which in turn is a special simpler case of the general EM-algorithm. The models we merge or analyze themselves are trained using the Baum-Welch algorithm, which is also a special form of EM, tailored to Hidden Markov Models (HMMs). Our project in its final form is really a machine learning project, using Baum-Welch training, K-means clustering, Monte Carlo Model-sampling and Dynamic Programming as central tools. Baum-Welch Training, which is a special case of the general EM-algorithm is discussed in the next section, as are different dynamic programming algorithms used with HMMs. K-means clustering and Monte Carlo Model-sampling are discussed in light of our new algorithms in the second part of this thesis.

2.4 Hidden Markov Models

Hidden Markov Models encode both spatial and temporal properties of time sequences. Spatial properties are modeled by means of probability distributions over the discrete or continuous outputs of the states in the model. Gestures and meaningful movement sequences can be reduced to ordered sequences of positions or joint angles. Certain positions in space or joint-space are visited in a certain sequence and this makes up the gesture. Those typical position sequences can be further abstracted as sequences of states, with every state associated to certain prototypical positions. This is exactly the view taken in HMMs. To encode the prototypical position associated with every state and the amount of expectable deviation from it, Gaussian Models or Gaussian Mixture Models are very suitable. They

model the expected point (the Mean) to be observed at the state, as well as the variance from that point (given by the Covariance Matrix). Temporal behavior may be reduced to the typical state sequences one expects to observe for a certain gesture. In the HMM this is coded by assigning probabilities to the different possible transitions from every state to every other state including itself. These probabilities, so called transition chance, are compactly stored in a Transition Matrix for the HMM. To recap, the combination of states with state transition chances and state output distributions allows HMMs to adequately model the structure of many time-spatial phenomena.

HMMs have several big advantages over the alternatives that can be used for sequence learning. First of all they are conceptually simple and their behavior can be directly understood from the model parameters. Secondly their exists an efficient and well understood learning algorithm to customize them to the data they must predict, namely the Expectation-Maximization (EM) Algorithm. Two other important problems in Sequence Learning for recognition are the computation of model likelihood given a certain observation sequence and finding the best hidden state sequence given an observation sequence and the HMM. The Forward Algorithm offers an efficient solution to the first problem, and the Viterbi Algorithm effectively solves the second. All these algorithms rely on dynamic programming, exploiting the Markov properties of the model, and having low polynomial complexities. The Forward and Viterbi Algorithm both have complexity O(n2 * l) with for n the number of states and for l the sequence length, while the Viterbi Algorithm has complexity ... Alternative Algorithm for Sequence Leaning

2.4.1 Formal Description HMMs

A Markov process is a stochastic process with the property that the output generated by the process only depends on a limited amount of history, i.e. only the previous state for discrete first order Markov processes. A discrete first order Markov model is a model consisting of states, state transitions and emissions, with the state transitions and emissions being only dependent on the current state. A Hidden Markov Model (HMM) is double stochastic process of which one part is not directly observable. The states in a HMM can be only be indirectly viewed through a sequence of observations produced by the second stochastic process, that produces observations from the (hidden) states. A HMM

$$\lambda(T, O, \pi) \tag{2.1}$$

with N states is specified by a transition matrix T, an output distribution O and an initial state distribution π. T specifies for every state pair S_i, S_j the probability of going from S_i to S_j $P(S_j|S_i) = a_{ij}$ with $\forall_i \sum_{j=1}^{N} a_{ij} = 1$. The initial state distribution π assigns a probability π_i to starting in each state $1 \leq i \leq N$ with $\sum_{i=1}^{N} \pi_i = 1$. Finally, the output distribution O

specifies how output is generated in every state and can be either a discrete (i.e. multinomial) distribution or continuous output distribution, typically a Gaussian Mixture Model. In the case of a continuous output distribution modeled by a mixture of Gaussians, the output distribution is given by a set of mean vectors μ and covariance matrices Σ specifying all mixture components as well as a mixmatrix m that specifies the weights of the different mixture components. A Gaussian Mixture Model HMM (GMM-HMM)

$$\lambda(T, \mu, \Sigma, m, \pi) \tag{2.2}$$

is thus completely specified by its parameters T for transitions, μ, Σ, m for Gaussian output distributions and π for the initial state distribution.

2.4.2 The Three basic problems for HMMs

Following Rabiner [38] there are three basic problems that must be solved to use HMMs in real-world applications, those are:
Problem 1: Given the observation sequence $O = O_1 O_2 ... O_T$ and the model π , how to efficiently compute te likelihood $P(O|\lambda$ of the observation sequence given the model ?
Problem 2: Given the observation sequence and the model, how to determine the most likely state sequence $Q = q_1 q_2 .. q_T$?
Problem 3: How to adjust the model parameters λ to maximize the chance of the observations $P(O|\lambda)$, in order to find the best model for them ?

Solution to Problem 1

The probability of the observation sequence O given the model λ is given by summing the joint probability of state sequence and observation sequence over all state sequences:

$$\begin{aligned} P(O|\lambda) &= \sum_{allQ} P(O|Q, \lambda) P(Q|\lambda) \\ &= \sum_{q_1, q_2, \ldots, q_T} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) ... a_{q_{T-1} q_T} b_{q_T}(O_T) \end{aligned} \tag{2.3}$$

In this form however, the calculation requires on the order of $2T \times N^T$ calculations. There is an efficient way to compute $P(O|\lambda)$ using dynamic programming, exploiting the Markov property of state transitions in the HMM. Consider the forward variable

$$\alpha_t(j) = P(O_1 O_2 ... O_T, q_t = S_i|\lambda) \tag{2.4}$$

in other words the chance of observations 1 to T plus state S_i at time t given the model λ. Define $b_j(O_i)$ to be the chance on observation i in state j, a_{ij} the transition chance of going from state i to state j, and π_i the prior chance

for state i. By induction we can solve $\alpha_t(j)$, as follows:
1) Initialization:

$$\alpha_1(i) = \pi_i b_i(O_1) \; 1 \leq i \leq N \tag{2.5}$$

2) Induction:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \quad 1 \leq t \leq T-1$$
$$1 \leq j \leq N \tag{2.6}$$

3) Termination:

$$P(O|\lambda) = \sum_{i-1}^{N} \alpha_T(i) \tag{2.7}$$

Solution to Problem 2

The second problem involves finding the best state sequence $Q = \{q_1, q_2, ..., q_T\}$ for the given observation sequence $O = \{O_1, O_2, ..., O_T\}$. We introduce a new quantity $\delta_t(j)$ which is most likely state sequence that generates the first t observations and ends in state i:

$$\delta_t(j) = max_{q_q, q_2, .., q_{t-1}} P(q_1 q_2 ... q_t = i, O_1 O_2 ... O_t|\lambda) \tag{2.8}$$

Induction gives:

$$\delta_{t+1}(j) = (max_i \delta_t(i) a_{ij}) b_j(O_{t+1}) \tag{2.9}$$

To retrieve the best state sequence, we need to keep the argument that maximizes (2.9). We do this by keeping an array of pointers $\psi_t(j)$ storing for every state j and time t the previous state S_{t-1} that gave the most likely path into state j at time t. The complete procedure for finding the best state sequence, called the *Viterbi Algorithm* is then as follows:
1) Initialization:

$$\delta_1(j) = \pi_i b_i(O1) \; 1 \leq i \leq N \tag{2.10}$$
$$\psi_1(i) = 0 \tag{2.11}$$

2) Induction:

$$\delta_t(j) max_{1 \leq i \leq N} \; \delta_{t-1}(i) a_{ij}) b_j(O_t) \quad 2 \leq t \leq T$$
$$1 \leq j \leq N \tag{2.12}$$
$$\psi_t(j) = argmax_{1 \leq i \leq N} \; (\delta_{t-1}(i) a_{ij}) \quad 2 \leq t \leq T$$
$$1 \leq j \leq N \tag{2.13}$$

3) Termination:

$$p^* = max_{1 \leq i \leq N} \; [\delta_T(i)]$$
$$p_T^* = argmax_{1 \leq i \leq N} \; [\delta_T(i)] \tag{2.14}$$

4)Path backtracking :

$$q_t^* = \psi_{t+1}(q_{t+1}) \qquad t = T-1, T-2, ..., 1 \tag{2.15}$$

Solution to Problem 3

The final and most difficult problem involves the adaptation of model parameters λ to best explain the observations, formally to maximize $P(O|\lambda)$. There exists no closed form solution to this maximization, but an iterative solution called *the Baum-Welch algorithm*, which is basically a variant of the general EM-algorithm exists. For this algorithm we first need another probability value, given by a backward variable $\beta_t(i)$ defined as

$$\beta_t(i) = P(O_{t+1}O_{t+2}...O_T|q_t = S_i, \lambda) \tag{2.16}$$

the probability of the partial observation sequence starting at t+1 and going to the end, given state S_i at time t and the model λ. Again $\beta_t(j)$ can be solved by induction, as follows:
1) Initialization:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{2.17}$$

2) Induction:

$$\beta_T(i) = \sum_{j=1}^{N} a_{ij}b_jO_{t+1})\beta_{t+1}(j)t = T-1, T-2, ..., 1, \quad 1 \leq j \leq N \tag{2.18}$$

Another variable we need is $\xi_t(i,j)$, defined as the probability of being in state S_i at time t and in state S_j at time t + 1, given the observation sequence and the model:

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j|O, \lambda) \tag{2.19}$$

Using the definitions of forward and backward variables we can write $\xi_t(i,j)$ in the form:

$$\begin{aligned} \xi_t(i,j) &= \frac{\alpha_i(t)a_{ij}\beta_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} \\ &= \frac{\alpha_i(t)a_{ij}\beta_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i(t)a_{ij}\beta_j(O_{t+1})\beta_{t+1}(j)} \end{aligned} \tag{2.20}$$

$$\tag{2.21}$$

The numerator in the term is $P(q_t = S_i, q_{t+1} = S_j, O|\lambda)$ so that division by $P(O|\lambda)$ gives the desired probability measure. Now define

$$\gamma_t(i) = P(q_t = S_i|O, \lambda) \tag{2.22}$$

the probability of being in state S_i at time t, given the observation sequence and the model. Using the forward and backward variables this can be expressed as:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)} \tag{2.23}$$

Now $\gamma_t(i)$ can be related to $\xi_t(i,j)$, by summing over j:

$$\gamma_t(i) = \sum_{j=1}^{N}\xi_t(i,j) \tag{2.24}$$

The expected number of transitions from a state S_i and the expected number of transitions from S_i to a particular state S_j can next be computed by summing $\gamma_t(i)$ and $\xi_t(i)$ respectively over the time t:

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{expected number of transitions from } S_i \tag{2.25}$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) = \text{expected number of transitions from } S_i \text{ to } S_j \tag{2.26}$$

Using the above formulas a set of re-estimation formulas for the parameters of an HMM can be derived:

$$\overline{\pi_j} = \text{expected number of times in state } S_i \text{ at time (t = 1)} = \gamma_i(i) \tag{2.27}$$

$$\overline{a_{ij}} = \frac{\text{expected number of transitions from } S_i \text{ to } S_j}{\text{expected number of transitions from } S_i \text{ to } S_j} = \frac{\sum_{t=1}^{T-1} \gamma_t(i)}{\sum_{t=1}^{T-1} \xi_t(i,j)} \tag{2.28}$$

Those formulas reestimate the state priors and transition chances. Re-estimation of the mixture components requires us to first define one more variable: Define

$$\zeta_t(j,k) = \left[\frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}\right] \left[\frac{c_{jk} N(O_t, \mu_{jk}, \Sigma_{jk}}{\sum_{M=1}^{M} c_{jm} N(O_t, \mu_{jm}, \Sigma_{jm}}\right] \tag{2.29}$$

the probability of being in state j at time t with the kth mixture component accounting for O_t. Here c_{jk} is the prior chance for the kth mixture component and $N(O_t, \mu_{jk}, \Sigma_{jk})$ is the probability given to the observations by the kth mixture component. Now using $\zeta_t(j,k)$ we can finally also give the re-estimation formulas for the mixture components:

$$\overline{c_{jk}} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T} \sum_{k=1}^{M} \gamma_t(j,k)} \tag{2.30}$$

$$\overline{\mu_{jk}} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) O_t}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{2.31}$$

$$\overline{\Sigma_{jk}} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)(O_t - \mu_{jk})(O_t - \mu_{jk})'}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{2.32}$$

$$\tag{2.33}$$

with the prime in the denoting vector transpose in (2.33).

Using the above formulas the iterative model adaptation is performed in two steps.

E-Step: First for every state the expected numbers of transitions are computed as well as the expected number of activations of every mixture component of the state.

M-Step: Next using those counts/expectations, the formulas are used to

recompute the new model parameters, maximizing the likelihood of the model given the new observations.

We finally note that this re-estimation procedure can be derived directly by maximizing Baum's auxilary function

$$Q(\lambda, \overline{\lambda}) = \sum_{Q} P(Q|O, \lambda) log[P(O, Q|\overline{\lambda})] \tag{2.34}$$

over $\overline{\lambda}$ using Lagrange multipliers. Alternatively, the model parameters can be optimized with gradient techniques giving similar results.

2.4.3 HMM Model Merging and Compression by Data re-sampling

The easiest way to merge two HMMs into one model, is to generate new data D by sampling from both models and then training a new model on D with standard Baum-Welch training. Similarly, the easiest way to compress a HMM is to re-sample data from the model and then train a new simpler model on the sampled data.

The GMM-HMM Sampling Method

We now describe a sampling method that takes a GMM-HMM and generates a sequence of T states and T observations. This sampling method is used in re-sampling based merging and compression, but also in the computation of sample-based HMM distance metrics. The sampling method starts by selecting the initial state S_0, mapping a random number to one of the states with chances corresponding to the prior state distribution π. Then from state S_n a next state S_{n+1} is randomly chosen according to the the transition chances of going from state S_n to the other states, which are stored in the Transition Matrix. This is repeated until a sequence of T states has been sampled. The next step is to generate T observations for the T sampled states. To do this, for every sampled state first a mixture component must be sampled with for every component a chance equal to the mixture component weight divided by the total component weight in the state. These weights are taken from the mixmatrix m of the GMM-HMM. As a last step for every sampled mixture component an actual output is sampled from the Multivariate Gaussian distribution of that component.

2.5 Alternative Solutions

There exist some interesting alternatives to HMMs for sequence learning, most prominently Recurrent Neural networks. The problem with neural

networks however is that there are no efficient learning algorithms, so convergence is very slow. For recurrent neural network this problem becomes even bigger. Recently Echo State Networks (ESNs) [21] has been proposed as a special alternative that is well suitable for sequence learning, and does not suffer from the normal drawbacks of recurrent neural networks, since it only trains the connections to the final outputs and all other connections are randomly set and kept that way. ESNs are very similar in functioning to Support vector Machines. They first perform an complex nonlinear transformation on the data, and then apply regression to find the best mapping from this new space to the outputs. The non-linear transformation is performed by the reservoir which is a large set of randomly connected states containing cycles. The amount of states in the reservoir is relatively large, so that in a way its representational power is much greater than is necessary to represent all the inputs, and this has the advantage that there is a high chance that the inputs can be separated in a linear way in the new high-dimensional space spanned by the reservoir. There exist some other *reservoir methods* that are very similar in philosophy as ESNs, a complete overview of such methods is given in [31].

Echo state Networks are resolving some of the drawbacks of neural networks, but not all. They are still almost as hard to analyze as recurrent neural networks, and they still generalize just as poorly also. Principal Component Analysis is a tool that can give some insight in the structure of neural networks and how it encodes the concepts it learns, but this remains a very difficult process, and so often the networks are just used as a black box. .

Chapter 3

The state of the Art in Unsupervised Gesture Recognition and Clustering

There are currently different approaches towards Unsupervised gesture recognition, segmentation and clustering. We will review the important recent works in this domain, that are most relevant to our own research. Those are:
1.Incremental, Hierarchical Clustering and learning of a HMM tree
2.Clustering and compression of small HMMs to discover symbolic structure
3.Training of sparse HMMs with a new *Entropic Prior* Baum-Welch variant
4.Hierarchical Hidden Markov Models
5.Hierarchical Hidden Markov Models combined with Bayesian Model Adaptation
6.Simple Segmental Clustering
7.Hidden Markov Model Induction by Bayesian Model Merging

Other significant extensions to the standard HMM include the Parametrized HMM (PHMM) [50], Variable-length HMM (VHMM) [16], Coupled HMM (CHMM) [4], Input-Output HMM (IOHMM) [3], Factorial HMM [17] and HMM Decision Trees (HMDT) [23].

3.1 Incremental HMM Clustering

The first approach towards unsupervised gesture learning is to train HMMs for gesture data that is recorded spread out over time. The idea of this approach is to learn models incrementally, merge models that are very similar, and this way form a tree structure of models. This idea is introduced in *Incremental on-line hierarchical clustering of whole body motion patterns* [28] by Kulic et.al, and in this paper is used for the learning of motion

pattern primitives, which can be used for both motion recognition and motion generation.

The algorithm for the incremental Clustering and Grouping of Observation sequences starts from a new observation sequence and a Hierarchical tree of observations and associated HMMs, and ends with an adapted tree that is adapted to include the new observation.

After a new observation sequence is received, a new HMM is trained for it. The Akaike Information Cirterion (AIC) is used [7] to find the best HMM structure. The trained model is compared with a tree of other models and placed in the deepest tree node that has sufficient similarity with it. Kullback divergence between two HMMs is the used similarity measure. This distance can not be computed analytically, but it can be approximated as

$$D(\lambda_1, \lambda_2) = \frac{1}{T}[logP(O^{(2)}|\lambda_1) - logP(O^{(2)}|\lambda_2)] \tag{3.1}$$

where $O^{(2)}$ is an observation sequence generated by λ_2 and T is the length of the observation sequence. Since this is not symmetric, the average of two intra HMM distances is used to form a symmetric distance

$$D_s = \frac{D(\lambda_1, \lambda_2) + D(\lambda_2, \lambda_1)}{2} \tag{3.2}$$

When a new HMM is added to a cluster, a clustering procedure is invoked on that cluster. Complete link clustering is used to perform the clustering, and a minimum number of cluster elements as well as a maximum within-cluster distance are used to see if a sub-cluster should be formed for the cluster.

For every newly formed cluster, a new group is created containing the within cluster elements. Every group has an associated *group HMM*, which is trained on its associated group elements.

Motion Generation

The group HMMs for the formed tree nodes constitute motion primitive abstractions. Motion trajectories are formed from the group HMMs by the Monte-Carlo like averaging method of Takano. Since motion generation is outside the scope of our research, we refer to *Takano, 2006* [45] and the original paper [28] for more details.

Results

The Algorithm developed by Kulic. et.al was tested on a motion pattern database containing 9 different human motion observation sequences obtained through a motion capture system. Their experiments showed that the

resulting tree is partly dependent on the data presentation order, especially with a low cutoff factor K_{cutoff}, that causes new groups to form more quickly. However, they also showed that the leaf nodes of the resulting tree structure represent the correct segmentation/clustering, regardless of the presentation order. Periodical application of a tree correction algorithm is thus suggested as a way to reposition leaf nodes to the correct branch.

3.2 Data Segmentation by clustering small HMMs

The second approach towards discovering structure in long unlabeled gesture data streams is through data partitioning and training of many small models followed by clustering. This approach is explored by Wang et.al in "Unsupervised Analysis of Human Gestures" [47]. We will now discuss their approach stepwise.

First the data stream is split into many small parts. Wang et.al exploit the fact that a type change of human movement usually causes dips in velocity or abrupt changes in movement direction. The local minima of velocity and local maxima of change in direction are therefore used as segmentation points in their research.

Second, after the data has been partitioned, for every part a HMM model is trained. Then, the set of HMM models of all parts is clustered using hierarchical clustering in combination with a HMM distance metric. In this step all models that are very similar will be put close together in the generated clustering, and can then be considered to be different instances of the model. Similar enough models are then merged, and the merged models are denoted by a symbol.

Now that all parts from the original data stream are labeled with the symbol of their small HMM or merged HMM, we have a higher level description of the data in terms of a symbol sequence. As fourth step then, this symbol sequence is further compressed by finding frequent subsequences in it with an algorithm called COMPRESSIVE [47] which finds the structure within symbol sequences in a Hierarchical way. The final higher-order symbol sequence can give a good description of the data in terms of abstract types. In combination with the symbol data-ranges in the original data, it can also be effectively used for data segmentation.

Wang et.al test their system with an experiment on musical conduction, were a conducting gesture sequence is segmented and then the structure is found. The used sequence contains 5 basic beat-patterns The 8 minutes record they used was partitioned into 163 segments, and for each segment a 5-state Gaussian HMM was trained. After application of COMPRESSIVE

on the clustered HMMs, a set of patterns was extracted, that shows close resemblance to the original primitives. This proves the success of their approach in segmenting continuous human gestures.

3.3 Training of sparse HMMs with a new *Entropic Prior* Baum-Welch variant

In *"Learning concise models of human activity from ambient video via a structure-inducing M-step estimator"* [5] M. Brand introduces a new prior and associated *maximum a posteriori* (MAP) estimator that can be understood as an exact formulation of minimal description length for Bayesian point estimation. The prior provides a method for structure discovery in data and is used to learn a world model from coarse image representations. The theory takes the form of a continuous-output hidden Markov model (HMM). The learning algorithm exploits the entropic prior for fast, simultaneous estimation of model structure and parameters. In contrast to conventionally trained HMMs, entropically trained models are so concise and highly structured that they are interpretable, and can be automatically converted into a flowchart of activities. .

3.3.1 Entropic Prior

Entropic estimation favors parameter values as far as possible from their initial random values. The corresponding prior should imply that parameters which do not reduce uncertainty are improbable.

The used prior is conjugate to the used multinomial distribution, meaning the resulting posterior probability distribution s in the same mathematical family of functions as the multinomial. With scarce evidence the entropic distribution favors stronger odds (parameter values near the extrema), but with increasing evidence converges to "fair" odds for the parameters and is therefore consistent. The opposite behavior is obtained from a Dirichlet prior, which favors weaker odds when data is scarce.

3.3.2 MAP estimator

As a next step, Brand derives a MAP estimator for the entropic prior. Manipulation of the negative log posterior probability of data plus model, which is minimized by the algorithm, allows for understanding the MAP estimator in terms of entropy. The log posterior can be rewritten as:

$$H(\theta) + D(\omega||\theta) + H(\omega) \tag{3.3}$$

which is the sum of parameter entropy, cross entropy between data and parameters and data entropy. When data entropy is fixed, the MAP estimate

minimizes the sum of the parameter entropy and the cross-entropy between the parameters and the datas sufficient statistics. Equivalently, it minimizes coding length. Brand shows also how the model can be simplified, such that the data's sufficient statistics change and $H(\omega)$ declines as well.

3.3.3 Entropic HMM training

In entropic estimation of HMM transition probabilities, the normal E-step of the Baum-Welch algorithm is followed, calculating the probability mass for each transition to be used as evidence. For the M-step, the MAP estimator computes new parameter values from a vector of evidence for each kind of transition out of a single state.
In recursive estimation (e.g., EM), the entropic estimator drives weakly supported parameters toward zero, concentrating evidence on surviving parameters until their estimates converge to near the ML estimate, at which point the algorithm terminates.

Brand shows in [6] that remaining HMM parameters close to zero can be deleted with no loss of probability. Trimming bumps the model out of a local probability maximum and allows further training in a lower-dimensional and possibly smoother parameter subspace. A similar test licenses state deletion.

3.3.4 Results

The Entropic HMM training algorithm was tested on half an hour video of ambient office activity, using stripe and blob features extracted from the foreground pixels to model the human subject. The HMM model produced by Entropic HMM training has a very strong structure and is readable as a flowchart of office activities. This is in strong contrast with the model generated by normal Baum-Welch training, which fails to discover a structured model (see Figure **??**).

3.4 Hierarchical Hidden Markov Models

In "The Hierarchical Hidden Markov Model: Analysis and Applications" [15], Fine et.al introduce a recursive hierarchical generalization of hidden Markov models. The work was motivated by complex multi-scale structure appearing in natural sequences such as language, handwriting and speech. It has already found its application to *Information Extraction* in the domain of biomedical articles [41] as well.

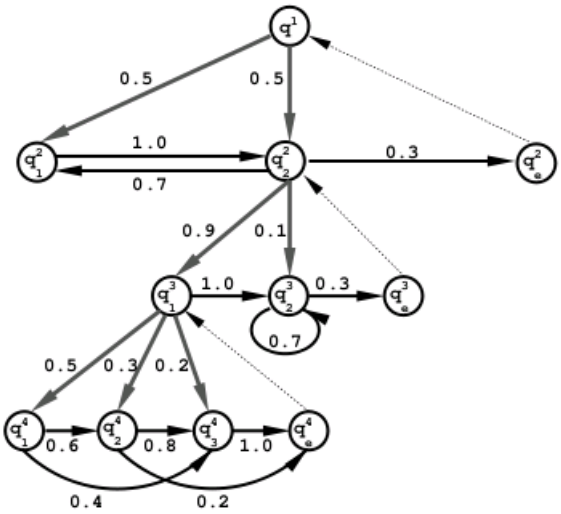


Figure 3.1: An illustration of an HMM of four levels. Gray and black edges respectively denote vertical and horizontal transitions. Dashed thin edges denote (forced) returns from the end state of each level to the level's parent state. For simplicity, the production states are omitted from the figure.

3.4.1 Model description

Hierarchical Hidden Markov Models(HHMMs) generalize the standard HMMs by making each of the hidden states an autonomous probabilistic model on its own, that is, each state is an HHMM as well. Therefore, the states of an HHMM emit sequences rather than a single symbol. An HHMM generates sequences by a recursive activation of one of the sub-states of a state. This process of recursive activations ends when we reach a special state which we term a pro- duction state. The production states are the only states which actually emit output symbols through the usual HMM state output mechanism: an output symbol emitted in a production state is chosen according to a probability distribution over the set of output symbols. A HHMM has two kind of state transitions. Horizontal transitions move from one state to another state at the same level, vertical transitions move control to another (typically deeper) level in the hierarchy. Each level in the hierarchy has a terminal state, which returns control to the parent state that activated the level. Generation of the observation sequence completes when the control of all recursive activations is returned to the root state. Figure 3.1 shows a HHMM with four layers.

3.4.2 Inference and Learning

Similar as for normal HMMs, for HHMMs three fundamental problems arise in applications that use HHMMs :
1. Calculating the likelihood of a sequence
2. Finding the most probable state sequence
3. Estimating the parameters of a model
Solutions for the above problems are more complex for HHMMs as for normal HMMs, but the ideas are the same, only they are now generalized for the hierarchical case. The algorithms used rely on dynamic programming for computing the probabilities and determining the model parameters, just as is the case for normal HMMs. Computational Complexity therefore remains acceptable as well. Computing the likelihood of a sequence and finding the most probable state sequence both have a complexity of $O(N \times T^3)$, with N the total number of states and T the length of the observation sequence.

3.4.3 Applications

Fine et.al describe two applications of HHMM. In the first application they construct a hierarchical representation of natural English text. They compared a shallow HHMM consisting of two levels, with an unbalanced HHMM of three levels with a variable number of sub-states for the internal states and production states at all levels. After training on 500 sentences of classical English stories, the shallow HMM much resembled the normal HMM trained on the same data. The unbalanced HHMM in contrast revealed strong structure in the model. Different sub-states had strongly varying distributions, and accounted for different hardly overlapping sets of strings. Furthermore, a multi-scale behavior was found. The deepest production states produced sequences corresponding roughly to phonetic units, while the second layer produced frequent words. The top layer had a output distribution corresponding to the sentence scale.
In the second application HHMMs were used to perform unsupervised learning of cursive handwriting. Based on a dynamic coding scheme by Singer et.al [40], they trained HHMMs for different words. Then as a next step they applied generalized Viterbi algorithm to perform a multi-scale segmentation of the motor control sequences for those words. In this way 'natural' units constituting the sequences are extracted. This has advantages over manual construction of such units, namely not requiring manual segmentation and taking temporal interaction into account .

3.5 Hierarchical Hidden Markov Models combined with Bayesian Model Adaptation

In "Unsupervised Discovery of Multilevel Statistical Video Structures using Hierarchical Hidden Markov Models" [52] Xie et.al combine HHMMs with a Bayesian method for model adaptation to perform a completely unsupervised structure discovery on and segmentation of soccer video. They use a version of the general Markov Chain Monte Carlo (MCMC) [1] method to perform the optimization of the model structure. Their algorithm automatically finds descriptions of high level structures, and furthermore performs slightly better than a supervised HMM approach that exploits domain knowledge.

3.6 Simple Segmental Clustering

In "Unsupervised Clustering of Ambulatory Audio and Video" [12] Clarkson and Pentland introduce a very simple yet effective way of incrementally clustering video data based on a variation of the Segmental K-Means Algorithm [38].

Data of a person performing normal daily activities in the city at daytime were recorded using a mounted camera and microphone. To find only obvious events and prevent sensitivity to minor fluctuations in the data, crude audio and visual Features were extracted.

The Segmental K-means clustering procedure they used is as follows:

1. Given: N , the number of models, T the number of samples allocated to a state, S, the number of states per model, f the expected rate of class changes.
2. Initialization: Select N segments of the time series each of length $T*S$, spaced approximately l/f apart. Initialize each of the N models with a segment, using linear state segmentation.
3. Segmentation: Compile the N current models into a fully-connected grammar. A nonzero transition connects the final state of every model to the initial state of every model. Using this network,re-segment the cluster membership for each model.
4. Training: Estimate the new model parameters using the Forward-Backward algorithm on the segments from step 3. Iterate on the current segmentation until the models converge and then go back to step 3 to re-segment. Repeat steps 3 and 4 until the segmentation converges.

By varying the number of models T the time series can be modeled at different time scales. To capture the hierarchical structure of events, which can not be modeled by just changing T, the authors used the activation of HMMs trained with a small T as features for higher order HMMs that model

scenes. The system was tested by comparing the most likely derived models with a hand-made ground truth of scene labels.

3.7 Hidden Markov Model Induction by Bayesian Model Merging

The work by Stolcke and Omohunro on HMM merging [43, 44] describes a technique for learning both the number of states and the topology of Hidden Markov Models from examples. Their approach is based on the idea of first simply storing examples, then merging their corresponding specific models to form complex and generalizing models as more data becomes available. The theoretical foundation of their research is that the *posterior probability* $P(M|X)$ of the model M given the observed data X and the prior $P(M)$ should be maximized. This posterior can be defined using Bayes Law as:

$$P(M|X) = \frac{P(M)P(X|M)}{P(X)} \tag{3.4}$$

Therefore the most likely model M_{MAP} is:

$$M_{MAP} = arg\,max_M \frac{P(M)P(X|M)}{P(X)} = arg\,max_M P(M)P(X|M)(= P(M,X)) \tag{3.5}$$

This implies that often a different model may be preferable, if this gives an increase in the prior, while giving no or only a smaller decrease in the conditional probability of the data given the model. Indeed, the intuitive function of the Prior is to assign more likelihood to simpler models, with a smaller amount of states and transitions. The maximization in (3.5) is equivalent to minimizing

$$-logP(M,X) = -logP(M) - logP(X|M) \tag{3.6}$$

This in turn can be interpreted as minimizing the description length of the data X plus the underlying coding model M, with $-logP(M,X)$ being the optimal coding length of the model under the prior, whereas $-logP(X|M)$ corresponds to the optimal code for the data using M as a probabilistic model. therefore finding the Bayes optimum for the model is homologous to finding the Minimal Description Length (MDL) optimum.

3.7.1 Model Priors

A HMM can be described in two stages, first specifying the *model structure* and next the parameter values given this structure. Therefore a model prior $P(M)$ can be written as:

$$P(M) = P(M_S)P(\theta_M|P_S) \tag{3.7}$$

with $P(M_S)$ giving the Prior for the model structure and $P(M_S)P(\theta_M|P_S)$ the probability of the parameters given the chosen structure. The approach of the authors is to compose a prior for both the structure and the parameters of a HMM as a product of independent priors for transitions and emissions of states along with a global factor. The global prior for a model then becomes a product of priors for the state and the global prior. While this independence assumption about the parameters of different states is a simplification, it should not introduce a systematic bias towards a particular model structure. The big advantage it gives however, is that it greatly simplifies the computation of global model posteriors, which forms the core of the merging approach.

The authors introduce a global priorλ that determines the amount of weight given to the data and consequently the amount of generalization performed by the algorithm. Taking the logarithm of P(M,X) they obtain

$$logP(M, X) = logP(M) + logP(X|M)$$

as the quantity to be maximized. This can be modified by a *prior weight* λ to get:

$$\lambda logP(M) + logP(X|M) \tag{3.8}$$

with the effect of assigning more weight to the model prior for bigger λ thus increasing the amount of generalization, or more weight to the data for smaller λ thereby limiting the amount of generalization.

3.7.2 Parameter Priors

We now discuss the priors used for the individual parameters of emissions and state transitions. For discrete output HMMS, model parameters can be described entirely as parameters for a collection of multinomial distributions. Therefore priors for such distributions can be used as priors for discrete HMM parameters. In the multinomial representation transitions correspond to finite probabilistic choices for next states and similarly emissions correspond to choices among output symbols. Let n be the number of choices in a multinomial and $\theta = (\theta_1, ..., \theta_n)$ be the parameters associated with those choices. A standard prior for multinomials, used by the authors is the *Dirichlet Distribution*:

$$P(\theta) = \frac{1}{B(\alpha_1, ..., \alpha_n)} \prod_{i=1}^{n} \theta_i^{\alpha_i - 1} \tag{3.9}$$

Here $\alpha_i, ..., \alpha_n$ are the prior weights: the the prior expectation of θ_i is $\frac{\alpha_i}{\alpha_0}$ with $\alpha_o = \sum_i \alpha_i$ the total prior weight. The normalizing constant $B(\alpha_1, ..., \alpha_n)$

is the n-dimensional Beta function

$$B(\alpha_1, ..., \alpha_n) = \frac{\Gamma(\alpha_1)...\Gamma(\alpha_n)}{\Gamma(\alpha_1) + ... + \Gamma(\alpha_n)} \tag{3.10}$$

with

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t}\, dt \tag{3.11}$$

being the Gamma function, which is an extension of the factorial function to real and complex numbers. The *Dirichlet Prior* expresses the bias towards certain choices in the multinomial distribution, by choosing α_i increasingly bigger than 1, an increasing bias towards the i-th choice in the multinomial is expressed. Another way to formulate this, is that for $\alpha_i > 1$ the prior adds $\alpha_i - 1$ virtual samples of choice i to the likelihood expression. The *Dirichlet Prior* has the advantage of mathematical simplicity. It is a *conjugate prior* meaning that it has the same functional form as the likelihood function of the multinomial for which it is defined. As a result the posterior, which is the integral over the product of the multinomial and the prior, has a closed form solution. By using independent priors for all parameters based on the *Dirichlet Distribution*, the posterior for a given model structure can then be effectively computed in closed form as well.

3.7.3 The Algorithm

The HMM merging process starts with the most specific model consistent with the training data and generalizes by successively merging states. The choice of states to merge and the moment to stop the merging process are both guided by a Bayesian posterior probability. The Algorithm then is as follows:

Best-first merging(batch version)
A. Build the initial, maximum-likelihood model M_0 from the dataset X.
B. Let i := 0.Loop:

1.Compute a set of candidate merges K among the states of model M_i.
2.For each candidate $k \epsilon K$ compute the merged model
$k(M_i)$, and its posterior probability $P(k(M_i)|X)$.
3. Let k^* be the merge that maximizes $P(k(M_i)|X)$. Then let $M_{i+1} := k^*(M_i)$.
4. If $P(M_{i+1}|X) < P(M_i|X)$, break the loop
5. Let $i := i + 1$

The number of Potential merges $|K|$ is the main determining factor for the computational cost of the merge algorithm. Since $|K| = O(|Q|^2)$, with $|Q|$ the number of states, and since $|Q|$ grows linearly with the total size of the samples processed at a time, the batch version is only feasible

for small amounts of data. As an alternative to batch merging, a very similar incremental version of the merging algorithm is proposed as well. This algorithm contains an extra loop to incorporate new data samples into the new model, after which the merging loop is executed until convergence as in the batch version. The incremental algorithm has the advantage of not requiring all samples at once. The division of the data into small samples makes the computational cost linear in the total data size instead of quadratic as is the case for the batch version. This makes it a good solution for bigger amounts of data.

3.7.4 Evaluation

The merge algorithm was compared with the Baum-Welch method of estimating fixed-size models. In principle Baum-Welch only finds parameter values and no model structure, but by pruning transitions with very low values it can be used to derive model structure as well, which is the approach taken by the authors. The algorithms were tested in three different scenarios. In the first scenario, simple artificial languages and artificial training samples were used to provide a proof of concept for the method. The target model, that defines the language is used to sample a set of examples, and next those samples are used to train a model using the merge algorithm or standard Baum-Welch. The trained models are evaluated in three ways. The first evaluation score is the Kullback-Divergence between the target model and the trained model. If the trained model and target model assign equal probabilities to the same samples the Kullback-Divergence is small. When the probabilities they assign are more dissimilar, the Kullback-divergence becomes bigger, meaning the models are more different. The second score measures what fraction of samples generated from the target model can be parsed by the trained model, which in information extraction terms can be seen as a kind of "Recall". The third score measures the opposite: what fraction of samples generated by the trained model are parsable by the target model. This score is a kind of "Precision" measure. Results showed that the merging procedure was successful in finding the target model, whereas Baum-Welch training produced inconsistent results that were highly dependent on the initial parameter settings.

In the second scenario, the algorithm was tested on the TIMIT (Texas Instruments - MIT) database of hand-labeled speech samples. This database contains acoustic data segmented by words and aligned with discrete labels from an alphabet of 62 phones. In the experiments the acoustic data was ignored, and the database was used as a collection of string samples over a discrete alphabet. In the tests, models were trained on the training portion of the data and then generalization was tested on the testing part of the data. To overcome problems with zero probabilities assigned to samples, a mixture model of the HMM and a simpler bi-gram model was used in

the evaluation of test samples. The results showed a small but significant difference in favor of merging when comparing the best model merging result with the best Baum-Welch result. Another big advantage of model merging over Baum-Welch training revealed by the experiment is that it generates much simpler and more compact model.
In the last scenario the merging algorithm was applied to a speaker independent spontaneous continuous speech understanding system. Comparison of the multiple-pronunciation system with an identical system using only one phone sequence per word showed a significant reduction in both word-level error rate and errors on the semantic interpretation level.

Part I

New Research

Chapter 4

Segmenting, Compressing and Merging HMMs

In the following section we discuss our first method for HMM simplification. We take a new approach where we analyze and use the structure within a classical HMM directly. By finding strongly associated states within the HMM that are part of *Cycles* or *Trajectories* we are able to partition those states into meaningful groups. Combined with labeled HMMs those groups can be directly used to perform classification.

4.1 Hidden Markov Model Implicit Structure Analysis

A HMM Transition Matrix encodes a fully connected directed and weighted graph of transition chances between states. Given a limited number of states, different choices for the transition chances still allow an infinite amount of different model structures. However, if we only consider strong structures , defined by the fact that for some states a particular transition is more likely than all others, then the type of distinguishable structures is small. In fact all such structures are then expressible in terms of two atomic structure primitives, namely *Cycles* and *Trajectories*. *Cycles* are ordered sequences of states beginning and ending in the same state. They have the property that starting from every state in the cycle the chance is high of returning to that same state by stepwise visiting all states in the cycle's ordered state sequence. *Trajectories* are similar, except that they don't end in the same state – they may be seen as a weaker form of *Cycles*.

4.1.1 Cycle extraction

Our algorithms are concerned with the extraction of *Cycles* and *Trajectories* in the HMM using the Transition Matrix T.
Cycles are extracted in three steps. We compute for every pair of states S_A and S_B the likelihood of the most probable path that goes from S_A to S_B and then back from S_B t S_A. This defines $CycleChance_T(S_A, S_B)$:

$$
\begin{aligned}
&CycleChance_T(S_A, S_B) = \\
&P(ViterbiPath(S_A, S_B)) \times P(ViterbiPath(S_B, S_A)) \\
&= max_{S_i \epsilon States} P([S_A, \ldots, S_B]) \times P([S_B, \ldots, S_A])
\end{aligned}
$$

Since all transition chances are smaller than 1, it follows that the most likely sub-path $[S_A, \ldots, S_B]$ and $[S_B, \ldots, S_A]$will not contain duplicate states . We define

$$CycleDistance(S_A, S_B) = \frac{1}{CycleChance(S_A, S_B)} - 1$$

. The cycle chance and cycle distance are dependent on the Transition Matrix used to compute them. We use a modified version T_m of the original Transition Matrix, in order to achieve some time-scale invariance and guarantee minimal transition chances for retrieved cycles. To modify the Transition Matrix, we go through the following steps. We first set diagonal elements to zero and then re-normalize. This eliminates the influence of within-state cycles to the distance, giving a better relative time scale invariance. As a next step, we set all transitions below a threshold $Threshold_{CycleTransition}$ to a very small number N_{small}. When we later form cycles from states based on the cycle distance, this guarantees a minimum transition chance above the threshold for all between state transitions part of the cycle.
We apply the Viterbi algorithm on the thus Modified Transition Matrix to compute the cycle distance between all pairs of states.

Next, we use *Complete Link Clustering* on the Cycle Distance Matrix, finding groups of states likely to be on the same cycle. Complete Link Clustering recursively groups elements bottom up. At every iteration it merges the two groups G_1 and G_2t which give the smallest new maximum within group distance. Formally:

$$arg\ min_{G_1,G_2}\ max_{e_A \epsilon G_1, e_B \epsilon G_2}(dist(e_A, e_B))$$

Let the result be a tree-structure T with the leave nodes L of the tree containing the single states, and every non-leave node i containing a list of states rooted at that node and a value *distance* which is the maximum between-state distance over all states merged in that node. Then we seek the largest possible state groups that satisfy $distance < cycleGroupThresh$. By setting $cycleGroupThresh = \frac{1}{Threshold_{CycleTransition}}$, the retrieved groups are

guaranteed to have for all transitions that make up their cycle a transition chance $P_{transition} > Threshold_{CycleTransition}$ We find those state groups by starting at the root of the tree, and recursively descending towards the children nodes until we either:
1. Find a node with *distance* < *CycleThresh*, whose state set is than added to the set of cycles.
2. We reach a leave node and stop.

4.1.2 Trajectory extraction

After *Cycles* have been found, a is new Transition Matrix is extracted from the modified Transition Matrix T_m used in the previous step. This matrix is created by deleting all rows and columns from T_m that correspond to states belonging to cycles. The new sub-Transition Matrix $TransMat_{rem}$ is used to find trajectories. The algorithm for finding *Trajectories* is simple and intuitive. We start from the set of singleton states S. For every state $s_n^i \epsilon S$ a trajectory is repeatedly extended by appending the most likely next state s_{n+1}^i as long as $P(s_{n+1}^i|s_n^i) > Threshold_{TrajectoryTransition}$. In this way we find a set of possibly overlapping trajectories T_o.
The process by which we form trajectories guarantees that possible overlap is always on the tail of the trajectories. Thus for a trajectory $[s_1, \ldots, s_n]$ overlap is always on some sub-trajectory $[s_m, \ldots, s_n]$, with $1 \leq m \leq n$, that ends with the last state. This property is exploited by an algorithm that partitions the overlapping trajectories in a set of longest non-overlapping trajectories. To do this we start with the full set of overlapping trajectories $T_{rem} = T_o$ and an empty set of partitioned trajectories T_p. We then pick a trajectory $t_x \epsilon T_{rem}$, and find the set $T_{overlap}(t_x)$ of overlapping trajectories, having a sequence of last states overlapping with t_x. If $T_{overlap}(t_x)$ is empty, it means that t_x did not overlap with anything and can be removed from T_{rem} and added to T_p. If not, we take the longest tail *shared by all trajectories in the overlapping set*, add this tail to T_p, and remove it from t_x and all trajectories in $T_{overlap}(t_x)$. We continue taking trajectories from T_{rem} until it becomes empty, and at this point T_p will be containing the set of non-overlapping trajectories. Some non-overlapping trajectories in the final set will consist of only one state and are therefore not really trajectories at all but rather *Singleton states*.

All states are now assigned to be part of a particular cycle, trajectory or to be singleton. Groups of states corresponding to *Cycles*, *Trajectories* and *Singleton states* are returned and sub-HMMS for them are extracted from the original HMM. The complete algorithm that performs all steps, first the Cycle extraction and then the Trajectory extraction, is called *HISAA* - short for *Hidden Markov Model Implicit Structure Analysis Algorithm*.

4.1.3 Setting the Thresholds

One important issue in the use of the algorithms is how to set the thresholds $Threshold_{CycleTransition}$ and $Threshold_{TrajectoryTransition}$. An important constraint is that it must always be the case that $Threshold_{CycleTransition} \leq Threshold_{TrajectoryTransition}$ to prevent that state sets that are not accepted as cycles can later be accepted as trajectories. This would lead to algorithmic failure because a cycle contains many trajectories, one for each state in the the cycle's state set. From a theoretical point of view it is also clear that if we don't accept a certain cycle because we consider (some of) the involved transitions to weak, then we also shouldn't accept a trajectory for the same states and thus relying on the same transitions. This intuition is reflected by the constraint on the thresholds.
Apart from this constraint however, we are free to choose the thresholds as we like, and unfortunately it is hard to predict beforehand what the best thresholds are, because this depends on the HMM that is to be segmented as well as on the application. High thresholds will lead to small state groups, or even only singletons if there are no strongly connected states in the HMM. On the other hand, to low thresholds will lump all states together into one group, which is also meaningless. Intuitive values for the thresholds lie somewhere in the range around 0.5, meaning that every transition of the cycle/trajectory must have at least 50% chance of occurring. While it is clear that neither very low nor very high thresholds will work, a range of thresholds in the middle might all work for a certain problem, and experiments have to be done to find the best values. This was also reflected by both our initial experiments and our later extensive tests of the algorithm.

4.2 HMM compression and HMM merging

In the following paragraphs we cover our second approach towards HMM simplification, which is HMM compression and HMM merging. Compressing and merging Models are important issues in computer science nowadays, with the increasing amount of data and facilities to do computing in a distributed manner. In gesture and motion recognition research model merging can be required to merge new data with existing models and forgo the need to keep all data and perform all computation in a centralized way. The advantage of merging models is that it decreases the number of model parameters to be kept as well as the runtime for recognition by the factor

$$\frac{\text{Number Original models}}{\text{Models remaining after merging}}$$

(assuming the merged models have the same number of parameters as the original models). Furthermore model merging has also important applications to the generalization of different demonstrations of a gesture to

a single model, which is an important task for human robot interaction in the domain of Imitation Learning. Of course the result of merging models could also be achieved by simply keeping all data and then training a new model from this data. This however puts high demands on the storage capacity of the system. Furthermore it comes at the price of a higher computational cost, since the computational cost of HMM training increases linearly with the size of the used training set, as discussed at the end of this chapter. With an incremental or distributed approach to gesture learning exploiting model merging, the total computational cost may eventually be not smaller than it is for batch learning. The big advantage however is that the work in training the models can be divided much simpler over time, people and computers than can be done in a centralized approach that requires learning the models on the full data in one shot. Similarly model merging greatly facilitates the adaptation or customization of gesture models to new data and new users in case the starting model is still largely appropriate.

In [28] the authors perform incremental hierarchical learning of gestures, using HMM distance [14, 38] to determine whether new data can be merged with an existing model or should be put in a new model.
Compressing and merging of models can be approached in two different manners. Either one can try to regenerate data from the model(s) to be compressed/merged and learn a new model on this data, or one can try to compress/merge the model(s) directly from the parameters. The second approach is typically much more efficient, but may not always achieve the same performance as the first one. For Hidden Markov Models [38], compression and merging are mainly done by data re-sampling so far in the literature. This method consists of generating new data by sampling from the model(s), and then using this data to train a new HMM model in the normal way using the Baum-Welch algorithm. While this simple method gives good results, it comes at a relatively high computational cost. In this paper we explore the other alternative, which is to Compress/Merge HMMs directly.

In "Hierarchical Clustering of a Mixture Model" [18] a new algorithm (to which we refer as *GMM Clustering*) was presented for direct clustering and compression Gaussian Mixture Models, without the need for data re-generation.

Inspired by this work we created a new HMM compression and merging algorithm that uses GMM clustering to compress and merge GMM-HMMs without any data re-sampling or model training from the scratch. This makes the HMM compression/merging computationally more efficient. Especially for HMMs with a large amount of states, the advantage of avoiding Baum-Welch re-training can be considerable. The question is if direct model merging or compression works just as well as the current method that relies on data re-sampling. If not, it is unclear how big

the loss of performance will be, and if this can be still acceptable given the computational gain. To answer this question, we did several test on the Interactplay gesture database, comparing the performance of direct model compression/merging with re-sampling compression/merging and direct training on all data.

4.2.1 HMM Compression

While standard HMM compression methods rely on re-sampling, our method compresses models directly from their parameters.

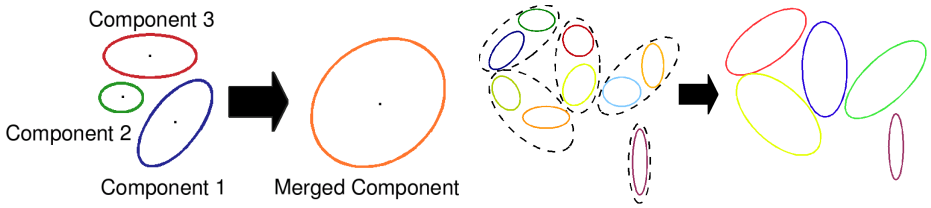
The essence of the algorithm

The intuition behind our algorithms is to compress Gaussian Mixture Model HMMs (GMM-HMMs) based on only the spatial information as provided by the GMM part. This means considering the set of all mixture components for of all states, and using this *spatial* view of the HMM as a basis for compression. In this view it is easy to find mixture components or states (which are sets of mixture components) that are spatially close. Since most signals are continuous in space it is reasonable to expect that states that are spatially close are temporarily close as well, so they are good candidates for merging when compressing the HMM. Based on spatial proximity our algorithm decides which states are to be merged. After that, constructing the new states and new Transition Matrix comes down to a straightforward form of addition of the merged parts.

The algorithm stepwise

We now give a more detailed description of the steps involved in the algorithm. For those who want to implement the algorithm, we also provide pseudo code in the Appendix, closely matching the following discussion.

The compression algorithm starts by compressing the multiple mixture components per state into just one Gaussian (see 4.1 (a)). .



(a) Step 1: Merge Components (b) Step 2: Merge States

Figure 4.1: Step 1 and 2 of HMM compression algorithm

This state-Gaussian is the weighted sum of the mixtures for the state, as defined by the following formulas:

$$w_j = \sum_{i \epsilon \pi(j)} w_j^i \tag{4.1}$$

$$\widehat{\mu}_j = \frac{\sum_{i \epsilon \pi(j)} w_j^i \mu_j^i}{\widehat{w}_j} \tag{4.2}$$

$$\widehat{\Sigma}_j = \frac{\sum_{i \epsilon \pi(j)} w_j^i (\Sigma_j^i + (\mu_j^i - \widehat{\mu}_j)(\mu_j^i - \widehat{\mu}_j)^T)}{\widehat{w}_j} \tag{4.3}$$

In these formulas $i \epsilon \pi(j)$ is the set of Mixture components of state j.

Secondly the algorithm uses GMM-clustering [18, 2] to compress the state-Gaussians of the previous step into a smaller set of Gaussians, corresponding to a smaller set of states (see 4.1 (b)).. GMM-clustering solves the problem of compressing a Gaussian Mixture density

$$f(y) = \sum_{i=1}^{k} \alpha_i N(y; \mu_i, \Sigma_i) = \sum_{i=1}^{k} \alpha_i f_i(y) \tag{4.4}$$

into a reduced mixture of $m < k$ components. This can be formalized as finding the "closest" reduced mixture $\widehat{g}$ under some distance metric. If we take as distance metric

$$d(f, g) = \sum_{i=1}^{k} \alpha_i min_{k=1}^{m} KL(f_i || g_j). \tag{4.5}$$

it can be proven that the optimal $\widehat{g}$ is a Mixture of Gaussians obtained from grouping the components of f into clusters and collapsing all Gaussian within a cluster in a single Gaussian. While there is no closed form solution for finding $\widehat{g}$, an iterative procedure exists that is much like a higher order version of the K-means clustering algorithm. Normal K-means iteratively assigns points to spherical clusters and then recomputes the clusters based on the new assignments, until convergence. GMM-clustering works not on the level of points but on the level of entire mixture components. At initialization a set of mixture components (Gaussians) is taken and every component is assigned to one of a smaller number of new components. Then this new components are computed as the weighted sum of original components assigned to them, using equations (4.1),(4.2), (4.3). After this initialization, the algorithm iterates through a *Regroup* and *Refit* step just as in normal k-means.

$$\pi^g = arg\ min_\pi d(f, g, \pi) \qquad (REGROUP) \tag{4.6}$$

$$g^\pi = arg\ min_g d(f, g, \pi) \qquad (REFIT) \tag{4.7}$$

In the *REGROUP* step, every original component is assigned to the closest new component, using the Kullback Divergence between two components as a distance metric. The Kullback Divergence is a measure of how similar distributions are, and can be computed analytically for two Gaussians as:

$$KL(N(\mu_1, \Sigma_1), N(\mu_2, \Sigma_2)) = \tag{4.8}$$

$$\frac{1}{2}(log\frac{|\Sigma_2|}{|\Sigma_1|} + Tr(\Sigma_2^{-1}\Sigma_1) + \tag{4.9}$$

$$(\mu_1 - \mu_2)^T\Sigma_2^{-1}(\mu_1 - \mu_2) - d \tag{4.10}$$

Next, in the *REFIT* step the new components are recomputed based on the new set of components that is assigned to them in the *REGROUP* step. We use equations (4.1),(4.2), (4.3) again for this computation.

The GMM-clustering eventually converges, when the component assignments no longer change, which typically happens within a few iterations. The final assignments are returned in an *assignment Matrix*, which gives for every original state the index of the new state to which it is mapped

Using the *assignment Matrix* and the mixture components from the original HMM, we collect for each new state all mixture components that are to be combined in it. Those combined components are then compressed into a smaller set of only *NumberOfMixtures* components using GMM-clustering again, while simultaneously producing a new *Mix-Matrix* by taking

$$weight(newC) = \sum_{c\ :\ mapping(c)=newC)} mixtWeight(c) \times prior(state(c))$$

as the weight of the newly formed mixture components.

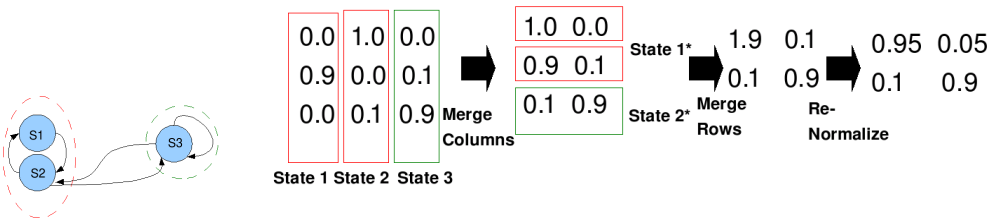
As a final step, a new Transition Matrix and set of Priors is produced. For this we first sum all columns in the old Transition Matrix corresponding to states that are merged together. Then, taking the result of this operation, we do the same for the rows (see Figure 4.2).

Summing on the columns means that we sum the probabilities of mapping to states that are merged in the new model. Summing on the rows averages the transition chances for moving to the next state, of those states that are merged in the new model. As a last step, we normalize the so acquired new Transition Matrix so that the probabilities on every row sum to one. Finally, to compute the priors for the new states we simply sum the priors of the old states that map to those states.

4.2.2 HMM Merging

The algorithm

The merging algorithm is simply a small extension of the compression algorithm. First,if necessary, the states of the two merged HMMs are again



(a) Simple Compressed HMM

(b) Adaptation Transition Matrix

Figure 4.2: The simple HMM of three states is compressed by merging states S1 and S2 (left). This corresponds to first summing the columns of S1 and S2 in the Transition Matrix and next their rows (right)

reduced to contain only one mixture component per state. Next, the States, Transition Matrices, Mixture Matrices and Priors for the two (reduced) merged HMMs are appended to form a new big HMM (see 4.3).

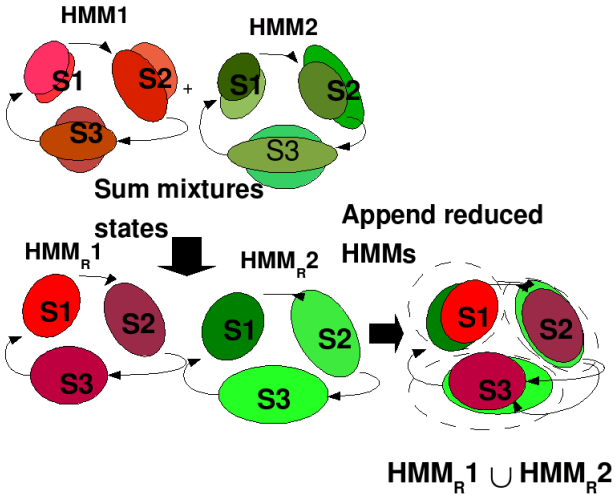


Figure 4.3: Append HMMs: First reduce HMMs to have one component per state, then append resulting reduced states.

Appending States , Mixture Matrices and Priors boils down to simple Matrix concatenation. We just need to take care that we concatenate on the right dimension, so that we import the parts of the second HMM as new states and not accidentally add them to extend the states of the first HMM. Appending Transition matrices is only slightly more complicated. Given are Transition Matrices T_1 with size $n \times n$ and T_2 with size $m \times m$ belonging to the first and second HMM respectively. To append them, we take T_1 and

then paste the T_2 at the lower right corner below it, so that it fills positions $(n+1 : n+m, n+1 : n+m)$ in the newly formed Matrix. The blocks $(1 : n, n+1 : n+m)$ and $(n+1 : n+m, 1 : n)$ of the new Matrix are filled with zeros (see.4.4).

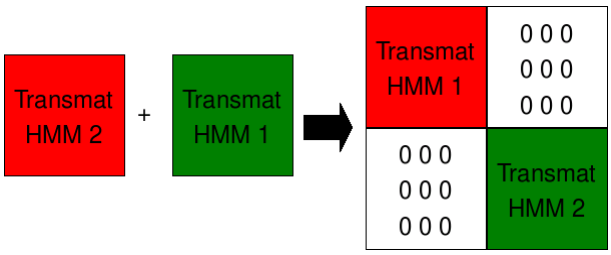


Figure 4.4: Original Data Subgroup Classification results

After all structures of the two merged HMMs have been properly appended to form a new big HMM, this big HMM is compressed with the HMM compression algorithm. This automatically takes care of Merging the states and fixing the Transition Matrix and other parameters, and thus finishes the merging operation.

The importance of good Initialization

One important detail for the merging is the way we initialize the GMM-clustering in the second step of the HMM-Compression algorithm. Our current solution is to just initialize the new states with the states of only one of the two merged HMMs. During the iterations of the algorithm, the states of the other HMM will then be automatically combined with the closest states of this first HMM. This assumes that both merged models have a more or less equal amount of states, and that the produced model will have an amount of states equal to that of the first of the merged models. Other initialization schemes are possible, but should assure that for every state of both merged HMMs there is a new State that is reasonably close. Provided that the merged models are reasonably close, our simple initialization scheme is effective in achieving this most important goal.

Combining the two merging methods

It would be interesting if we could find a computationally cheap yet effective method to guess the quality of a model formed by direct model merging. We could then use direct model merging in most cases and only resort to the more expensive method of re-sampling followed by training of a new model if this is strictly necessary. A simple and intuitive idea to estimate the difficulty of merging is to use the HMM distance between the two HMMs to be merged

as a metric, were higher distances correspond to a more difficult merging cases. The HMM distance between two HMMs λ_1 and λ_2 is computed as

$$D(\lambda_1, \lambda_2) = \frac{1}{T}[logP(O^{(2)}|\lambda_1) - logP(O^{(2)}|\lambda_2)] \tag{4.11}$$

where $O^{(2)}$ is an observation sequence generated by λ_2 and T is the length of the observation sequence. Since this is not symmetric, the average of two intra HMM distances is used to form a symmetric distance

$$D_s = \frac{D(\lambda_1, \lambda_2) + D(\lambda_2, \lambda_1)}{2} \tag{4.12}$$

We implemented this idea in a hybrid algorithm, that allows re-sampling for the most difficult cases and uses our merging algorithm for all other cases.

4.2.3 Computational Complexity Analysis

In this paragraph we shortly compare the computational complexity of our algorithm with the complexity of training a new model. A more complete discussion is given in the Appendix. A new model is trained using either the original data or data that is re-sampled from the model that must be compressed. HMM training is done by the Baum-Welch algorithm, so we must analyze the complexity of this algorithm as well. It turns out that the computational complexity of Baum-Welch is $O(3N^2T + NT + NTMD)$ for continuous HMMs, where T is the number of observations, M is the number of Gaussian components in the mixtures used to represent the output distributions in continuous HMMs, and $D = \sum_{i=1}^{R} d_i$ with d_i the dimension of the data stream i the total number of Gaussian components of data vectors. Most noticeable here is the linear dependency on the amount of training data. For our HMM compression algorithm it can be shown that the computational complexity is $O(N \times M \times D^2)$., with for N and M in this formula the total number of mixture components in the original and compressed HMM, and D the dimensionality of the mixture components. Notice that in this formula there is no dependence upon the size of a dataset. Hence we see the gain of our compression algorithm over generating a new model based on original or re-sampled data. Training a new model has complexity linear dependent on the amount of data used. To get good performance we may well require a re-generated dataset that has a size even much larger than the original dataset, so that the computational cost of compression by re-sampling will be much larger than that of using the *direct compression algorithm* which is independent of data size.

4.3 Implementation

4.3.1 HMM Libraries

While our algorithms could have been implemented in many different programming languages, one condition was the availability of a good implementation of the basic HMM algorithms so that we would save the work of programming all basic things from the scratch. We looked at different implementations. Our first candidate was the *the General Hidden Markov Model library (GHMM)* [27], which is a C implementation of the basic HMM algorithms and has Python support as well, which is advantageous for fast application development. After some initial experiments, we found out that in the current version GHMM did not support multivariate Gaussian output variables, making it unusable to our purpose. This feature is expected to appear in newer versions however. Another serious candidate we considered was the *Hidden Markov Model Toolkit (HTK)* [19]. This professional package is the standard in the speech processing community, and was developed for this domain originally. While it is possible to use it for gesture recognition, this is quite complicated so that there was even another layer developed by Georgia Tech (Georgia tech gesture toolkit [49]) to make the development process for gesture recognition using HTK easier. The latest version of this gesture toolkit is programmed in Java, which makes it nicer to work with than the former shell-script based implementation. Still, the many layers make the whole process complex, and make it very hard to get control of what is going on at the algorithmic level. The best solution we finally settled on is the Hidden Markov Model Toolbox for Matlab written by Kevin Murphy (1998) [36], which offers a clean and complete implementation of basic HMM functionality. Working with Matlab furthermore has the great advantage that coding goes very fast since it is a scripting language, and debugging is easy since all variables can be easily inspected. The only big disadvantage of Matlab is that it is slow, and hence less suitable for real-time applications. This is the reason we didn't decide to use it right from the start.

Chapter 5

Experiments

5.1 Gesture Databases

5.1.1 Dataset for HMM segmentation

To test our HMM segmentation algorithm we developed a gesture database containing arm- and head movement data recorded with the *X-sens Xbus Kit system* [8]. The *X-sens* posture measuring device consists of various sensors that must be mounted on the different body parts. Using gyroscopes it computes the relative positions of the different body parts, and angles between "limbs" can be computed from those. Using all sensors, including the head sensors, the system was used to generate 14 dimensional data streams of angles between limbs.
We generated a database of 9 gestures containing the three sub-categories "slow", "moderate" and "fast" for different execution speeds. With 12 examples per sub-category of each gesture, this amounts to 36 examples per gesture and 324 examples in total.

We recorded five repetitive gestures: 1: "Come to me", 2: "Go away", 3: "Two hands rotating", 4: "Normal clapping" 5: "Swimming" and four trajectorial gestures: 6: "Simple pointing", 7:"You are dead", 8: "Up yours", 9: "Timeout". All gestures are started with both arms besides the body (rest position), and also typically end in this way. The Appendix gives details about the lengths and the PCA reconstruction error.

5.1.2 Dataset for HMM Compression and HMM Merging

In our tests of the HMM compression and HMM merging algorithms we used the Interactplay dataset [24], which is a hand gesture database made of a 3D hand trajectories recorded by France Telecom Research and Development. It contains 16 hand gestures from 22 persons and provides 5 sessions and 10 recordings per session. The database is 12 dimensional and contains the coordinates of two hands, and the head and the torso, of segmented gestures.

To speed things up, we used two smaller subsets of this big dataset in our tests. The first subset consists of the data of the first four sessions of person 1, the second contains all data from person 1 and 2. We used PCA to lower the dimensionality of the data, keeping 99 % of the data variance, effectively 7 dimensions. The PCA reconstruction error graphs for both subsets are shown in the Appendix in Figure 7.2 and Figure 7.3.

Figure 5.1: Example of the "swim" gesture from the Interactplay dataset. The final 3-d coordinates in the dataset are extracted from the multi-camera video sequence by 3-d visual tracking of the colored markers.

5.2 Experiments

In this section we discuss the experiments that were executed to test our HISAA algorithm and our HMM compression and HMM merging algorithms. While the algorithms have many common features, the task of segmentation and the tasks of HMM compression and HMM merging require significantly different input data and a different form of evaluation. This is reflected by the fact that they both have their own experimental set-ups, discussed in the following paragraphs.

5.2.1 Experimental set-up HISAA tests

In the design of tests for our algorithm HISAA, we pursue different goals simultaneously. First and foremost we want to test if the algorithm succeeds in robustly extracting sensible state groups, so that it can be used to segment a model and give a more informative and compact description of its structure. Our secondary objective is to prove that the algorithm can be used for clustering of unlabeled gesture data, and segmentation and extraction of unlabeled gesture examples. This last goal is obviously more difficult to achieve, also since it depends on many other algorithms besides HISAA.

In our pilot experiments we worked with 2-d gestures consisting of circles, lines and letters that were constructed by hand in Matlab. Using this gestures we trained a big HMM for a sequence of atomic sub-gestures, and then segmented this big HMM with HISAA into cycles and trajectories. The promising results we achieved on those tests motivated us to pursue our research further with more serious tests on a multi-dimensional gesture database. When using HISAA we have to chose both the *cycle transition threshold* and the *trajectory transition threshold* such that the cycle and trajectory state groups can be effectively extracted by the algorithm. In our pilot tests we worked with values of 0.5 for both these thresholds.

The 2-d gestures are nice for illustrative purposes, since the GMM-part of GMM-HMMs is easily visualized with Matlab as a set of ellipses in a plane. With multi-dimensional gestures there is no clear way to visualize the GMM-HMM, which makes it harder to see if the chosen partition of the HMM-states into state-groups is correct.

Our final experiments were done using the gesture database we recorded with the X-sens. We repeatedly choose 2 or 3 gesture types from the database, and then randomly divided the data for those types into a part for training and a part for testing. The different examples in the train and test set are then concatenated in random order to create a long data sequence for training and a similar sequence for testing. With the training sequence, our

"big gesture", we train a HMM called "big HMM" The goal is to segment this "big HMM" into meaningful parts, namely *Cycles*, *Trajectories* and *Singletons* which should map to the original gesture types. We used PCA to compress the gesture data,using only the first 12 PCA components we maintained > 99 % of the variance from the original data.

Both our Cycle-extraction and Trajectory-extraction algorithm depend on a threshold on the minimum required transition chance for the transitions that are part of the *Cycle* or *Trajectory*. We do not know what the best values are for those thresholds. We therefore test with different combinations of Cycle-transition and Trajectory-Transition thresholds, starting with 0 and ending with 1 for both, and trying all intermediate combinations using a step-size of 0.05.
After the *Cycles*, *Trajectories* and *Singletons* have been extracted, we map those state groups to the closest labeled HMMs. For this mapping we use a simple Gaussian Mixture Model distance [18]

$$d(f,g) = \sum_{i=1}^{k} \alpha_i min_{k=1}^{m} KL(f_i||g_j)$$

The advantage of this distance is that it can be computed exactly, and does not rely on sampling as many other distance measures do [14] , that attempt to compute the unconstrained KL-divergence between two MOGs or HMM models. The KL-divergence between two Gaussians $KL(f_i||g_j)$ in this formula is computed as.

$$\begin{aligned} &KL(N(\mu_1,\Sigma_1),N(\mu_2,\Sigma_2)) = \\ &\frac{1}{2}(log\frac{|\Sigma_2|}{|\Sigma_1|} + Tr(\Sigma_2^{-1}\Sigma_1) + \\ &(\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2) - d \end{aligned}$$

We only use labeled HMMs corresponding to gestures that also really appeared in the "big gesture" data, since we may assume we know which gestures we are looking for (open ended classification is another, more difficult problem). In our tests, the labeled HMMs are trained with all the gesture data (from the "moderate" subset) of our database.
After the mapping has been completed, we use the Viterbi Algorithm on the "big HMM" to generate a Viterbi-state-sequence for the training and test sequence. Those state sequences are then mapped to class sequences, using the state to group to class mapping computed in the previous steps. Finally, the achieved class sequences for the training sequence and the test sequence are compared with the "ground truth" of the real class sequence of the training and test data to compute an accuracy in retrieving the right class from the structural groups. We define the segmentation accuracy then as the fraction of data points from the gesture sequence that is via its corresponding

datapoint → *State* → *StateGroup* → *labeled HMM* mapping assigned to the correct labeled HMM:

$$\text{SegmentationAccuracy} = \frac{\text{Datapoints mapped to correct labeled HMM}}{\text{Number of Datapoints in Big Gesture Data Sequence}} \quad (5.1)$$

We tested one variant of our original algorithm HISSAA-SC that is constrained to only use the labeled HMMS chosen by the *Cycles* and *Trajectories*. The motivation is that every gesture should have at least some part that is a cycle or trajectory, and those structures are more reliable for selecting HMMS than singleton states.

Apart from the accuracy of our Algorithms, we compute two other performance measures. The *Baseline Performance* is computed as the performance gotten when we use no structure, so every single state is mapped to the closest labeled HMM. The *Theoretical best possible performance* is computed by mapping every state to the best class, i.e. the class that will lead to the best performance. If the classes were perfectly separable by the states, the *Theoretical best possible performance* would be 100%; since gestures do tend to overlap,the best performance possible is lower.

5.2.2 Experimental set-up HMM compression and HMM merging tests

To test our compression algorithm we compared the performance of directly trained HMMs and compressed HMMs. The first are directly trained on the database, the second are formed by compressing HMMs with a larger amount of states trained on the same database. For our test we took the data of all five sessions, and all 16 gestures for the first person from the Interactplay database [24]. For every test the data was randomly split into half train set, half test set (25 training/test examples per HMM). As usual for basic gesture recognition, in the evaluation then the test examples (gestures) are matched with the HMMs trained for all different gestures. The gesture label of the HMM that assigns the highest likelihood to the example is taken as the retrieved gesture. Accuracy is then computed as:

$$\textit{RecognitionAccuracy} = \frac{\text{Number of Correctly labeled test examples}}{\text{Total number of test examples}} \quad (5.2)$$

which can be multiplied by 100 to get the percentage of correctly recognized gestures.

In our test, we compressed HMMs of 12 states into simpler HMMs with a smaller amount of states. We compressed to HMMs having 1 to 10 states, and also directly trained HMMs on the original data with this amount of states. Compression was done either by our Direct Compression Algorithm or alternatively by re-sampling data from the to-be-compressed

HMM followed by training of a smaller model on this data. The Baum-Welch HMM training was initialized by K-means over all data. We allowed 5 trials for training every HMM to compensate for the sensitivity of the Baum-Welch algorithm to its initialization. We kept for every gesture the model with the highest data likelihood.

For testing our Merge Algorithm we used two data subsets, which in turn were divided in two parts for training and testing sets. Then on the train parts of both subsets we trained HMMs for all gestures. A full data train/test set was created by merging the train/test part of both train/test subsets. We then ran different tests.
To get reference scores we first tested the HMMs trained with the subset train parts on their corresponding subset test parts. We also tested those subset-HMMs on the full data test set, to see how well they would already naturally generalize to the other subset without any merging being performed. For comparison we tested also HMMs trained on the full train set on the full test set.
Then, we formed new HMMs by merging the HMMs trained with the train part of the first and second subset. This Merging was performed either by our Algorithm or by simple re-sampling of data from both merged HMMs followed by training of a new HMM. A hybrid Algorithm, using our Algorithm for most cases and re-sampling for the two most difficult merging cases was tried as well. In all those tests the performance measure used was the simple recognition accuracy (see 5.2).

The merge Algorithm was applied to merge either HMMs trained on different sessions for the same person, or HMMs trained for different persons. In the first test, the data subsets were formed by taking all data from session 1 + 2 for the first subset and session 3 + 4 for the second subset. With 10 examples per session and 16 gestures, this gave sub train-/test sets of 160 examples and full train/test sets of size 320.examples. In the second test, the first subset was formed by taking all data from person 1 and the second subset by taking all data from person 2. With a total of 5 sessions per person, this gave sub train-/test sets of 400 examples and full train/test sets of size 800 examples.

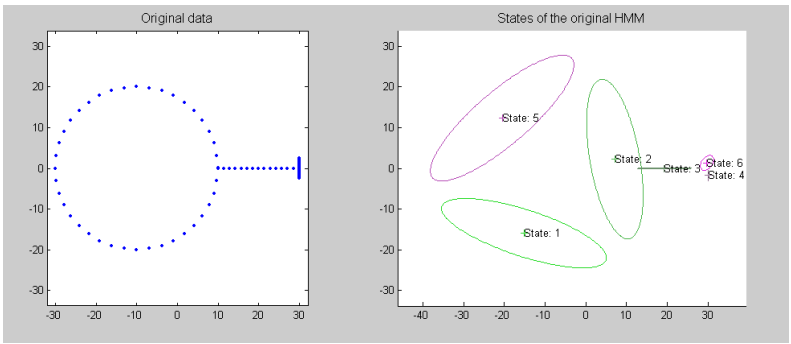
All our tests were repeated 30 times, in every repetition using a different randomly generated split of the data into train and test parts, to get significant results.

5.3 Results

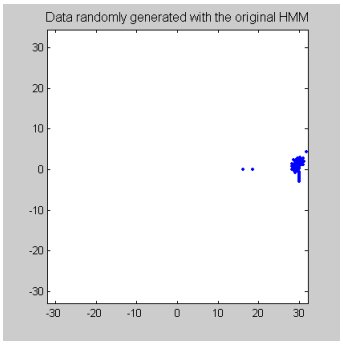
5.3.1 Results HISAA tests

Results Pilot tests

In the first pilot experiment we tested the effectiveness of the HISAA algorithm to partition a strongly structured data sample into its underlying strong cycles and intermediate states. The artificially generated data consists of repeatedly three times a circle, followed by some intermediate data that goes from the left to the right, followed by three times a line up and down, followed by intermediate data back to the circle again. This sequence of [circle, intermediate, line, intermediate] is itself repeated four times. The original HMM trained for this data contains 6 states with only one mixture component per state. In Figure 5.2 we show the original data and the original HMM extracted from it, as well as the data randomly generated by sampling from the found HMM. Since the data is two-dimensional, we plot only the positions and leave out the time information in our plot of the original data.



(a) Original Data (b) *Original Data HMM*



(c) Data Sampled from *Original Data HMM*

Figure 5.2: Original Data and *Original Data HMM*

As a next step we applied the HISAA algorithm to the original HMM. The algorithm segments the HMM into groups of states corresponding to cycles, trajectories and singleton states (see Figure 5.3).The algorithm finds two cycle state groups, which neatly correspond to the two cycles (the circle and the line). The state corresponding to the transition between the circle and the line is adequately segmented as a singleton state.

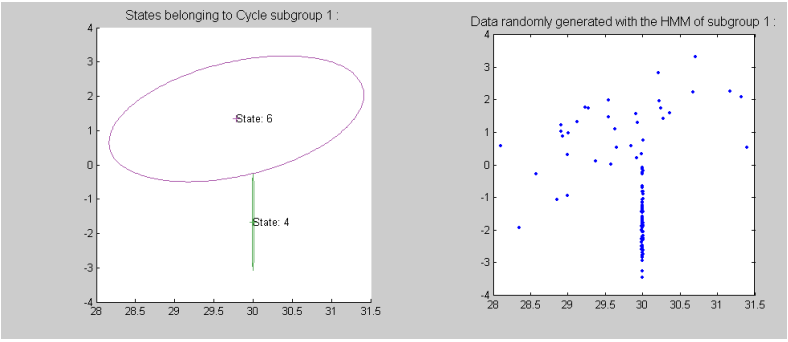
As a last step in this experiment, we compute the Viterbi state sequence for the original data, and map this state sequence to a subgroup number sequence, using the known state to subgroup mapping. Here (see Figure 5.4) we see the pattern of circle, intermediate, line, intermediate, circle ... is clearly retrieved. It must be remarked that this experiment sometimes goes wrong as well, with the intermediate state becoming part of one of the two other state groups. Effectiveness also depends on the original HMM found by Baum-Welch training. Since this is contains random elements, and can easily converge upon suboptimal solutions, it sometimes finds a set of Gaussians that makes a perfect segmentation of the circle and the line a priori impossible.

In the second pilot experiment we tested the effectiveness of the combined usage of strong cycles and strong trajectories to partition a data sample containing both cycles and trajectories into these natural sub-structures. The artificially generated data consists of both the letters Z and W, but additionally also contains the circle from the first example. The complete sequence of data is : [z; w; circles; w; circles; z; circles]. Strong trajectories are a weaker version of strong cycles. In other words, if we have a strong cycle, then we could also have a whole set of strong trajectories that start from any state on the cycle and end just before the cycle turns back to this starting state. It is therefore natural to first find all strong cycles, and remove their corresponding state from the transition Matrix, and then search for trajectories only in the reduced transition Matrix of the remaining states. This is the approach taken throughout this work, when combining cycles and trajectories. Indeed it turned out, that in this way the Algorithm succeeded well at first extracting the states corresponding to the circle, and then from the remaining state identifying to trajectory-subgroups corresponding to the two letters.
In this experiment we used 15 states with one mixture component per state for the trained HMM.

In Figure 5.5 we show the original data and the original HMM extracted from it, as well as the data randomly generated by sampling from the found HMM.

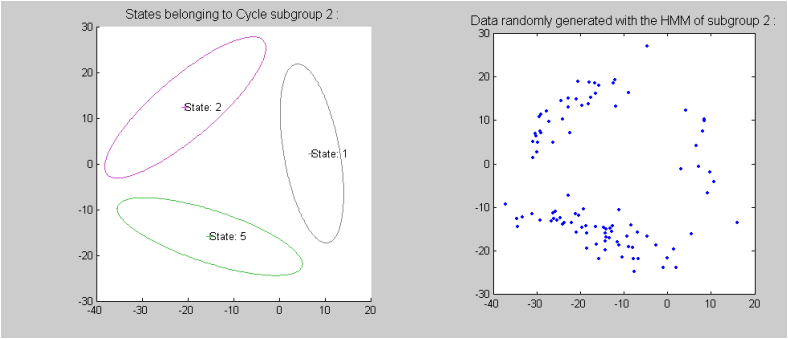
Next, in Figure 5.6 we show the retrieved sub-HMMs. The first sub-HMM corresponds to the cycle, the last two to the two trajectories of the letters.

Finally, we show the classification results. Figure 5.7 shows the result of mapping the Viterbi state sequence of the original data to the corresponding



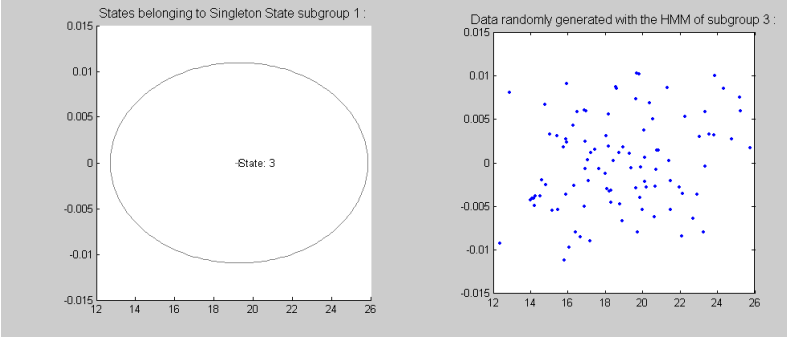
(a) First Sub-HMM

(b) Data Sampled from First Sub-HMM



(c) Second Sub-HMM

(d) Data Sampled from Second Sub-HMM



(e) Third Sub-HMM

(f) Data Sampled from Third Sub-HMM

Figure 5.3: The three Sub-HMMs found

sub-HMM groups. If we compare the group sequence here with the class sequence [z; w; circles; w; circles; z; circles] of the original data, we see that the group sequence matches the original data. Figure 5.8 shows the results for the classification of the original data with the real class labels. Since the

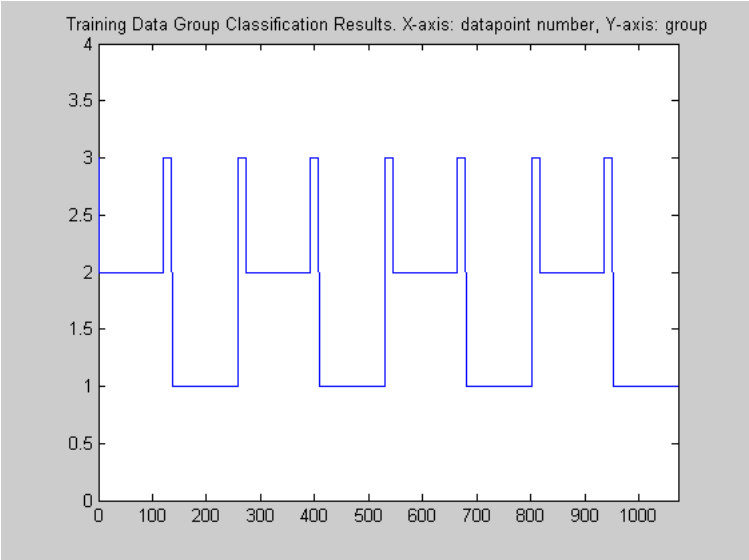


Figure 5.4: Original Data Subgroup Classification results

states are mapped to the right groups, and the groups get mapped to the right classes the results are perfect for this example.

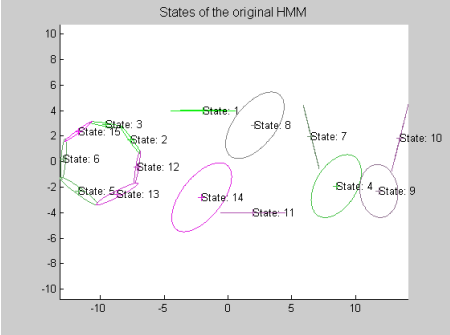
The third pilot experiment is an exact repetition of the second, except that we have now added uniform Gaussian noise with a mean of 0 and a covariance of 0.6 in all dimensions.

In Figure 5.9 we show the original data and the original HMM extracted from it, as well as the data randomly generated by sampling from the found HMM.

Next, in Figure 5.10 we show the retrieved sub-HMMs. The first sub-HMM corresponds to the cycle, the last two to the two trajectories of the letters.

Again we show the classification results. Figure 5.11 shows the result of mapping the Viterbi state sequence of the original data to the corresponding sub-HMM groups. This mapping is still consistent with the original data sequence. Figure 5.12, and the classification proves to be correct as well. We performed experiments with even more noise as well, but while the algorithm still sometimes works with noise levels of covariance 1 or higher, it starts making errors. Those errors can occur in the state grouping or state group to labeled HMM mapping, which leads to big errors. Alternatively sometimes the higher amount of noise can also lead to small errors in the Viterbi state decoding, effectively retrieving the wrong state for the observation. Then, even though the state groups and even state to labeled HMM mapping are still correct, the classification graph starts to contain small errors.

(a) Original Data (b) *Original Data HMM*

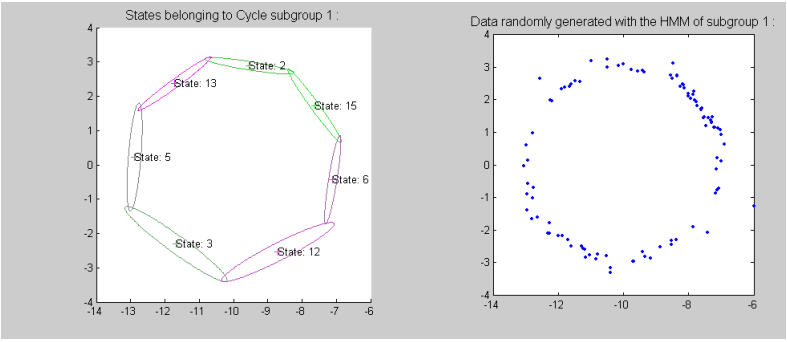


(c) Data Sampled from *Original Data HMM*

Figure 5.5: Original Data and *Original Data HMM*

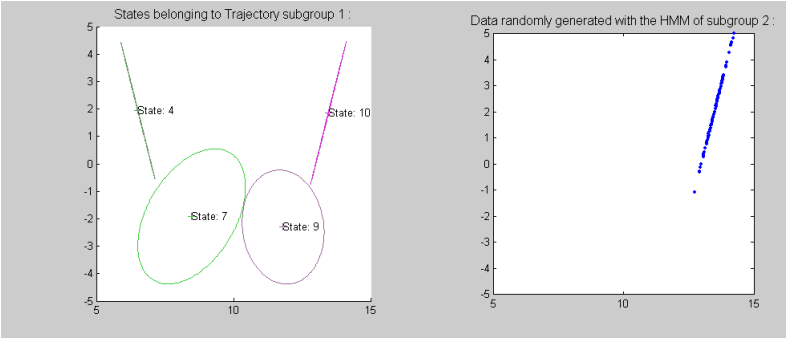
Results tests on database

We performed different tests with our gesture database. We list the results of two tests, which were performed with the "moderate" subset and respectively 2 and 3 gesture types. Ten trials were used in the construction of every HMM to compensate for the tendency of the EM-Algorithm to convergence to local optima and its strong sensitivity to the initialization. We verified with simple two-dimensional gestures, that keeping the HMM with the highest data likelihood over several trials usually leads to much better models. The number of EM-iterations itself was limited to 20. We used 15 states with 2 mixture components per state in the "big HMM". For the labeled gestures, we took 10 states with two mixture components. The data was split with 2/3 (8 examples per type) for training and 1/3 (4 examples per type) for testing. In our test with 2 gesture types, the gestures were drawn from two different groups in an attempt to lower the amount of spatial overlap between the different gestures, which leads to badly separable models. We used complete link clustering on the GMM distance to form groups to have maximum spatial separability.



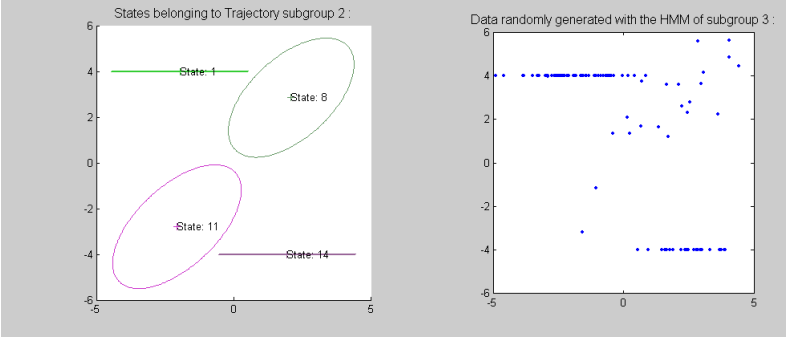
(a) First Sub-HMM

(b) Data Sampled from First Sub-HMM



(c) Second Sub-HMM

(d) Data Sampled from Second Sub-HMM



(e) Third Sub-HMM

(f) Data Sampled from Third Sub-HMM

Figure 5.6: The three Sub-HMMs found

Every test was repeated 30 times, to allow the different combinations of gestures to be selected, and get statistically valid results.

In the first test with 2 gesture types (see Table 5.1 – 5.4) we achieved a training performance of 78.0% and a test performance of 71.5%. In the

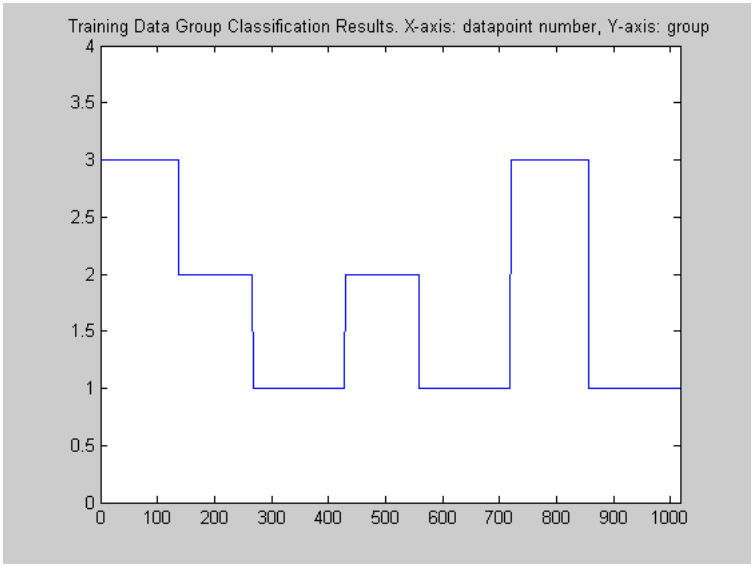


Figure 5.7: Original Data Subgroup Classification results

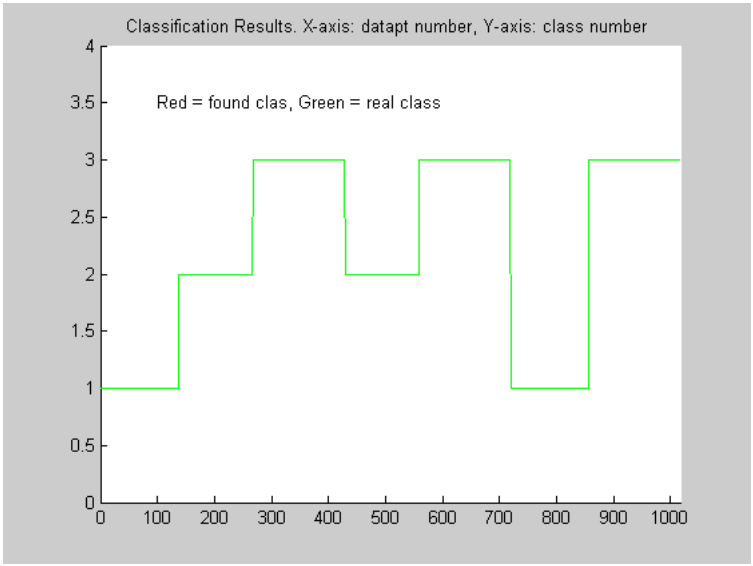
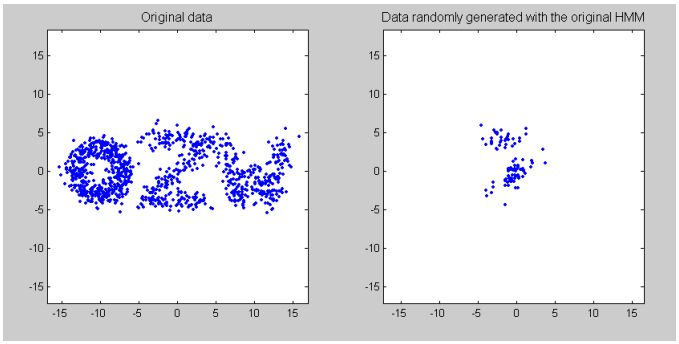
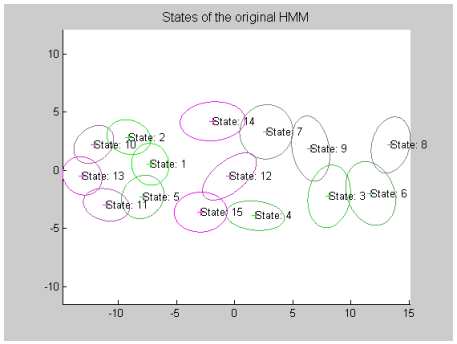


Figure 5.8: Original Data Real Class Classification results

second test with 3 gestures (see Table 5.5 – 5.8), the performance decreased somewhat and we had 69.2% for training sequence and 64.3% for the test sequence. Those performances were significantly higher than the results for the Baseline method (with 76.5% train, 68.5 % test for the first experiment



(a) Original Data (b) *Original Data HMM*



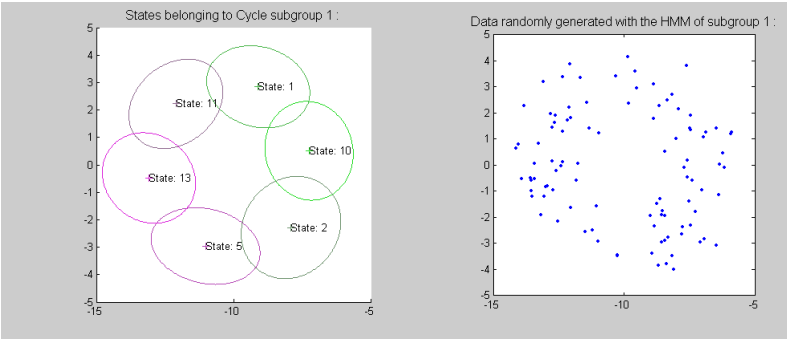
(c) Data Sampled from *Original Data HMM*

Figure 5.9: Original Data and *Original Data HMM*

and 66.1% train, 62.1% test for the second experiment).
Above three gesture types the performance deteriorates, since the overlap between the different gestures then causes almost every state of the "big HMM" to belong to different classes simultaneously, and at the same time no strong temporal structure won't be learnable anymore since the spatial separability lacks.

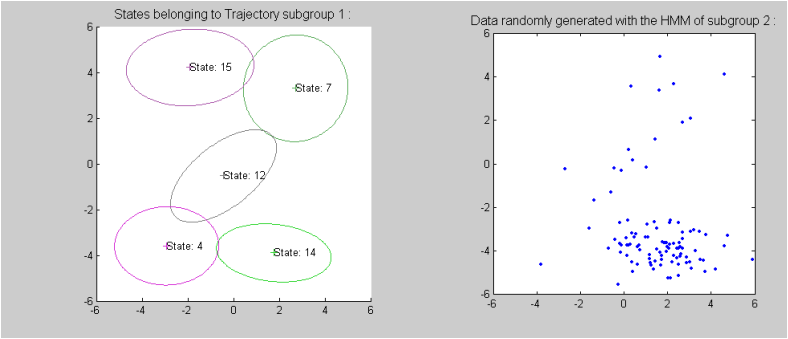
5.3.2 Results HMM compression and HMM merging tests

The test of our compression Algorithm (see Table 5.9) showed that compression from a bigger model to a smaller model almost gives the same performance as directly training smaller model on the original data ("direct training"). Direct training is still superior, but with an average performance loss only 0.42% (not considering the performances with only one state) the difference is almost negligible.
It is remarkable that compression through data re-generation does not perform better than direct compression in our test. The average performance



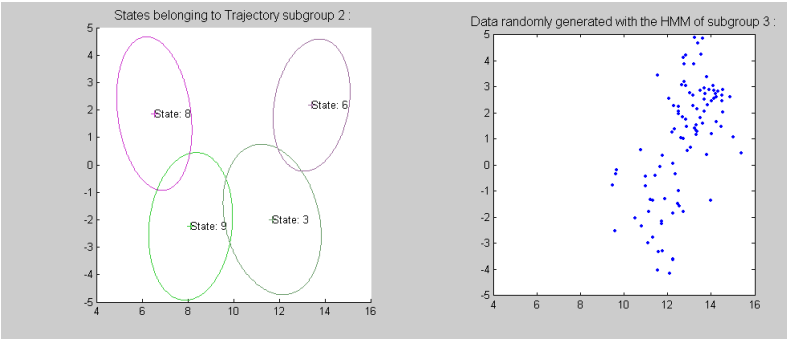
(a) First Sub-HMM

(b) Data Sampled from First Sub-HMM



(c) Second Sub-HMM

(d) Data Sampled from Second Sub-HMM



(e) Third Sub-HMM

(f) Data Sampled from Third Sub-HMM

Figure 5.10: The three Sub-HMMs found

loss over direct training of re-sampling compression is 2.1% for resampling with 20 samples and 0.9% for re-sampling with 50 examples. This is more than two times worse than the average performance loss of 0.42% obtained by direct compression. When looking at the performance graphs (see Figure

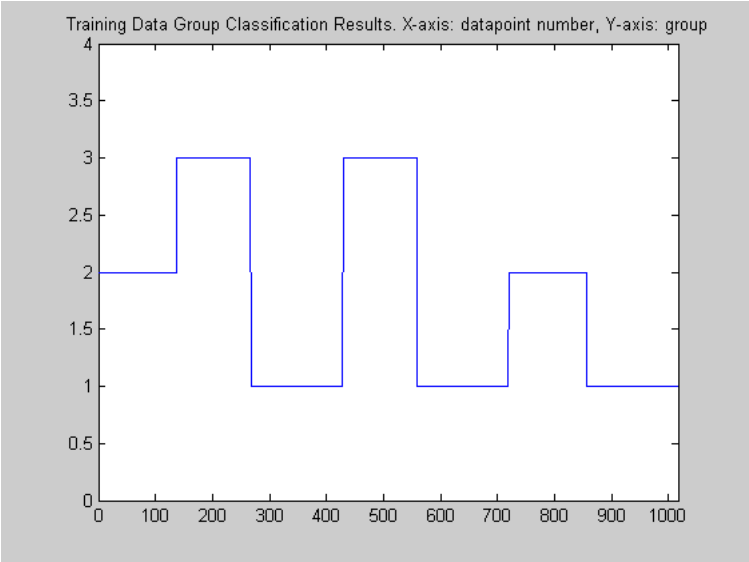


Figure 5.11: Original Data Subgroup Classification results

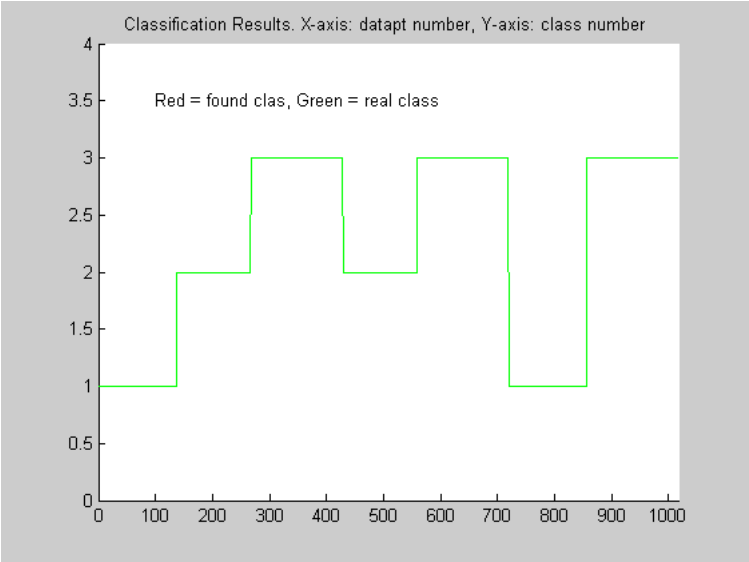


Figure 5.12: Original Data Real Class Classification results

5.14) we see that direct training works best, but direct compression gives almost the same performance. Compression by data re-generation from the compressed model improves with the amount of data samples generated and used for the training of the compressed model. The lowest graph corresponds

Table 5.1: Results for Trainsequence

Method	Segmentation Accuracy
Baseline	0.765 ± 0.084
Theoretical best possible	0.852 ± 0.050
HISAA best[1]	0.780 ± 0.075
HISAA-SC best[2]	0.765 ± 0.084

[1] Using Cycle- and Trajectory thresholds of 0.5 and 0.8
[2] Using Cycle- and Trajectory thresholds of 0.7 and 1.0

Table 5.2: Best Grouping TrainSequence

	HISAA	HISAA-SC
cycles	0.767 ± 0.935	0
trajectories	1.167 ± 0.913	0
singletons	10.700 ± 2.693	15
total	12.633 ± 1.671	15

Table 5.3: Results for Testsequence

Method	Segmentation Accuracy
Baseline	0.685 ± 0.157
Theoretical best possible	0.804 ± 0.129
HISAA best[3]	0.715 ± 0.145
HISAA-SC best[4]	0.706 ± 0.141

[3] Using Cycle- and Trajectory thresholds of 0.50 and 0.80
[4] Using Cycle- and Trajectory thresholds of 0.70 and 1.00

Table 5.4: Best Grouping TestSequence

	HISAA	HISAA-SC
cycles	1.600 ± 1.102	1.600 ± 1.102
trajectories	2.233 ± 1.165	2.233 ± 1.165
singletons	3.833 ± 2.653	3.833 ± 2.653
total	7.667 ± 2.537	7.667 ± 2.537

to re-training with 20 samples per HMM. The second lowest dotted graph corresponds to training with 50 samples per HMM. With 50 samples we have twice the amount of data on which the original HMMs were trained (25 samples), but still the results are worse than for direct compression. The difference in performance between 20 and 50 samples shows that with more samples the compression by data re-generation improves, so that in the limit it may well even slightly improve over our direct compression method. On the other hand, the computational cost of EM-training increases linearly in the amount of data. We are thus likely to pay a great increase in computational cost to beat our direct compression method by compression through data re-generation.

Our Merge tests show different things. In the first test (see Table 5.10) we see that training on all data gives a performance of 98.7% (the best score). Our Merge Algorithm performs not that good, but gets close nonetheless with a performance of 94.5% (a relative performance loss of 426% from the best). The alternative method that uses re-sampling performs even better, with 98.1% (a relative performance loss of 0.61%

Table 5.5: Results for Trainsequence

Method	Segmentation Accuracy
Baseline	0.661 ± 0.089
Theoretical best possible	0.793 ± 0.059
HISAA best[5]	0.692 ± 0.094
HISAA-SC best[6]	0.672 ± 0.099

[5] Using Cycle- and Trajectory thresholds of 0.35 and 0.90
[6] Using Cycle- and Trajectory thresholds of 0.30 and 0.50

Table 5.6: Best Grouping TrainSequence

	HISAA	HISAA-SC
cycles	2.100 ± 0.923	2.433 ± 1.006
trajectories	0.167 ± 0.379	0.800 ± 0.925
singletons	6.967 ± 2.977	4.233 ± 2.582
total	9.233 ± 2.909	7.467 ± 2.700

Table 5.7: Results for TestsequenceSequence

Method	Segmentation Accuracy
Baseline	0.621 ± 0.153
Theoretical best possible	0.790 ± 0.127
HISAA best[7]	0.643 ± 0.140
HISAA-SC best[8]	0.630 ± 0.173

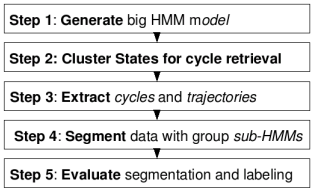
[7] Using Cycle- and Trajectory thresholds of 0.45 and 0.95
[8] Using Cycle- and Trajectory thresholds of 0.30 and 0.35

Table 5.8: Best grouping TestSequence

	HISAA	HISAA-SC
cycles	1.367 ± 0.928	2.433 ± 1.006
trajectories	0.300 ± 0.466	1.233 ± 1.165
singletons	10.200 ± 2.882	2.767 ± 1.654
total	11.867 ± 2.330	6.433 ± 2.063

from the best). The reference scores show that the HMMs trained on only the subsets give much lower scores when tested on the complete data test set (71.8%,70.9%). Testing those HMMs on only the subset test sets gives very high performances (99.5%,96.4%), which proves that the higher performances gained by model merging are not just an artifact of training with more data.
In the second test (see Table 5.11) all performances decrease, with the performance achieved when training and testing on all data now being 94.4% (the best score). For the re-sampling merging the performance decreases to 92.3% (a relative performance loss of 2.23% from the best). Our merging Algorithm however drops to 81.5% (a relative performance loss of 13.67% from the best). Combining the results from both tests we can compute that the relative performance loss from the best score for the resampling method increased from 0.6% in the first test to 2.23% in the second test, a relative 366% increase in relative error. On the other hand, the relative performance loss of the direct merging method increased from 4.26% to 13.67%, a relative 321% increase in error. We see that while the two methods

Figure 5.13: Flowchart HISAA experiments



show very different absolute decreases in performance for more difficult merging problems, their relative decreases in performance are remarkably quite comparable.

We conclude from this test that the absolute performance of our Merging Algorithm deteriorates when the HMMs to be merged become too different, which is the case for the HMMs of two different persons. The Hybrid Algorithm shows some improvement over purely using our Merge Algorithm, but it fails to achieve a performance equal to the re-sampling method. Inspecting the confusion Matrices during testing suggested that this is mainly explainable by the fact that HMM distance fails to find the HMMs that gives most problems in merging. It is however also true that as all HMMs become more dissimilar, selecting a few hard cases and using only re-sampling for those is no longer a feasible solution in any case.

5.4 Discussion and further work

5.4.1 HISAA

The experiments established the ability of the HISAA algorithm to extract meaningful structure from the Transition Matrix of an HMM, and hinted that with some improvement of the method used for mapping groups

Table 5.9: Recognition Accuracies on Testset: Compression test

States:	Direct Training	Direct Compr
1	0.756 ± 0.040	0.877 ± 0.019
2	0.930 ± 0.021	0.916 ± 0.023
3	0.944 ± 0.020	0.944 ± 0.020
4	0.959 ± 0.018	0.956 ± 0.017
5	0.969 ± 0.015	0.964 ± 0.016
6	0.973 ± 0.014	0.970 ± 0.016
7	0.975 ± 0.013	0.972 ± 0.010
8	0.978 ± 0.011	0.976 ± 0.011
9	0.981 ± 0.011	0.977 ± 0.012
10	0.983 ± 0.012	0.981 ± 0.010

States:	Data-Resampling Compr 20 samples	Data-Resampling Compr 50 samples
1	0.705 ± 0.037	0.718 ± 0.045
2	0.909 ± 0.025	0.919 ± 0.017
3	0.921 ± 0.021	0.932 ± 0.018
4	0.934 ± 0.022	0.943 ± 0.016
5	0.944 ± 0.020	0.954 ± 0.013
6	0.950 ± 0.019	0.962 ± 0.012
7	0.956 ± 0.019	0.969 ± 0.009
8	0.959 ± 0.020	0.974 ± 0.010
9	0.963 ± 0.019	0.977 ± 0.009
10	0.967 ± 0.017	0.980 ± 0.008

Method:	Average performance loss compared to direct Training
Direct Compr	-0.004 ± 0.004
Resampling Compression 20 samples	-0.021 ± 0.003
Resampling Compression 50 samples	-0.009 ± 0.005

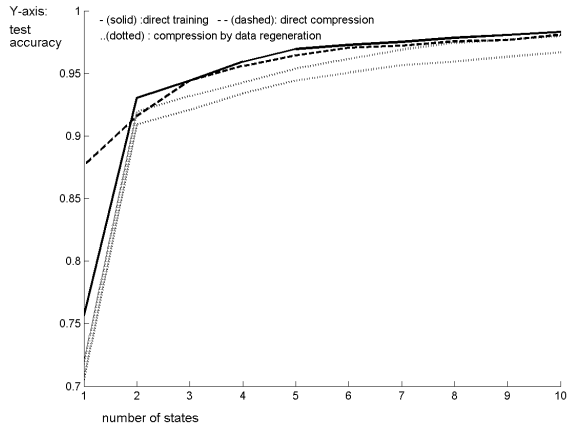


Figure 5.14: Compression Test results

to labeled HMMs, it might be useful for clustering and segmentation of gestures. The pilot experiments showed that a big gesture consisting of clearly separable sub-gestures can be perfectly segmented into those sub-gestures. Condition for such perfect segmentation is that the sub-gestures

Table 5.10: Recognition Accuracies on Testset: Merge test person 1

Test:	Recognition Accuracy:
Trained on both subsets together	0.987 ± 0.007
HMMs Merged by Merge Algorithm	0.945 ± 0.022
HMMs merged by Resampling	0.981 ± 0.009
Hybrid Algorithm	0.959 ± 0.019
Trained: subset 1, Tested: subset 1	0.995 ± 0.008
Trained: subset 2, Tested: subset 2	0.964 ± 0.024
Trained: subset 1, Tested: subset 1 + 2	0.718 ± 0.017
Trained: subset 2, Tested: subset 1 + 2	0.709 ± 0.031

Table 5.11: Accuracies on Testset: Merge tests person 1 and 2

Test:	Recognition Accuracy:
Trained on both subsets together	0.944 ± 0.012
HMMs Merged by Merge Algorithm	0.815 ± 0.035
HMMs merged by Resampling	0.923 ± 0.009
Hybrid Algorithm	0.842 ± 0.035
Trained: subset 1, Tested: subset 1	0.980 ± 0.010
Trained: subset 2, Tested: subset 2	0.938 ± 0.014
Trained: subset 1, Tested: subset 1 + 2	0.639 ± 0.013
Trained: subset 2, Tested: subset 1 + 2	0.658 ± 0.020

have strong temporal coherence and no mutual spatial overlap (so that states are not shared between different sub-gestures) The experiments also showed limitations of the algorithm. It works bad when gestures are too much spatially overlapping. We noticed that the problem of spatial overlap is also a problem of the Baum-Welch Algorithm that trains the HMM model. Often there is much spatial but not temporal overlap, for example if there are two overlapping circles, one traversed clockwise and the other counter clockwise. Duplicating states at points of much spatial overlap, and using one of the duplicates for every overlapping gesture, would be an intuitive idea to get better models. However,experiments showed that the Baum-Welch algorithm cannot exploit such duplicate states, even if we manually generate them. It sticks with suboptimal models that put all spatially close points in same states independent of their temporal properties. Stronger models such as Hierarchical Hidden Markov Models (HHMMs) might offer a solution to this problem. A conceptual weakness of the HISAA algorithm is its reliance on thresholds, which have to be optimized experimentally. Our experiments showed that while optimal values have reasonable values (Cycle- and Trajectory thresholds of 0.35/0.45 and 0.90/0.95 for training and testing respectively) those values show variance over tests, and hence it is difficult to find single best values through experimental optimization. The problem here is that we search for cycles and trajectories, which are in principle fuzzy concepts that can occur with different *strengths* within the HMM. To extract those structures efficiently however, we have to define a strict criterion that defines when a collection of connected states can be considered a trajectory

or cycle and when it no longer can. The most straightforward way to do this is through thresholds on the transition chances, but we considered other alternatives as well. One can think in the direction of finding multiple candidate sets for cycles, trajectories and singletons, and then performing some global optimization scheme that finds the best segmentation amongst those candidates according to some heuristics and prior knowledge on the total number of gesture classes. At the cost of much more complex cycle and trajectory extraction algorithms such an approach might eventually lead to better segmentations.

One other thing we noticed is that the use of labeled HMMs to map state-groups to a ground-truth of labeled gesture sequences is a somewhat unsatisfying solution to the problem of formal evaluation of our HISAA algorithm. The problem with this approach is that we are just as much measuring the adequacy of our HMM distance metric, on which the quality of the mapping relies, as we are evaluating the quality of our partition into groups. Apart from the GMM-distance metric, we also tried wit the Kullback-Divergence sample based HMM distance measure. This "more advanced" distance measure however is very unreliable as it often gives infinite distances, which happens when a certain sample from one model has zero probability for the other model, which is often the case. This method thus relies greatly on the used samples, and requires very big sample sizes to make it at least a bit robust, which was not feasible for our already computationally intensive tests. It would clearly be nice to have a meaningful evaluation method for HISAA that does not rely on performance on a segmentation task, and thereby indirectly depending on many other methods as well. Application of HISAA to other problems might naturally provide such a better evaluation metric.

We envisage another promising application for HISAA besides its current use for gesture segmentation. This would be to use it in generating a better HMM distance metric, that uses the model structure. The popular Kulback-Divergence distance metric [38] is inappropriate for computing distance between HMMs that overlap in many parts, but are very different on some small part. Since it works with sampling, the small non-overlapping parts of both models will then generate data inexplicable by the other model. This means the whole distance becomes $(\pm)\infty$. A distance metric based on HISAA could work state-group wise, thus distinguishing no overlap for some parts of the models from no overlap for all parts of the models. This important distinction cannot be made by standard KL-divergence, which only give reasonable, non-infinite distances when the models are quite close for all parts.

5.4.2 HMM compression and HMM merging

We've tested the model based Compression and Merging algorithm. Test of the compression algorithm showed that it achieved a very high performance, only slightly worse (at most 0.42 % on average) than direct training on the original data. Compression by data re-generation did not achieve better results, while sampling up to two times the number of original examples that were used in the original training. This proves that direct model compression without retraining of a new model is feasible, and might in some scenarios be an interesting option if many HMMs need to be compressed and computation time is a significant issue. There are many scenarios where HMMs might be needed at different level of complexity/precision for the same data. One example would be mobile computing where the systems on site have limited computational resources, so that they need more simplified models than a server system that can perform more intensive computation if required. Especially when the data changes often, it might be more efficient to just keep one complex HMM and derive the more simplified HMMs from it by model compression, then to retrain all small HMMs as well from scratch.

The tests showed that our Merging Algorithm works reasonably well in most cases, although it cannot reach the same level of performance as training of a new HMM based on re-sampled data. They also suggest that direct model merging is most beneficial if the merged models are very similar, for in this case the performance loss compared to the re-sampling method is smallest, while the computational gain is the same. When HMMs are more fine-grained (have more states), the performance of direct model Merging will typically be better, while the cost of re-computation of a new HMM increases. Still, most HMMs used in practical applications have less than 30 states, so that the computational cost of recomputing a new HMM is not always a big issue. Despite the loss of performance when compared to the re-sampling alternative, the Algorithm might have a function in Algorithms that must merge many HMMs realtime. Since the computation times of our Algorithm does not grow linearly with the amount of (re-sampled) data as discussed in the previous section, it can merge sets of HMMs faster than can be done by training a new model with Baum-Welch from re-sampled data. The results are interesting from a theoretical point of view, as they show that also for HMM models the parameters can be exploited directly to construct compressed or merged models, similarly to the way this has already been done for GMMs. From a practical point of view it may be questioned if there are many scenarios were computational cost would be the dominant factor for HMM merging, so that a more imprecise but also computationally cheaper algorithm would be preferable to the more expensive alternative of data re-sampling.

5.4.3 Comparison of the two approaches

We now compare our two approaches towards HMM analysis and simplification. Segmenting HMMs into structural parts, as we do with our first approach, has the advantage of keeping all information and providing a better insight into the structure of the HMM. At the same time it requires additional steps to compress the HMM based on its structural decomposition, or to exploit the structural decomposition for other things such as data segmentation. Our second approach of directly compressing or merging HMMs has the advantage of providing a more direct result in the sense of a smaller compressed or merged model. At the same time it does not provide the structural insight our first method gives and has a smaller promise of being useful in some form to other methods. It is also possible to combine both approaches, that is to first segment a HMM into parts, and then to compress the parts separately with the algorithms from our second approach. Alternatively, it may be the case that different HMMs are learned, that have a big overlap. In that case our first approach may be helpful to segment those models into parts, then identify the parts that reappear in different models, and finally compress by storing the shared parts only once.

While HMMs are often used as black boxes, our research shows that for the analysis and simplification of HMMs much can be gained with the approach of working directly on the model parameters.

5.4.4 Further work

Better segmenting spatially overlapping sub-gestures

The HISAA algorithm is limited in its ability to segment sug-gestures that are spatially overlapping. Such gestures are typically sharing states in the BIG-HMM for the entire gesture sequence. Our segmentation works by partitioning the complete state set, assigning every state to belong to only one sub-gesture. Therefore it is unable to deal properly with spatial overlap, states that are shared by multiple sub-gestures. One extension to overcome this problem would be to adapt the algorithms so that states can be assigned to multiple cycles and trajectories. While such an extension is relatively simple to make, it would introduce some new problems in turn. When states become shared, a straightforward mapping from a state to a labeled sub-gesture is no longer possible.

Replacing Thresholds

Another point of improvement is the usage of thresholds in the extraction of cycles and trajectories. These thresholds must now be optimized through testing. A similar algorithm that would not rely on any thresholds, or

alternatively would set them automatically to appropriate values would be a good improvement.

Automatic determination new number of states

Analogous to the K-means algorithm our direct compression and merging algorithm requires a user-set amount of new states to perform its compression and merging. A nice extension would be to determine an appropriate amount of new states automatically. In case of the Compression algorithm methods that trade-of model accuracy against model complexity, such as the Baysian Information Criterion (BIC) or Akaike Information Cirterion (AIC) [7] could be used. In case of the direct merging algorithm it might be appropriate to introduce a threshold on the maximum allowable distance between states that are merged. Then the algorithm would determine the required new amount of states automatically, performing a lot of merging if the merged models are very similar and only little merging if the models are very distinct.

Incorporating temporal information in compression and merging algorithm

Currently our direct compression and merging algorithm only uses spatial information and ignores temporal information in the Merging process. This is a clear limitation, and overcoming it is likely to improve performance at least in some scenarios. It would be desirable to only merge states if not only their spatial properties but also their temporal properties align. This second feature could be checked for by making a local representation of every state, that contains besides the state itself a representation of the local neighbors of this state. In other words, a feature set representing the most likely predecessor and successor states as extracted from the Transition Matrix might do the job. Using again 4.5 as a distance metric to compute the distance between two mixtures of Gaussians, we could require that two states S_1 and S_2 are only to be merged provided that the distance between their predecessor and successor state sets is small enough. Formally: $d(predecessorStates(S_1), predecessorStates(S_2) < threshold_1$ and $d(successorStates(S_1), successorStates(S_2) < thresh_2$ for some sensible values of the thresholds $threshold_1$ and $threshold_1$. Here we could also define a local temporal distance between two states S_1 and S_2:

$$\begin{aligned} d_t(S_1, S_2) &= d(predecessorStates(S_1), predecessorStates(S_2) \\ &\quad + d(successorStates(S_1), successorStates(S_2)) \end{aligned} \tag{5.3}$$

To make this idea even more concrete, we could implement this idea by using all states as predecessors and successors, but giving them weight according to the Transition chance they have to move to the target state

(for predecessor states) or be moved to from the target state (for successor states).
Instead of applying a maximum on such a local temporal distance as a constraint state merging, we could also make it a more integrated part of the merging algorithm itself by defining a new distance d_{s-t} metric that combines the *spatial* state distance (the distance between the GMMs of the two states) and the *local temporal distance*:

$$d_{s-t}(S_1, S_2) = \alpha \times d(S_1, S_2) + (1 - \alpha) \times d_t(S_1, S_2) \tag{5.4}$$

The iterative state merging algorithm would then use this new distance in the assignment of states to the closest compressed state in the *Regroup* phase of the merging algorithm.
While the described extension has not been implemented yet and indeed their may be better alternatives, it is clear that the incorporation of temporal information to the compression and merging process provides an important direction for further research.

To summarize we discussed better spatial segmentation and the replacement of thresholds as two directions of further work for our HMM segmentation research. Secondly we discussed automatic determination of the new number of states and incorporation of temporal information as accents for future research on HMM compression and merging.

Chapter 6

Conclusion

In this Master project we worked on the general problem of Gesture Recognition, focusing on torso-arm movement gestures as recorded by the posture capturing device Xsens or derived by tracking the limbs in multi-camera video sequences. Gesture recognition can be approached with different methods, but the Hidden Markov Model and its variations like the Hierarchical Hidden Markov Model (HMM) are by far the most popular and most successful tools used for the problem. Our work centers on the problem of Hidden Markov Model Simplification. Simplification is motivated by different demands. One goal is to limit the computational cost or memory requirements. Simplification solves this by limiting the number of models, parameters per model or total parameters. Another important goal is to make models more general, possibly at the cost of some precision. Here simplification offers a solution as well. Simplification covers a range of related approaches that all have in common that they eventually reduce the total description length of the set of all models. One approach is to segment a big model into meaningful smaller parts. If the smaller parts are (partly) shared by different models, they have to be stored only once and the number of total parameters is reduced. A second approach is to merge similar models into one model, which clearly directly reduces the number of models and hence total description length. A third possibility is to compress a model itself directly, replacing it with a simpler one that keeps as much information as possible with the decreased number of parameters. To solve the problem of simplification as discussed above, we studied the current state of the art, extended on existing HMM implementations and introduced new methods. We explored two different methods towards simplifying complex HMMs and gaining more insight into their structure. Our first method is to segment HMMs into structural parts, corresponding to the first simplification approach. Our second method is to directly merge or compress HMMs based on spatial information, corresponding to the second and third simplification approach respectively.

For our first method we introduced a new HMM structure analysis Algorithm called HISAA, which decomposes the states of a HMM into structural groups corresponding to *Trajectories* and *Cycles*. One specific application of the algorithm is the usage for clustering and segmentation of unlabeled (gesture) data. Tests on a small gesture database of 9 gestures recorded with the XSens resulted a training performance of 78.0% and a test performance of 71.5%. in correctly retrieving the type labels for an unlabeled gesture sequence In a second, more difficult test with 3 gestures the performance decreased somewhat and we had 69.2% for training sequence and 64.3% for the test sequence. A baseline method that works only with single states to perform the segmentation task scored 76.5% train performance, 68.5% test performance on the first test an 66.1% train performance, 62.1% test performance on the second test respectively. This showed that the discovery of state groups improved the performance of a segmentation algorithm over the state-group free baseline. The relatively low performances on the segmentation task could be partly explained by spatial overlap, which leads to the sharing of states by different gestures. If states are shared by multiple gestures, any method that attempts to perform classification/segmentation based on state to gesture mappings will suffer in performance. Our algorithm HISAA succeeds in deriving meaningful information about the structure of an HMM, as well as succeeds in extracting groups of associated states corresponding to cycles and trajectories in the model. The experiments showed that this information can have some function in a segmentation task, but with other tasks the algorithm might even be done better justice.

For our second method,we implemented a new HMM model Merging and Compression algorithm, that uses GMM clustering to group states that are close and then uses it again to compress or merge them. Compressing HMMs offers a very direct way to get smaller models that generalize better and are computationally cheaper to work with. It is not always feasible to keep all original data, nor to keep models of many different levels of complexity. This makes compression of a sufficiently rich model an attractive solutions for many applications were a simpler model must be derived on the fly from a more complex one. Similarly, merging models offers a very direct way to decrease the number of parameters to keep, decrease the number of models that must be tested during recognition, and improve generalization of the models.

Our tests on the Interactplay database showed the compression algorithm worked good. While direct compression lost only 0.4% performance over direct training with the original data on average, compression by data re-sampling lost 2.1% with 20 samples and 0.9% with 50 samples on average. This shows that at least for limited sample sizes the direct compression method is superior to the compression by data re-sampling method. In fact, even if for much larger sample sizes re-sampling-compression would work

better, direct-compression is still attractive because of its good performance combined with low computational cost.

Other tests on the Interactplay Database showed that our merging method worked reasonably well. Merging models trained for different sessions of one person, we achieved on average 94.5% recognition accuracy with the models merged by direct merging. With the re-sampling merging method we achieved even 98.1%, compared to 98.7% achieved by direct training on all data. In the second test that involved merging the models of two different persons, we achieved performances of 81.5% and 92.3% for direct merging and re-sampling-merging respectively; compared with 94.4% recognition accuracy achieved with direct training on the full data. From this we concluded that the re-sampling method gives superior results for merging, and the differences between the performances of the merging methods become bigger when the merged models are more different (which makes the merging task harder). On the other hand, our direct merging is computationally cheaper and for merging similar models reaches high performances as well. We experimented with a hybrid merging scheme as well, which was intended to combine the computationally cheap direct merging with the more expensive but better performing re-sampling-merging. Unfortunately, the unreliability of the distance metric that measures similarity for HMMs however made it difficult to come up with a robust method to separate hard- from the easy cases.

To summarize, we introduced a new Algorithm that gives superior results for HMM compression and reasonable results for HMM merging, for which problem re-sampling-merging turned out to work better. Compression and merging of HMM models are tasks that both have applications in their own right, and together account for a significant part of the task framework of HMM simplification.

We saw that the simplification of complex HMMs is an important problem framework, to which solutions have many different applications, particularly within the domain of Gesture recognition. We showed that segmentation through structural analysis, merging and direct compression of HMMs provide promising as well as complementing approaches for this problem. Some of the new methods and Algorithms we introduced such as the algorithm for direct HMM compression can be readily used in new applications and future research.

Bibliography

[1] C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan. An introduction to mcmc for machine learning. *Machine Learning*, 50:5–43, September 2003.

[2] M.Gelgon A.Nikseresht. Fast decentralized learning of a gaussian mixture model for large-scale multimedia retrieval. In *Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP06)*, volume 06. IEEE Computer Society, 2006.

[3] Y. Bengio and P. Frasconi. Input/output hmms for sequence processing. *Advances in Neural Information Processing Systems*, pages 427–434, 1995.

[4] M. Brand, N. Oliver, and A. Pentland. Coupled hidden markov models for complex action recognition. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 994–999, 1997.

[5] Matthew Brand. Learning concise models of human activity from ambient video via a structure-inducing m-step estimator. Technical Report TR-97-25, MERL – A MITSUBISHI ELECTRIC RESEARCH LABORATORY, November 1997.

[6] Matthew Brand. Structure discovery in hidden markov models via an entropic prior and parameter extinction. Technical report, Mitsubishi Electric Research Labs,, October 1997.

[7] K.P. Burham and D.R. Anderson. Multimodel inference: Understanding aic and bic in model selection. *Sociological Methods and Research*, 33(2):261–304, 2004.

[8] Xsens Technologies B.V. Xsens xbus kit. http://www.xsens.com.

[9] S. Calinon and A. Billard. Recognition and reproduction of gestures using a probabilistic framework combining pca, ica and hmm. In *Proceedings of the International Conference on Machine Learning (ICML),*, pages 105–112, 2005.

[10] S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 255–262, 2007.

[11] C. Chen and C. Huang. Hand gesture recognition using a real-time tracking method and hidden markov models. *Image and Video Computing,*, 21(8):745–758, August 2003.

[12] B. Clarkson and A. Pentland. Unsupervised clustering of ambulatory audio and video. In *Proceedings of the Acoustics, Speech, and Signal Processing*, volume 06. IEEE Computer Society, 1999.

[13] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. . *Journal of the Royal Statistical Society, Series B*, 39(1):(1):1–38, 1977.

[14] Markus Falkhausen, Herbert Reininger, and Dietrich Wolf. Calculation of distance measures between hidden markov models. In *Proc. of Eurospeech 95*, pages 1487–1490, 1995.

[15] S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32:41–62, 1998.

[16] A. Galata, N. Johnson, and D. Hogg. Learning variable-length markov models of behavior. *Computer Vision and Image Understanding*, 81(3), March 2001.

[17] Z. Ghahramani and M. I. Jordan. Factorial hidden markov models. *Machine Learning*, 29(2-3):245–273, November 1997.

[18] Jacob Goldberger and Sam Roweis. Hierarchical clustering of a mixture model. *Proc. of Neural Information Processing*, pages 505–512, 2004.

[19] Hidden markov model toolkit (htk). http://htk.eng.cam.ac.uk/.

[20] J. Hu, Brown. M. K., and W. Turin. Hmm based on-line handwriting recognition. *EEE Transactions on Pattern Analysis and Machine Intelligence*, 18(10):1039 – 1045, October 1996.

[21] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report 148,, German National Research Center for Information Technology,, 2001.

[22] A. James and S. Sebe. Multimodal human computer ineraction: A survey. *Computer Vision and Image Understanding*, Volume 108 , Issue 1-2:116–134, October 2007.

[23] M. I. Jordan, L. K. Saul, and G. Zoubin. Hidden markov decision trees. *Advances in neural information processing system*, 1997.

[24] A. Just, O. Bernier, and S. Marcel. Hmm and iohmm for the recognition of mono- and bi-manual 3d hand gestures. In *In Proceedings of the British Machine Vision Conference (BMVC)*, 2004.

[25] B. Kisacanin, V. Pavlovic, and T. S. Huang. *Real Time Vision for Human-Computer Interaction*. Springer-Verlag, 2005.

[26] J. Kleinberg. An impossibility theorem for clustering. In *Advances in Neural Information Processing Systems (NIPS) 15*. NIPS, 2002.

[27] B. Knab, B. Wichern, A. Schliep, B. Steckemetz, and S. Meller. General hidden markov model library (ghmm). http://ghmm.sourceforge.net/.

[28] Dana Kulic, Wataru Takano, and Yoshihiko Nakamura. Incremental on-line hierarchical clustering of whole body motion patterns. In *16th IEEE International Conference on Robot & Human Interactive Communication*, pages 1016–1021, August 2007.

[29] R. Li, M. Yang, S. Sclaroff, and T. Tian. Monocular tracking of 3d human motion with a coordinated mixture of factor analyzer. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 137–150, 2006.

[30] R. Liang and Ouhyoung. M. A real-time continuous gesture recognition system for sign language. In *Proceedings of the Third IEEE International Conference on Automatic Face and Gesture Recognition*, 1998.

[31] M. Lukosecicius and H. Jaeger. Overview of reservoir recipes. Technical Report 11, Jacobs University, 2007.

[32] S. Marcel. Gestures for multi-modal interfaces: A review. Technical Report 02-34, IDIAP-PR, 2002.

[33] G. McLachlan, , and T. Krishnan. *The EM algorithm and extensions*. Wiley series in probability and statistics. John Wiley & Sons. John Wiley & Sons., 1997.

[34] T. Minka. Expectation-maximization as lower bound maximization. Tutorial published on the web at http://www-white.media.mit.edu/ tpminka /papers/em.html, 1998.

[35] T. M. Mitchell. *Machine Learning*. MCGraw Hill Internation Editions, 1997.

[36] K. Murphy. Hidden markov model (hmm) toolbox for matlab. http://www.cs.ubc.ca/ murphyk/Software/HMM/hmm.html, 1998.

[37] V.I. Pavlovic, R. Sharma, and T.S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):677–695, 1997.

[38] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proc. of the IEEE,*, pages 257–286,, 1989.

[39] G. Rigoll, A. Kosmala, and S. Eickeler. High performance real-time gesture recognition using hidden markov models. In *Gesture and Sign Language in Human-Computer Interaction: International Gesture Workshop*, Lecture Notes in Computer Science, 1997.

[40] Y. Singer and N. Tishby. Dynamical encoding of cursive handwriting. *Biological Cybernetics*, 71(3):227–237, 1994.

[41] Marios Skounakis, Mark Craven, and Soumya Ray. Hierarchical hidden markov models for information extraction. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[42] T. Starner and A. Pentland. Real-time american sign language recognition from video using hidden markov models. In *IEEE International Symposium on Computer Vision,*, pages 265–270, 1995.

[43] A. Stolcke, , and S.M. Omohundro. Hidden markov model induction by bayesian model merging. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, 5, pages 11–18. Morgan Kaufman, 1993.

[44] A. Stolcke and S.M. Omohundro. Best-first model merging for hidden markov model induction. Technical Report TR-94-003, International Computerscience Institute, Berkley, California, 1994.

[45] Wataru Takano. *Stochastic segmentation, proto-symbol coding and clustering of motion patterns and their application to significant communication between man and humanoid robot.* PhD thesis, University of Tokyo, 2006.

[46] M. Turk. *Handbook of Virtual Enviroment Technology*, chapter Gesture Recognition. Lawrence Erlbaum Associates, 2001.

[47] Tian-Shu Wang, Heung-Yeung Shum, Ying-Qing Xu, and Nan-Ning Zheng. Unsupervised analysis of human gestures. In *Lecture Notes In Computer Science*, volume 2195 of *Proceedings of the Second IEEE Pacific Rim Conference on Multimedia: Advances in Multimedia Information Processing*, pages 147–181. Springer-Verlag, 2001.

[48] A. Webb. *Statistical Pattern Recognition, John Wiley and Sons Ltd.* John Wiley and Sons Ltd., 2002.

[49] T. Westeyn, H. Brashear, A. Atrash, and T. Starner. Georgia tech gesture toolkit: supporting experiments in gesture recognition. In *Proceedings of the 5th international conference on Multimodal interface*, Multimodal architectures and frameworks, pages 85 – 92, 2003.

[50] A. D. Wilson and F. Bobick. Parametric hidden markov models for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):884–900, September 1999.

[51] Y. Wu and T.S. Huang. *Gesture-Based Communication in Human-Computer Interaction Vision-Based Gesture Recognition: A Review*, chapter Gesture-Based Communication in Human-Computer Interaction. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1999.

[52] Lexing Xie, Shih-Fu Chang, Ajay Divakaran, and Huifang Sun. Unsupervised discovery of multilevel statistical video structures using hierarchical hidden markov models. In *Multimedia and Expo*, volume 3 of *03*, pages 29–32. IEEE, July 2003.

Chapter 7

Appendix

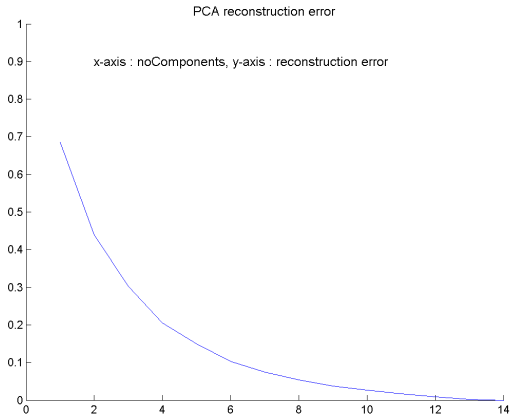
7.1 Appendix HISAA

7.1.1 Algorithms

7.1.2 Results

Figure 7.1 shows the PCA reconstruction error of the "moderate" data subset which is used in our experiments.Table 7.1 gives statistics for the lengths of the different gestures in the database.

Figure 7.1: PCA Reconstruction error for "moderate" subset



Algorithm 7.1.1: CycleDistanceMatrix()

Inputs: TransitionMatrix, transitionThreshold

$noStates \leftarrow size(TransitionMatrix, 2)$
$\forall_{i \epsilon \{1 \ldots n\}} \ : \ TransitionMatrix(i,i) = 0$
NORMALIZE($TransitionMatrix$)
$\forall_{i,j \epsilon \{0 \ldots (n-1)\}} \ :$
$\begin{cases} \textbf{if } TransitionMatrix(i,j) \leq transitionThreshold \\ \quad \textbf{then } TransitionMatrix(i,j) \leftarrow 1 * 10^{-20} \end{cases}$
$\forall_{i \epsilon \{1 \ldots n\}} \ : \ TransitionMatrix(i,i) = 1$

$\forall_{i,j \epsilon \{0 \ldots (n-1)\}} :$
$\begin{cases} probIJ \leftarrow \text{VITERBIPATH}(TransitionMatrix, i, j) \\ probJI \leftarrow \text{VITERBIPATH}(TransitionMatrix, j, i) \\ cycleDistance = (1/(probIJ * probJI)) - 1; \\ CycleDistanceMatrix(i,j) \leftarrow cycleDistance \end{cases}$
return ($CycleDistanceMatrix$)

Algorithm 7.1.2: FindStrongCycles()

Inputs: group
comment: Returns list of lists of states belonging
to strong cycles

if $size(group.elements) < 2$
  then $return\{\}$
$cycles \leftarrow \{\}$
if $group.distance > 10^{-20}$
  then $cycles = cycles \cup \{group.elements\}$
  else $\begin{cases} \forall_{child \ \epsilon \ \text{CHILDREN}(group)} \ : \\ \{cycles = cycles \cup \text{FINDSTRONGCYCLES}(child) \end{cases}$
return ($strongCycles$)

7.2 Appendix HMM Compression and Merging

7.2.1 Complexity Analysis

Baum-Welch Algorithm

The Baum-Welch algorithm is based on the computation of two functions, known as Forward and Backward probabilities, (i, t) and (i, t), for each state i [1, N] of an HMM and each frame t ϵ [1, T] of an observation sequence $O = O_1, O_2, ..., O_T$. Computing these functions yields a complexity of order $O(N^2T)$. Once computed, Forward and Backward probabilities are used to weight the contributions of each observation O_t to the HMM parameters.

If L observation sequences $O(l) = O_1, O_2, ..., O_{T_l}$, with $l = 1, 2, ..., L$, are explicitly available for an HMM, the resulting procedure is known as single

Algorithm 7.1.3: FINDLONGESTSTRONGTRAJECTORIES()

Inputs: TransMat,transThresh
comment: Returns set of ordered lists
corresponding to trajectories

$noStates \leftarrow size(Transmat, 2)$
$states \leftarrow \{1, \ldots, noStates\}$
$remainingStates \leftarrow \{1, \ldots, noStates\}$
$strongTrajectories \leftarrow \{\}$
while $remainingStates \neq \emptyset$
do
- $S_{start} \leftarrow remainingStates(1)$
- $P_{max} \leftarrow 1$
- $S_{next} \leftarrow S_{cur} \leftarrow S_{start}$
- **while** $(P_{max} > transThresh) \wedge (S_{next} \neq S_{cur})$
  - **do**
    - $S_{cur} \leftarrow S_{next}$
    - $traj \leftarrow [traj, S_{next}]$
    - $P_{max} = max_{S_{next} \epsilon states}(P(S_{next}|S_{cur}))$
    - $S_{next} = argmax_{S_{next} \epsilon states}(P(S_{next}|S_{cur}))$
- $strongTrajectories \leftarrow strongTrajectories \cup traj$
- $remainingStates \leftarrow remainingStates \cap elts(traj)$

return $(strongTrajectories)$

Algorithm 7.1.4: PARTITIONTRAJECTORIES()

Inputs: strongTrajs
Comment: Returns set of non-overlapping
ordered lists corresponding to atomic trajectories

$partTrajs \leftarrow \{\}$
while $strongTrajs \neq \emptyset$
- $traj = strongTrajs(1)$
- $overlapping \leftarrow$find strongTrajectories
  having overlapping last states with traj
- $overlappedSet \leftarrow overlapping \cap traj$
- $strongTrajs \leftarrow strongTrajs \cap overlappedSet$
- **if** $overlappedSet \neq \emptyset$
  - **then**
    - $tail \leftarrow$find longest overlapping tail which
      is part of all overlapping trajectories
    - $withoutTail \leftarrow \forall\ trajO\ \epsilon\ overlappedSet$:
      REMOVETAIL$(overlappedSet, tail)$
    - $partTrajs \leftarrow partTrajs \cup tail$
    - $strongTrajs \leftarrow strongTrajs \cup withoutTail$
  - **else**
    - $partTrajs \leftarrow partitionedTrajectories \cup traj$
    - $strongTrajs \leftarrow strongTrajs \cap traj$

return $(partitionedTrajectories)$

model Baum-Welch re-estimation [6]. This is the case for gesture recognition with labeled data. For each observation sequence O(l) , the Forward and Backward probabilities must be computed, and then various contributions

Algorithm 7.1.5: HISAA()

Inputs: hmm,cycleThresh,trajThresh

$cycDistMat \leftarrow$
  CycleDistanceMatrix($hmm.TransMat, cycleThresh$)
$group \leftarrow$ CompleteLinkClustering($cycDisteMat$)
$strongCs \leftarrow$ FindStrongCycles($group$)

$remainingStatesTransmat \leftarrow$
  getRSTransmat($TransitionMat, strongCycles$)
$overlappingStrongTrajs \leftarrow$
  LongestStrongTrajectories($TransMat, trajThresh$)
$strongTs \leftarrow$
  PartitionTrajectories($overlappingStrongTrajs$)
$singletons \leftarrow$ remainingStates($hmm, strongCs, strongTs$)
$subHMMS \leftarrow$
  getSubHMMs($hmm, strongCs, strongTs, singletons$)
return ($[strongCs, strongTs, singletons, subHMMs]$)

Table 7.1: Subsets "fast" and "moderate"

Id	"fast"	"moderate"
1	104.9 ± 7.1	127.3 ± 10.6
2	93.6 ± 13.8	121.9 ± 15.0
3	122.6 ± 26.2	109.8 ± 11.7
4	124.3 ± 14.5	152.2 ± 28.5
5	134.6 ± 26.8	267.8 ± 79.1
6	36.5 ± 2.7	53.6 ± 5.5
7	37.4 ± 4.4	73.4 ± 4.5
8	38.2 ± 2.1	61.0 ± 6.6
9	42.8 ± 4.3	84.4 ± 10.9

Subsets "slow" and subsets combined

Id	"slow"	subsets combined
1	186.1 ± 24.3	139.4 ± 38.0
2	159.1 ± 17.1	124.9 ± 31.0
3	158.6 ± 56.9	130.3 ± 41.3
4	226.7 ± 44.8	167.7 ± 53.6
5	384.7 ± 94.4	262.4 ± 125.4
6	104.4 ± 7.5	64.8 ± 29.8
7	109.6 ± 15.2	73.5 ± 31.3
8	125.0 ± 15.8	74.7 ± 38.6
9	187.0 ± 27.3	104.7 ± 63.8

and norms accumulated, which yields a computational complexity of order $O(3N^2T_l + NT_l + NT_lMD)$ for continuous HMMs, where M is the number of Gaussian components in the mixtures used to represent the output distributions in continuous HMMs, and $D = \sum_{i=1}^{R} d_i$ with d_i the dimension of the data stream i the total number of components of data vectors in the

continuous case. In the most common configurations, the last term of the summation is dominant. Summing for all the training sequences, we get complexities of order $O(3N^2T + NT + NTMD)$, respectively, where T is the length of the training database.

GMM Clustering complexity

The GMM Clustering
compression algorithm is very similar to the K-means algorithm, and consists of repeated regrouping and refitting of mixture components until convergence. In the regroup step every mixture component is compared with every component from the compressed set, to find the closest one. The distance computation itself is given by the formula

$$d(f,g) = \sum_{i=1}^{k} \alpha_i min_{k=1}^{m} KL(f_i||g_j)$$

. The KL-divergence between two Gaussians $KL(f_i||g_j)$ in this formula is comuted as.

$$\begin{aligned} &KL(N(\mu_1,\Sigma_1),N(\mu_2,\Sigma_2)) = \\ &\frac{1}{2}(log\frac{|\Sigma_2|}{|\Sigma_1|} + Tr(\Sigma_2^{-1}\Sigma_1) + \\ &(\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2) - d \end{aligned}$$

and has a complexity quadratic in the mixture dimensionality so then the regroup operation has complexity $O(C \times K \times D^2)$ with for C the number of components in the model to be compressed and K the number of components compressed to.
For the refit operation, we computed a weighted combination of the means and covariance matrices of all components that are compressed together. The formulas for those computations are given as: $w_r^j = \sum_{i\epsilon\pi^{-1}(j)} w_c^i$
$\widehat{\mu}_r^j = \frac{\sum_{i\epsilon\pi^{-1}(j)} w_c^i \mu_c^i}{\widehat{w}_j^r}$
$\widehat{\Sigma}_r^j = \frac{\sum_{i\epsilon\pi^{-1}(j)} w_c^i (\Sigma_c^i + (\mu_c^i - \widehat{\mu}_r^j)(\mu_c^i - \widehat{\mu}_r^j)^T)}{\widehat{w}_j^r}$

In these formulas $i\epsilon\pi^{-1}(j)$ is the set of mixture components that project onto component j in M_r

We can see that these computations have a complexity that is quadratically dependent on the dimensionality of the mixture components that are combined and linearly dependent on the number of old and reduced mixtures, hence the total complexity of refitting is also $O(C \times K \times D^2)$.

The entire algorithm, which combines the refit and regroup steps, then also has complexity $O(C \times K \times D^2)$; since the algorithm usually converges so fast that the iterative nature does not add an extra factor to the complexity.

Direct HMM Compression complexity

The HMM compression algorithm has the GMM Clustering as it's core, but furthermore also has to recompute a transition matrix, mixture matrices and priors. However, those operations have complexity of at most $O(C \times K)$, so they do not increase the complexity of HMM Compression over GMM Clustering. In other words, our HMM Compression algorithm has complexity $O(C \times K \times D^2)$, that is linear in the number of original and reduced components C and K, and quadratic in the mixture component dimensionality D. When we compare this with the complexity of Baum-Welch, we notice that it has no dependence on the size of a training set, but only depends on the number of states and the dimensionality of the mixture components. Hence our Direct compression Algorithm has a big computational advantage over Compression through retraining by data re-sampling, since in that method the computational cost there grows linearly with the amount of regenerated data, and thus becomes quite costly as the amount of data used for retraining gets large.

7.2.2 Algorithm Pseudocode

Algorithm 7.2.1: MERGEHMMS($hmm1, hmm2$)

Comment: Returns a merged hmm

$s1 = size(hmm1.transmat, 1)$
$s2 = size(hmm2.transmat, 2)$

$mid = s1 + 1$
$last = s1 + s2$
$nStates \leftarrow$ APPEND($hmm1.states, hmm2.states$)
$nMixmat \leftarrow$ APPEND($hmm1.mixmat, hmm2.mixmat$)
$nPriors \leftarrow$ APPEND($hmm1.priors, hmm2.priors$)

$nTransmat \leftarrow zeros(noStates1 + noStates2)$
$nTransmat(1 : s1, 1 : s1) = hmm1.transmat;$
$nTransmat((mid : last, mid : last) = hmm2.transmat;$

$appHMM \leftarrow$
  NEWHMM($nStates, nTransmat, nPriors, nMixMat$)
$mergedHMM \leftarrow$ COMPRESSHMM($appHMM$)
return ($mergedHMM$)

Algorithm 7.2.2: CompressHMM($hmm, numNewStates$)

Comment: Returns a compressed HMM

$numStates \leftarrow size(hmm.transmat, 2)$
$gmHMM \leftarrow$ ExtractGMHMM(hmm)
$[mapping, newC] \leftarrow$
  CompressGMM($gmHMM.states$, UniformWs($noStates$))
$[newStates, compMixMat] \leftarrow$
  ComputeNewStates($hmm, mapping, numNewStates$)
$[cTransmat, cPrior] \leftarrow$
  CompTransAndP($hmm.transmat, hmm.priors, mapping$)
$compHMM \leftarrow$
  newHMM($newStates, cTransmat, cPrior, compMixMat$)
return ($partitionedTrajectories$)

Algorithm 7.2.3: ExtractGMHMM(hmm)

Comment: Returns a GM-HMM by summing the
mixture components for every state

$newHMM \leftarrow hmm$
for $i \leftarrow 1$ **to** $noStates$
$\begin{cases} s \leftarrow hmm.states(i) \\ newHMM.states(i) \leftarrow \\ \quad \sum_{j=1}^{numMixt} hmm.mixmat(i,j) \times s.mixtureComps(i) \end{cases}$
return ($newHMM$)

Algorithm 7.2.4: compressGMM()

Inputs: gmmComps,numNewComp, weights
Comment: Returns a compressed GMM by iterative
regrouping and refitting similar to K-Means
nAS : the new assignments, nC = the new components
nW : the weights of the new components

$newHMM \leftarrow hmm$
$numOldComp \leftarrow size(gmmComponents, 2)$
$oldAssignments \leftarrow []$
$nAS \leftarrow initAssignments(numOldComp, numNComp)$
while $assignments \neq oldAssignments$
$\begin{cases} oldAssignments \leftarrow assignments \\ [nC, nW] \leftarrow \\ \quad \text{REFIT-MIXTURE}(gmmComps, assignments, weights) \\ nAS \leftarrow \text{REGROUP-COMPONENTS}(gmmComps, nC) \end{cases}$
return ($[nAS, nC, nW]$)

Algorithm 7.2.5: COMPUTENEWSTATES()

Inputs: hmm,assignments,numNewStates
Comment: Returns states: Mixtures of Gaussians
and mixmat: Matrix of mixture weights

for $i = 1$ **to** $numNewStates$
$\begin{cases} statesComb \leftarrow \cup_{state\ :\ assignments(state)=i} \\ compComb \leftarrow \text{COLLECTCOMPONENTS}(statesCombined) \\ weightsCC \leftarrow \text{COMPWEIGHTS}(hmm.priors, hmm.mixat) \\ [,newC, weights] \leftarrow \\ \quad \text{COMPRESSGMM}(compComb, numMixt, weightsCC) \\ states(i) \leftarrow newC \\ mixmat(:, j) = weights \end{cases}$
return $(states, mixmat)$

Algorithm 7.2.6: COMPTRANSANDP()

Inputs: transmat,priors,assignments
Comment: Returns new, compressed transmat and priors

for $i = 1$ **to** $numNewComponents$
$\quad cpTransmat1(:, i) \leftarrow \sum_{j:assignments(j)=i} transmat(:, j)$
for $i = 1$ **to** $numNewComponents$
$\begin{cases} cpTransmat2(i, :) \leftarrow \\ \quad \sum_{assignments(j)=i} cpTransmat1(j, :) \\ cpPriors(i) \leftarrow \sum_{assignments(j)=i} priors(j) \end{cases}$
$cpTransmat = \text{NORMALISE}(cpTransmat2, 2)$
return $([cpTransmat, cpPriors])$

7.2.3 Remaining Results

Figure 7.2 and 7.3 give the PCA reconstruction Error for the Training sets used in the Merge experiments. Table 5.9 gives the results for the compression test, in which the accuracy of the direct compression method is compared with the accuracy of directly learning a model from the original data or compressing by data re-sampling with a sample size of 20 or 50 data samples per compressed HMM respectively. Every sample with a fixed length equal to the mean example length of the training sets. Since all gesture signals in the Interactplay database have very similar lengths, this is a reasonable approach.

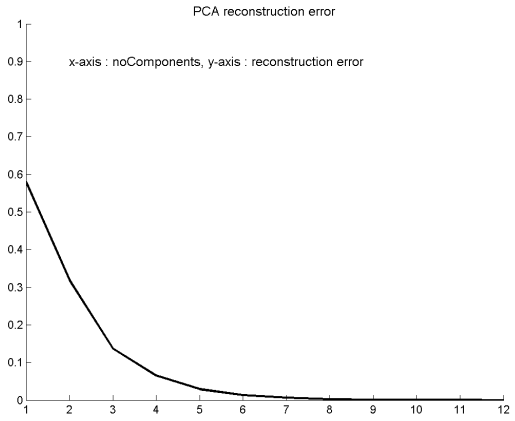


Figure 7.2: PCA Reconstruction Error, Data Person 1, sessions 1,2,3,4

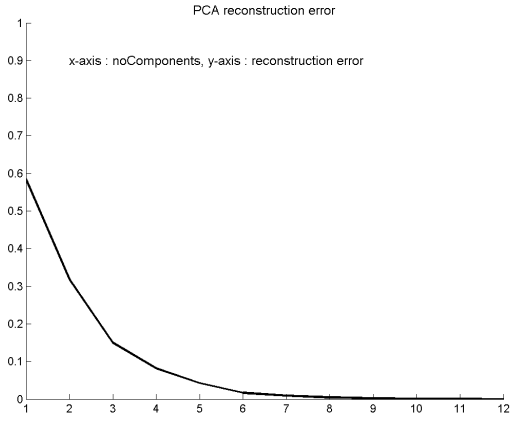


Figure 7.3: PCA Reconstruction Error, Data Person 1 and 2, all sessions