Fast State Discovery for HMM Model Selection and Learning

Sajid M. Siddiqi
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Geoffrey J. Gordon
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213

Andrew W. Moore
Google, Inc.
Pittsburgh, PA 15213

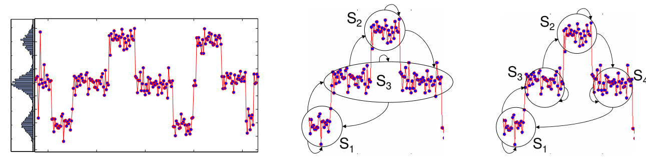
Abstract

Choosing the number of hidden states and their topology (model selection) and estimating model parameters (learning) are important problems for *Hidden Markov Models*. This paper presents a new state-splitting algorithm that addresses both these problems. The algorithm models more information about the dynamic context of a state during a split, enabling it to discover underlying states more effectively. Compared to previous top-down methods, the algorithm also touches a smaller fraction of the data per split, leading to faster model search and selection. Because of its efficiency and ability to avoid local minima, the state-splitting approach is a good way to learn HMMs even if the desired number of states is known beforehand. We compare our approach to previous work on synthetic data as well as several real-world data sets from the literature, revealing significant improvements in efficiency and test-set likelihoods. We also compare to previous algorithms on a sign-language recognition task, with positive results.

1 Introduction

Consider an observation sequence where the observation is correlated to the dynamics of an unobserved state variable, as in Figure 1(A). Hidden Markov Models (HMMs) are a popular tool for modeling the statistical properties of such sequences, which are ubiquitous in the real world. HMMs have been used extensively in speech recognition (Rabiner, 1989), bioinformatics (Krogh et al., 1994), information extraction (Seymore et al., 1999) and other areas.

There has been extensive work on learning the parameters of a fixed-topology HMM. Several algorithms



(A) (B) (C)

Figure 1: A. A time series from a 4-state HMM. Observations from the two states *down-middle-up* and *up-middle-down* overlap and are indistinguishable without temporal information. B. The HMM topology learned by ML-SSS and Li-Biswas on the data in A. C. The correct HMM topology, successfully learned by the STACS algorithm.

for finding a good number of states and corresponding topology have been investigated as well, but none of these are used in standard practice. Normally, the topology of an HMM is chosen *a priori* and a hill-climbing method is used to determine parameter settings. For model selection, several HMMs are typically trained with different numbers of states and the best of these is chosen. There are two problems with this approach: firstly, training an HMM from scratch for each feasible topology may be computationally expensive. Secondly, since parameter learning is prone to local minima, we may inadvertently end up comparing a 'good' N_1 state HMM to a 'bad' N_2 state HMM. The standard solution to this is to train several HMMs for each topology with different parameter initializations in order to overcome local minima, which however compounds the computational cost.

Because of these issues, many researchers have previously investigated top-down state-splitting methods as an appealing choice for topology learning in HMMs with continuous observation densities. This paper proposes *Simultaneous Temporal and Contextual Splitting* (STACS), a new top-down model selection and learning algorithm that constructs an HMM by alternating between parameter learning and model selection while incrementally increasing the number of states. Candidate models are generated by splitting existing states and optimizing relevant parameters, and are then evaluated for possible selection. Unlike previous methods, however, the splitting is carried out in a way that accounts for both *contextual* (observation density) and *temporal* (transition model) structure in the underlying data in a more general manner than the state-splitting methods mentioned above, which fail on the simple example in Figure 1(A). In this research, we closely examine these competing methods and illustrate the key differences between them and STACS, followed by an extensive empirical comparison. Since hard-updates training is a widely used alternative to soft-updates methods in HMMs because of its efficiency, we also examine a hard-updates version of our algorithm, *Viterbi STACS* (V-STACS), and explore its pros and cons for model selection and learning with respect to the soft-updates method. We also evaluate STACS as an alternative learning algorithm for models of *predetermined* size.

It should also be noted that STACS can be generalized to model selection in Dynamic Bayesian Networks or other directed graphical models that have hidden states of undetermined cardinality, since the sum-product and max-product algorithms for inference and learning in these models are generalizations of the Baum-Welch and Viterbi algorithms for HMMs.

2 Preliminaries

In defining HMMs, we will use the notation and terminology of Rabiner (1989). Let $O_1, ..., O_T$ be a sequence of M-dimensional observations of size T. An N-state HMM λ is characterized by its $N \times N$ transition matrix, observation model and $N \times 1$ prior distribution vector. Let q_t denote the state the HMM is in at time t, which can hold values from $s_1, \ldots, s_N$.The transition matrix is represented as $A = \{a_{ij}\}$, where $a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i)$. The observation model is $B = \{b_j(x)\}$ where $b_j(x) = P(O_t = x \mid q_t = s_j)$, the observation probability distribution in state j. $\pi = \{\pi_i\}$ is the initial state probability distribution, where $\pi_i = P(q_1 = s_i)$. The posterior distributions over state occupancy and transitions are typically denoted by $\gamma_t(s) = P(q_t = s \mid O_{1:T})$ and $\xi_t(s, s') = P(q_t = s, q_{t+1} = s' \mid O_{1:T})$, respectively.

Standard HMM algorithms for path inference and parameter learning are all $O(TN^2)$, though faster HMM algorithms for large state-spaces have been explored in Felzenszwalb et al. (2003) and Siddiqi and Moore (2005). The *Viterbi algorithm* (Viterbi, 1967) is a dynamic programming algorithm for finding the optimal

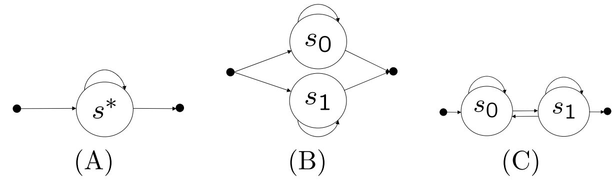


Figure 2: An illustration of the overly restrictive splits in ML-SSS. A. Original un-split state s^*. B. A *contextual split* of s^* in the ML-SSS algorithm. States s_0 and s_1 must have the *same* transition structures and different observation models. C. A *temporal split*. State s_0 has the incoming transition model of s^* and s_1 has its outgoing ones.

hidden state sequence. *Baum-Welch* (Rabiner, 1989) is an Expectation-Maximization (EM) algorithm for finding parameters that locally maximize $P(O \mid \lambda)$, and *Viterbi Training* is the equivalent hard-updates learning algorithm which is faster by a large constant factor but also less precise since it assumes the data at each timestep is from a single hidden state. Viterbi training yields parameters that maximize the complete likelihood $P(O, Q \mid \lambda)$, where Q is a sequence of hidden states that corresponds to the observed data sequence.

To determine the stopping point for state-splitting, we use the *Bayesian Information Criterion* (Schwarz, 1978), or BIC score, which asymptotically converges to the true posterior probability of the model assuming an uninformative prior.

3 Related Work

There has been extensive work on HMM model selection. However, most of this work is either tailored to a specific application or is not scalable to learning topologies with more than a few states. The most successful approaches are greedy algorithms that are either bottom-up (i.e., starting with an overly large number of states) or top-down (i.e., starting with a small or single-state model). One disadvantage of bottom-up approaches is having to know an upper bound on the number of states beforehand. In some cases, one also faces the problem of deciding which of N^2 pairs of states to merge. We therefore favor top-down methods for HMM model selection, especially when the number of states may be large. We define some terminology first: *split design* refers to the process of splitting an HMM state, optimizing parameters and creating an HMM for possible selection. HMMs created by designing different splits are called *candidates*. The two major alternative top-down approaches can be summarized as follows:

Li-Biswas: Li and Biswas (1999) decrease the number of model selection candidates to two: splitting the state with largest observation density variance, and merging the two states with closest means. These candidates are then optimized with EM on the entire HMM. The candidate with better likelihood is chosen at each step, terminating when the candidates are worse than the original model. The primary drawback with this heuristic is that it ignores dynamic structure while deciding which states to split: a single low-variance state might actually be masking two Markov states with overlapping densities, making it a better split candidate. Training two candidates with full EM is also inefficient especially when they may not be the best candidates, as our empirical evaluations will show.

ML-SSS: *Maximum-Likelihood Successive-State-Splitting* (Ostendorf & Singer, 1997) is designed to learn HMMs that model *contextual* (observation density) and *temporal* (transition model) variation in phones for continuous speech recognition systems. ML-SSS incrementally builds an HMM by splitting one state at a time, considering all N possible splits as candidates in each timestep. However, instead of full EM for each candidate, the split on state s^* into s_0 and s_1 (figure 2) is trained by a constrained iterative optimization of the expected likelihood gain from performing the split, while holding all $\gamma_t(s)$ and $\xi_t(s, s')$ constant for $s, s' \neq s^*$. The iterations are performed over all data points t with non-zero posterior occupancy probability. Each state is considered for two kinds of splits, a *contextual split* (Figure 2(B)) that optimizes only observation densities, and a *temporal split* (figure 2(C)) that also optimizes self-transition and inter-split-state transition probabilities.

Though more efficient than brute-force and Li-Biswas, the fact that ML-SSS does not model transitions to and from other states while splitting makes it fail to detect underlying Markov states with overlapping densities. For example, ML-SSS with BIC converges on the HMM in figure 1(B) while the true underlying HMM, successfully found by STACS, is shown in figure 1(C). Also, having to consider all data points with non-zero posterior probability $\gamma_t(s^*)$ per split is expensive in dense data with overlapping densities.

4 Simultaneous Temporal and Contextual Splits

The primary idea behind STACS is to trade off the amount of contextual information considered per split in return for increased modeling of temporal information. Here, contextual information is quantified by the number of data points considered during split design, and temporal information by the number of transition parameters optimized.

Specifically, we model the posterior belief at each timestep by a delta function at the Viterbi-optimal state. Each split is optimized over data points with non-zero belief in this approximated posterior, which number T/N on average. This is in contrast to ML-SSS where each split could consider $O(T)$ timesteps. Let T^s denote the set of timesteps owned by state s in the Viterbi path. We choose the split that maximizes the *partially observed likelihood* $P(O, Q^*_{\setminus T^s} \mid \lambda)$ at each model selection step, with all timesteps fixed at their optimal path states except those in T^s.

The increase in efficiency afforded by this simplification allows us to model more dynamic structure during splitting in two ways that are distinct from ML-SSS. Firstly, our split design operation updates all transition parameters related to a state, both incoming and outgoing, rather than just the inter-split-state and self-transition parameters. Secondly, we can afford to run our split design operation until convergence rather than for a fixed number of iterations as in ML-SSS. We experimentally found both these enhancements to be essential in achieving good splits on states with overlapping densities, such as in Figure 1. We also eliminate the distinction between temporal and contextual splits, performing a single split operation per state and allowing it to model both contextual and temporal HMM parameters.

4.1 Splitting Algorithms

We now describe the hard-updates and soft-updates splitting techniques used by V-STACS and STACS. First, some notation: when considering a split of state s, HMM parameters related to state s (denoted by λ_s), including incoming and outgoing transition probabilities, are replaced by parameters for two offspring states s_1 and s_2 (denoted by λ_{s_1,s_2}). The set of observation indices assigned to state s, denoted by T^s, are now assumed to have unknown hidden state values, but only in the restricted state space $\{s_1, s_2\}$. Therefore when searching for a locally optimal candidate, only the parameters λ_{s_1,s_2} change, and the only observations that will affect them are those at timesteps T^s, i.e., O_{T^s}. Let $\lambda_{\setminus s}$ denote parameters not related to state s.

4.1.1 Method 1: Split-State Viterbi Training

For split design, V-STACS uses *Split-State Viterbi Training*, which learns locally optimal values for the parameters λ_{s_1,s_2} by alternating the following two steps until convergence:

1. E-step:
   $Q^v_{T^s} \leftarrow \arg\max_Q P(O_{T^s}, Q^*_{\setminus T^s}, Q \mid \lambda_{\setminus s}, \lambda^v_{s_1,s_2})$
2. M-step:
   $\lambda^{v+1}_{s_1,s_2} \leftarrow \arg\max_\lambda P(O_{T^s}, Q^*_{\setminus T^s}, Q^v_{T^s} \mid \lambda_{\setminus s}, \lambda)$

Here, $Q^*_{\setminus T^s}$ denotes the base model Viterbi path excluding timesteps belonging to state s. The first step above computes an optimal path through the split-state space. The second step updates the relevant HMM parameters with their MLE estimates for a fully observed path, which are simply ratios of counts for transition parameters, and averages of subsets of O_{T^s} for the observation parameters. Convergence occurs when the state assignments on T^s stop changing. The E-step of Split-State Viterbi Training is carried out using an adaptation of Viterbi (called *Split-State Viterbi*, pseudocode in appendix) to the task of finding an optimal path over a binary state space through a subset of the data while constraining the rest to specific states.

The running time is clearly linear in the number of non-determined timesteps $|T^s|$ since each maximization in the algorithm is always over 2 elements no matter how large the actual HMM gets. Note that we allow the incoming and outgoing transition parameters of s_1, s_2 to be updated as well, which allows better modeling of dynamic structure during split design.

4.1.2 Method 2: Split-State Baum-Welch

Split-State Baum-Welch also learns locally optimal values for the state-split parameters λ_{s_1,s_2}, and like Baum-Welch it does so by modeling the posterior over the hidden state space which in this case consists of $\{s_1, s_2\}$. The following two steps are iterated:

1. E-step: Calculate stepwise occupancy and transition probabilities $\{\gamma^s, \xi^s\}$ from $\{\lambda_{\setminus T^s}, \lambda^v_{s_1,s_2}, O_{T^s}, Q^*_{\setminus T^s}\}$, compute expectations.
2. M-step: $\{\lambda_{s_1,s_2}\}^{v+1} \leftarrow \arg\max_\lambda P(O_{T^s}, Q^*_{\setminus T^s} \mid \lambda_{\setminus s}, \lambda)$

The E-step step is carried out using a specialized two-state partially-constrained-path version of the Forward-Backward algorithm to first calculate the forward and backward variables, which then give the required γ^s and ξ^s values for s_1 and s_2. The idea is the same as Split-State Viterbi but with soft counts and updates. The entire algorithm is $O(N \mid T^s \mid)$, since the update step requires summations over all observations in T^s for at least N transition parameters.

It is important to note that, though slower, Split-State Baum-Welch searches a larger space of candidate models than Split-State Viterbi Training, performing better on 'difficult' splits with high hidden variable entropy just as Baum-Welch performs better than Viterbi Training in such situations. Like the temporal split in ML-SSS, it is not possible to compute the updated overall likelihood from this split algorithm, so convergence is heuristically based on differences in the split-state α variables in successive timesteps.

4.2 Efficient Design and Scoring of Candidates

So far we have been successful in keeping the candidate search process an efficient $O(NT)$ procedure, since each split design operation is at most $O(N \mid T^s \mid)$, $\mid T^s \mid$ is $O(T/N)$ amortized, and we perform N split design operations per timestep. Note that ML-SSS also achieves $O(NT)$ splits but via a different route: each of N split-designs is $O(T)$ because incoming and outgoing transitions are not learned, and each state can have $O(T)$ data points with non-zero posterior probability.

To keep the overall algorithm efficient, we first choose the best candidate according to a fast-to-compute criterion, the updated Viterbi likelihood after optimizing the split parameters. We then compare this single candidate to the base model using BIC. Other approximations to the Bayes factor would also work. Computing the likelihood for BIC is an $O(TN^2)$ task, but we can afford this expense since we only have to do it for one candidate model. In V-STACS , the likelihood is approximated by the Viterbi path likelihood as in the *Viterbi approximation* (Ney, 1990). This avoids $O(TN^2)$ operations in V-STACS since the Viterbi path can be updated efficiently after split design.

5 Experiments

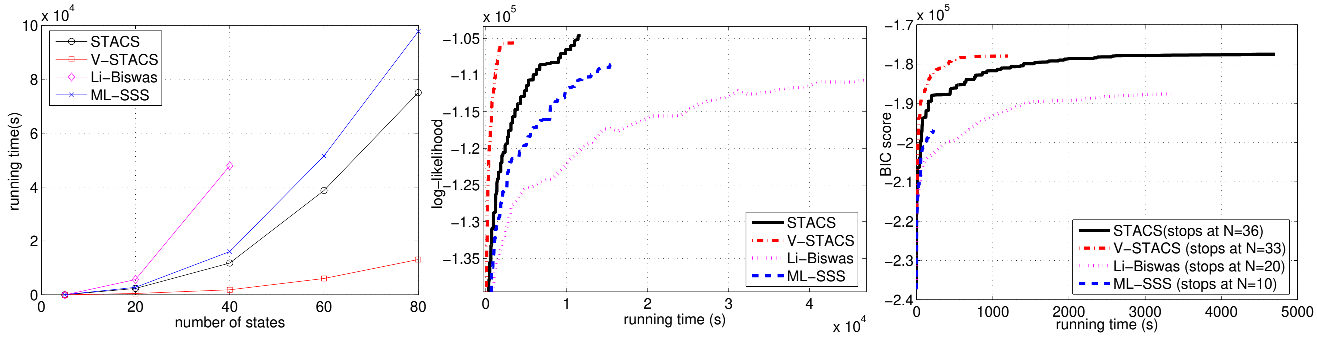
In our experiments we seek to compare STACS and V-STACS to Li-Biswas, ML-SSS, and multi-restart Baum-Welch, in these areas:

- Learning models of predetermined size
- Model selection capability
- Classification accuracy

We are concerned both with the quality of models learned, as indicated by test-set likelihoods and BIC scores, as well as running-time efficiency.

5.1 Algorithms and Data Sets

As described earlier, STACS uses Baum-Welch for parameter learning and Split-State Baum-Welch for split design. V-STACS uses Viterbi Training and Split-State Viterbi Training, followed by Baum-Welch on the final model. ML-SSS (with a tweak for generalizing to non-chain topologies) and Li-Biswas were implemented for comparison.



(A) (B) (C)

Figure 3: A. Running time vs. number of final states on the ROBOT data set. B. Log-Likelihood vs. running time for learning a 40-state model on the ROBOT data. C. BIC score vs. running time on the MOCAP data when allowed to stop splitting autonomously. The plots are representative.

Table 1: Test-set log-likelihoods (scaled by dataset size) and training times of HMMs learned using STACS,V-STACS, ML-SSS and regular Baum-Welch with 5 random restarts. The best score and fastest time in each row are highlighted. Li-Biswas had similar results as ML-SSS, and slower running times, for those N where it completed successfully.

Data	STACS	V-STACS	ML-SSS	Baum-Welch	STACS	V-STACS	ML-SSS	Baum-Welch
	$N = 5$				$N = 40$			
ROBOT	-2.41	**-2.41**	-2.44	-2.47	**-1.75**	-1.76	-1.80	-1.78
	40s	***13s***	*70s*	*99s*	*11790s*	***1875s***	*16048s*	*18460s*
MOCAP	-4.46	-4.46	-4.49	-4.46	-4.32	**-4.29**	-4.30	-4.37
	34s	***14s***	*49s*	*65s*	*5474s*	***1053s***	*6430s*	*7315s*
MLOG	-8.78	**-8.78**	-10.49	-8.78	**-8.25**	-8.26	-10.49	-8.38
	67s	*15s*	***9s***	*750s*	*29965s*	*8146s*	***1818s***	*42250s*
AUSL	-3.60	-3.60	-3.60	**-3.43**	-2.89	**-2.77**	-3.08	-2.99
	39s	***14s***	*33s*	*110s*	*7923s*	***1550s***	*8465s*	*22145s*
VOWEL	-4.69	-4.69	-4.68	**-4.67**	-4.34	**-4.32**	-4.44	-4.33
	13s	***8s***	*37s*	*95s*	*2710s*	***1011s***	*2874s*	*6800s*
	$N = 20$				$N = 60$			
ROBOT	**-1.93**	-1.93	-1.98	-1.96	-1.65	**-1.64**	-1.69	-1.75
	2368s	***512s***	*2804s*	*4890s*	*38696s*	***6086s***	*51527s*	*35265s*
MOCAP	-4.38	-4.37	-4.33	-4.33	**-4.23**	-4.26	-4.23	-4.46
	899s	***203s***	*800s*	*3085s*	*16889s*	***3470s***	*18498s*	*20950s*
MLOG	**-8.34**	-8.34	-10.49	-8.40	-8.25	**-8.23**	-8.29	-8.39
	3209s	*1173s*	***284s***	*12350s*	*116891s*	***29379s***	*108358s*	*87150s*
AUSL	-3.16	-3.18	-3.21	**-3.13**	**-2.71**	-2.71	-2.86	-2.89
	1128s	***284s***	*1410s*	*3655s*	*23699s*	***4613s***	*25156s*	*60035s*
VOWEL	**-4.40**	-4.41	-4.44	-4.41	**-4.30**	-4.31	-4.44	-4.31
	548s	***189s***	*1009s*	*1285s*	*8296s*	***2714s***	*4407s*	*13360s*

We choose a range of real-world data that has appeared in previous work on sequential data models. These are ROBOT, MOCAP, MLOG, AUSL and VOWEL. ROBOT (Howard & Roy, 2003) contains laser readings from a Pioneer robot moving indoors. MOCAP(Ren et al., 2005) contains motion capture data from people performing various actions.AUSL and VOWEL are from the UCI KDD archive (D.J. Newman & Merz, 1998). We also use AUSL for measuring classification accuracy. MLOG was previously used in a paper on large HMMs (Siddiqi & Moore, 2005).

5.2 Learning HMMs of Predetermined Size

We first evaluate performance in learning models of predetermined size. In Table 1 we show test-set log-likelihoods normalized by data set size along with running times for experiments using STACS and V-STACS along with ML-SSS and regular Baum-Welch with 5 restarts. Here we ignore the stopping criterion and perform the best split at each model selection step until N is reached. For Baum-Welch, the best score from its five runs is given along with the total time. Li-Biswas results are not shown because most desired model sizes were not reached. However, the instances that did successfully complete indicate that Li-Biswas is much slower than any other method considered, even Baum-Welch, while learning models with similar scores as ML-SSS.

We note that STACS and V-STACS have the fastest running times for any given HMM size and data set, except for cases when a competing algorithm got stuck and terminated prematurely. Figure 3(A) shows a typical example of STACS and V-STACS running times compared to previous methods for different N values.

As N grows larger and the possibility of local minima increases, STACS and V-STACS consistently return models with better test-set scores. Figure 3(B) shows training-set score against running time for the Robot data for $N = 40$. This is especially remarkable for V-STACS which is a purely hard-updates algorithm. One possible explanation is that V-STACS' coarseness helps it avoid overfitting when splitting states.

5.3 Model Selection Accuracy

5.3.1 BIC score

The final BIC scores and N values of STACS, V-STACS, Li-Biswas and ML-SSS are shown in Table 2 when allowed to stop splitting autonomously. Note that the exact BIC score was not used by V-STACS, just calculated for comparison. In all cases, STACS converges on models with the highest BIC score. For the Mlog data, STACS achieves a better BIC score even with a smaller model than V-STACS, indicating that the soft-updates method found a particularly good local optimum.

The consistent superiority of STACS here may seem to contradict results from the previous section where STACS and V-STACS were seen to be more comparable. A possible reason is that V-STACS uses the Viterbi path likelihood (which is computed during hard-updates training anyway) in place of the true likelihood in BIC. This is done to keep V-STACS as efficient as possible. However the resulting approximate BIC seems to undervalue good splits, resulting in early stoppage as seen here. We can conclude that, though the Viterbi approximation works well for state-splitting, the true likelihood is preferable for model selection purposes when using BIC.

Table 3: Australian sign-language word recognition accuracy on a 95-word classification task, and *average HMM sizes*, on AUSL data.

STACS	V-STACS	Li-Biswas	ML-SSS
90.9%	95.8%	78.6%	89.5%
12.5	*55*	*8.3*	*8.5*

5.3.2 Discovering the Correct Topology

We already saw that STACS is able to learn the correct number of states in the simple example of Figure 1, while Li-Biswas and ML-SSS are not. We generalized this example to a larger, more difficult instance by generating a $10,000$ point synthetic data set similar to the one in Figure 1 but with 10 hidden states with overlapping Gaussian observations.

Even on this data, both STACS and V-STACS consistently found the true underlying 10-state model whereas Li-Biswas and ML-SSS could not do so. Interestingly, regular Baum-Welch on a 10-state HMM also failed to find the best configuration of these 10 states even after 50 restarts. This reinforces a notion suggested by results in Section 5.2: even in fixed-size HMM learning, STACS is more effective in avoiding local minima than multi-restart Baum-Welch.

5.4 Australian Sign-Language Recognition

Though improved test-set likelihood is strong evidence of good models, it is also important to see whether these model improvements translate into superior performance on classification tasks. HMMs play their most important roles in the context of supervised classification and recognition systems, where one HMM is trained for each distinct sequence class. Classification is carried out by scoring a test sequence with each HMM, and the sequence is labeled with the class of the highest-scoring HMM.

One such classification problem is *automatic sign-language recognition* (Starner & Pentland, 1995). We test the effectiveness of our automatically learned HMMs at classification of Australian sign language using the AUSL dataset (Kadous, 2002). The data consists of sensor readings from a pair of Flock instrumented gloves, for 27 instances each of 95 distinct words. Each instance is roughly 55 timesteps. We retained the $(x, y, z, roll, pitch, yaw)$ signals from each hand and trained HMMs on an 8:1 split of the data, using STACS, V-STACS, Li-Biswas and ML-SSS. Table 3 shows classification results along with average HMM sizes. V-STACS yields the highest accuracy along with

Table 2: BIC scores scaled by dataset size, and *(number of states)*, of final models chosen by STACS, V-STACS, Li-Biswas and ML-SSS. STACS and V-STACS consistently find larger models with better BIC scores, indicating more effective split design.

Dataset	STACS	V-STACS	Li-Biswas	ML-SSS
ROBOT	**-1.79*(39)***	-1.81*(34)*	-1.98*(18)*	-2.01*(15)*
MOCAP	**-3.54*(36)***	-3.55*(33)*	-3.69*(20)*	-3.92*(10)*
MLOG	**-8.44*(14)***	-8.45*(20)*	-8.59*(11)*	-10.51*(1)*
AUSL	**-2.77*(44)***	-2.79*(42)*	-2.92*(31)*	-3.04*(28)*
VOWEL	**-4.47*(17)***	-4.49*(16)*	-4.48*(17)*	-4.94*(1)*

much larger HMMs than the other algorithms.

6 Discussion

Part of the contribution of this work is empirical evidence for the conjecture that better modeling of state context compensates more than adequately for considering fewer data points per split in hidden state discovery. In addition, we investigated whether improved dynamic modeling in split design can also compensate for approximating hidden state beliefs by less precise, more efficient hard-updates methods via the V-STACS algorithm. Evaluations show that even V-STACS produces models with higher test-set scores than soft-updates methods like ML-SSS, Li-Biswas and Multi-restart Baum-Welch.

An interesting phenomenon observed is that STACS and V-STACS consistently return *larger* HMMs (with better BIC scores) than ML-SSS and Li-Biswas when stopping autonomously. One possible interpretation is that the split design mechanism continues to find 'good' splits even after the 'easy' splits are exhausted. An illustration of this for the Mocap data is in Figure 3.C. This makes sense considering that model selection and parameter optimization are closely related; since parameter search is prone to local minima, determining the best size of a model depends on being able to find regions of parameter space where good candidate models reside. Similarly, it is surprising that that V-STACS yielded by far the highest classification accuracy in the sign-language recognition task (Section 5.4), that too with much larger final HMMs than any other algorithm. More investigation is needed in this area to see if these two things hold true in other sequence classification domains.

Previous work for finding the dimensionality of *discrete-valued* hidden variables in HMMs (Stolcke & Omohundro, 1994) and other Bayesian Networks (Elidan & Friedman, 2001) has demonstrated that hard-updates model selection algorithms can yield much greater efficiency than soft-updates methods without a large loss of accuracy. To our knowledge, however, this is the first work that demonstrates hard-updates model selection to be competitive for *real-valued* hidden variables (Sections 5.2, 5.4). One possible explanation is that the coarseness of hard-updates splitting helps avoid overfitting early on which might otherwise trap the algorithm in a local optimum.

Results from Sections 5.2 and 5.3.2 indicate that STACS is a competitive *fixed-size HMM learning algorithm* compared to previous approaches in terms of test-set scores *and* efficiency. To our knowledge, this is the first HMM model selection algorithm that can make this claim. Consequently, there is great potential for applying STACS to domains where continuous-density HMMs of fixed size are used, such as speech recognition, handwriting recognition, financial modeling, bioinformatics and even domains where real-valued hidden-variable models are currently the norm, such as mobile robot localization (Sharma et al., 2005).

There are several possibilities for building on this work, both in HMMs as well as in general Bayesian Networks. As massive data streams become increasingly common, one emerging goal is to be able to work with large HMMs with hundreds or thousands of states, as in Felzenszwalb et al. (2003). Several heuristics, such as lazy evaluation of split candidates, could be applied to accelerate STACS for learning such large HMMs more efficiently. Due to greedy splitting on subsets of data, some states may end up being redundant: these could be merged back in the final model. Recent work by Siddiqi and Moore (2005) investigates constraining the transition model to allow fast inference and learning in large non-sparse HMMs. It would be useful to investigate how the behavior of model selection algorithms differs under such model constraints. Finally, STACS can also be generalized to the task of finding the dimensionality of real-valued hidden variables in Bayesian Networks. Such a generalization for discrete-valued hidden variables was carried out in Elidan and Friedman (2001) with positive results.

Acknowledgements

Sajid Siddiqi was supported by a CDC award on “Efficient, scalable multisource surveillance algorithms for Biosense” (8-R01-HK000020-02) and DARPA’s CS2P program (HR0011-06-1-0023).

References

D.J. Newman, S. Hettich, C. B., & Merz, C. (1998). UCI repository of machine learning databases.

Elidan, G., & Friedman, N. (2001). Learning the dimensionality of hidden variables. *Proc. UAI.*

Felzenszwalb, P., Huttenlocher, D., & Kleinberg, J. (2003). Fast Algorithms for Large State Space HMMs with Applications to Web Usage Analysis. *Advances in Neural Information Processing Systems (NIPS).*

Howard, A., & Roy, N. (2003). The robotics data set repository (radish).

Kadous, M. W. (2002). *Temporal classification: Extending the classification paradigm to multivariate time series.* Doctoral dissertation, University of New South Wales.

Krogh, A., Mian, I., & Haussler, D. (1994). A hidden Markov model that finds genes in E. coli DNA. *Nucleic Acids Research*, *22*, 4768–4778.

Li, C., & Biswas, G. (1999). Temporal pattern generation using hidden markov model based unsupervised classification (pp. 245–256.).

Ney, H. (1990). Acoustic modeling of phoneme units for continuous speech recognition. *Proc.* 5^{th} *Europ. Signal Processing Conference*.

Ostendorf, M., & Singer, H. (1997). Hmm topology design using maximum likelihood successive state splitting. *Computer Speech and Language*, *11*, 17–41.

Rabiner, L. R. (1989). A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proc. IEEE*, *77*, 257–285.

Ren, L., Patrick, A., Efros, A. A., Hodgins, J. K., & Rehg, J. M. (2005). A data-driven approach to quantifying natural human motion. *SIGGRAPH 2005*, *24*, 1090–1097.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*.

Seymore, K., McCallum, A., & Rosenfeld, R. (1999). Learning hidden Markov model structure for information extraction. *AAAI’99 Wkshp Machine Learning for Information Extraction*.

Sharma, A., Morales, D., Kantor, G., & Choset, H. (2005). Towards removing artificial landmarks for autonomous exploration in structured environments. Master’s thesis, Carnegie Mellon University.

Siddiqi, S. M., & Moore, A. W. (2005). Fast inference and learning in large-state-space HMMs. *Proc. ICML.*

Starner, T., & Pentland, A. (1995). Visual Recognition of American Sign Language Using Hidden Markov Models. *Proc., Intl. Workshop on Automatic Face and Gesture Recognition (IWAFGR).*

Stolcke, A., & Omohundro, S. (1994). *Best-first Model Merging for Hidden Markov Model Induction* Technical Report TR-94-003). Intl. Computer Science Institute.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, *IT-13*, 260–267.

Appendix

Given below is pseudocode for the *Split-State Viterbi* algorithm used to compute the E-step of V-STACS. Intuitively the algorithm performs dynamic programming on a constrained subset of timesteps and a binary state space consisting of the two states produced by the split.

Split-State Viterbi$(\lambda, Q^*, T^s, O_{T^s})$

Initialization: For $i \in \{1, 2\}$
$\pi_{s_i} \leftarrow \frac{1}{2}\pi_s$
$a_{s's_i} \leftarrow \frac{1}{2}a_{s's}$
$a_{s_is'} \leftarrow a_{ss'}$
$a_{s_is_j} \leftarrow \frac{1}{2}a_{ss}$
$b_{s_i} \leftarrow$ initialize to MLE using O_{T^s} plus noise

Loop: for $l = 1 \ldots |T^s|$
$t \leftarrow T^s[l]$
if $t == 1$
then for $i \in \{1, 2\}$
$\delta^s_t(i) \leftarrow \pi_{s_i} b_{s_i}(O_1)$
$\psi^s_t(i) \leftarrow -1$
else if $(t-1) == T^s[l-1]$
then for $i \in \{1, 2\}$
$\delta^s_t(i) \leftarrow [\max_{j\in\{1,2\}} \delta^s_{t-1}(j) a_{s_js_i}] b_{s_i}(O_t)$
$\psi^s_t(i) \leftarrow \arg\max_{j\in\{1,2\}} \delta^s_{t-1}(j) a_{s_js_i}$
else for $i \in \{1, 2\}$
$\delta^s_t(i) \leftarrow a_{q^*_{t-1}s_i} b_{s_i}(O_t)$

Termination:
For all subsequences of T^s that are contiguous in $\{1 \ldots T\}$, backtrack through ψ^s from the end of the subsequence to its beginning to retrieve the corresponding portion of Q^*_s.

return Q^*_s.