RC23395 (W0410-170) October 26, 2004
Mathematics

IBM Research Report

Dynamic Model Selection in IOHMMs

Vittorio Castelli, Daniel A. Oblinger, Lawrence D. Bergman, Tessa A. Lau
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Dynamic Model Selection in IOHMMs

Vittorio Castelli[*]
IBM T.J.Watson Research Center
Yorktown Heights, NY 10598
vittorio@us.ibm.com

Daniel A. Oblinger
IBM T.J.Watson Research Center
Yorktown Heights, N.Y. 10598
oblio@us.ibm.com

Lawrence D. Bergman
IBM T.J.Watson Research Center
Yorktown Heights, N.Y. 10598
bergmanl@us.ibm.com

Tessa A. Lau
IBM T.J.Watson Research Center
Yorktown Heights, N.Y. 10598
tessalau@us.ibm.com

Abstract

In this paper we describe adaptive model selection methods for an extension of the IOHMM called SimIOHMM. We show how to select the initial number of states of the HMM, how to decide when to add new states during the Baum-Welch iterations, and how to modify the Baum-Welch algorithm to efficiently add new nodes. We show that the SimIOHMM with dynamic model selection yields substantial computational gains over the IOHMM with no or little impact on predicting abilities.

1 Introduction

In this paper we describe efficient adaptive model selection methods for Input-Output Hidden Markov Models. More specifically, we extend the IOHMM [3] to naturally support novel algorithms for the selection the initial number of states of the model, and for dynamically changing the number of states during the Baum-Welch iterations, in a way that can be adapted to incremental learning. The resulting algorithm, the SimIOHMM, the core idea of which is described in more details in [7], yields substantial computational gains over the traditional IOHMM, without compromising the prediction accuracy.

There is an extensive literature on how to select the initial number of states of a HMM. Most approaches adapt general model-selection criteria, such as MDL [2], BIC [9], or criteria designed for specific tasks, such as the Discriminative Information Criterion [4]. In a nutshell, these methods are based on an objective function that simultaneously penalizes large models and favors models that describe the data well. The model selection procedure consists of constructing a collection of models having different number of states (which is an expensive operation), and selecting the one for which the objective function is best. The objective function is often evaluated using cross-validation or holdout estimates. The high computational cost of these methods makes them unsuitable as the foundation for incremental learning.

[*]Webpage: http://www.research.ibm.com/people/v/vittorio

There are several algorithms for dynamically reducing the number of states in an HMM. For example, the approach described in [5] for mixture models can be easily adapted to select the model of an HMM. This method initializes the Baum-Welch algorithm with a number of states that is very large compared with the number of surviving states at convergence, and this initialization strategy makes it both computationally expensive and unsuitable as the foundation for incremental learning. A method for dynamically splitting HMM states is described in [1], consisting of computing, during each iteration, the expected likelihood gain that would result from splitting each state, and splitting the state with highest gain.

The main contribution of this paper is the description of algorithms for initial model selection and for adding new states (in a way that is more general than state-splitting), supported by the SimIOHMM, that can also be used as a basis for incremental learning. The SimIOHMM additionally accommodates the state-pruning methods of [5], and therefore supports a full set of algorithms for dynamically selecting the number of hidden states.

The motivation for our work is learning procedural knowledge by demonstration. The Sheepdog system, described for example in [6], observes users performing procedures on a GUI, translates the observations into sequences of input-output pairs (where the input is a description of what the user sees and knows about the system, and the output is the action taken by the user), and combines multiple sequences into a model of the procedure, that can be used to guide users through the procedure or to automatically re-execute the learned task. The Sheepdog system represents procedures as probability distributions over finite sequences of input-output pairs using the SimIOHMM to build the model. The SimIOHMM, however, is a general-purpose algorithm, applicable to a variety of problem domains.

The rest of the paper is organized as follows: in Section 2 we describe the SimIOHMM. The main contributions of this paper are in Section 3, describing the SimIOHMM initial model selection algorithm, and in Section 4, describing the algorithm for dynamically adding states as part of the Baum-Welch algorithm. Experimental results are presented in Section 5, and the conclusions are in Section 6.

2 The Similarity-based-IOHMM (SimIOHMM)

We begin this section with a brief overview of the IOHMM [3]. We use standard notation (see, for example, [8]) augmented by the IOHMM-specific notation used in [3]. The IOHMM models a sequence of input-output pairs $\{(U_i, Y_i)\}_{i=1}^{T}$ (where the input $U \in \mathbb{U}$ and the output $Y \in \mathbb{Y}$) as a finite-state process, namely, as a hidden Markov model with a finite state space $\mathbb{X}$. The learning algorithm is a version of the classical Baum-Welch algorithm where the complete data likelihood is the conditional likelihood of the output-state pairs given the inputs—hence, the IOHMM models the conditional distribution of the outputs given the inputs. The IOHMM is implemented using a pair of classifiers for each of the states in the model: a transition classifier, that embodies a conditional probability distribution over the states given the input and the state, and an action classifier, that embodies a conditional probability distribution over the outputs given the input and the state.

Our goal is to construct procedures by demonstration. The IOHMM can be used as the learning algorithm because it can predict the next user action given the observed sequence of input-outputs. However, in this paper we impose additional requirements on the learning algorithm, including automatic model selection and the ability to identify highly unlikely inputs given the observed input-output pairs—and for this specific reason, we need to model the joint distribution of inputs and outputs. The result is the SimIOHMM, the core algorithm of which is introduced in the companion paper [7] as a way of biasing the learning algorithm based on the similarity of inputs ([7] does not deal with model selection).

Structurally, a SimIOHMM consists of a IOHMM, a collection of representative inputs (one

for each hidden state[1]), a distance function between inputs,[2] $d(\cdot,\cdot)$ and a finite-support kernel function $K(\cdot)$, (say, taking value in $[0,1]$).[3] Representative inputs, distance, and kernel are incorporated in the Baum-Welch algorithm. Formally, we can substitute the input-output pair (U,Y) for the output Y (called O for observation by some authors) in the classical Baum-Welch algorithm and obtain its SimIOHMM version. The E-Step is then analogous to that of the classical HMM. The M-step updates three sets of quantities: the initial probability distribution on the states, π; the state-transition probability distribution $\mathbf{A}$, which in the IOHMM is a mapping from $\mathbb{X}\times\mathbb{U}$ to $\mathcal{P}(\mathbb{X})$, the set of probability distributions over the state space; and the conditional probability distribution of the observations given the states, $\mathbf{B}=\{b_x(u,y)\}=P_\Theta(U=u,Y=y\mid X=x)$, where Θ denotes the current (parameters of the) HMM. The chain rule for probabilities yields $b_x(u,y)=P_\Theta(Y=y\mid X=x,U=u)\,P_\Theta(U=u\mid X=x)$.

Both IOHMM and SimIOHMM estimate π using the standard approach, $\mathbf{A}$ with the transition classifier learning algorithm, and $P_\Theta(Y=y\mid X=x,U=u)$ with the output classifier learning algorithm. The IOHMM ignores the term $P_\Theta(U=u\mid X=x)$. The SimIOHMM, by contrast, estimates this term as follows: use Bayes' rule to expand $P_\Theta(U=u\mid X=x)$ as $P_\Theta(X=x\mid U=u)\;P_\Theta(U=u)/P_\Theta(X=x)$. The third term, $P_\Theta(X=x)$, is estimated from the $\gamma_x^{(k)}(t)\stackrel{\triangle}{=}P_\Theta\left(X_t=x\mid (U_i,Y_i)_{i=1}^{T_k}=(u_i^{(k)},y_i^{(k)})_{i=1}^{T_k}\right)$, the conditional probabilities of being in state x at time t given the entire of sequence k of length T_k, where $\gamma_x^{(k)}(t)$ is computed in the previous E-step. The term $P_\Theta(U=u)$ can be estimated from the data using the Maximum Likelihood Estimator. The distinguishing characteristic of the SimIOHMM is the estimation of $P_\Theta(X=x\mid U=u)$, performed by computing the distances between u and the representative input of each state, using these distances as inputs to the kernel function, and combining the results with the Nadaraya-Watson kernel density estimator:

$$P_\Theta(X=x\mid U=u)=\frac{K\big(d(u,u_x)\big)}{\sum_{z\in\mathbb{X}}K\big(d(u,u_z)\big)}. \tag{1}$$

Clearly, other methods to convert similarity between inputs into probabilities can be used in conjunction with the the SimIOHMM, but this investigation is left for future work.

The M-step for the SimIOHMM must also be modified to include the recomputation of the representative inputs. For each state x, the input-output pair $(u_x^\diamond,y_x^\diamond)$ that has the highest probability of being aligned with the state is identified, and $u_x^\diamond$ is selected as the new representative input of the state x. To support the dynamic addition of new nodes (Section 4), this step is performed before updating the parameters of transition and output models.

In practice, $d(\cdot,\cdot)$ is selected to capture gross differences between inputs. In our application, where inputs are descriptions of application windows, $d(\cdot,\cdot)$ considers features such as the title and type of the foreground window, and the name of the application owning it.

The first benefit of the SimIOHMM is computational efficiency: the entropy of $\gamma_x(t)$ quickly decreases, typically to a small fraction of the entropy produced by the IOHMM. The cost of training the transitions and action classifiers depends on the number of training samples, which is an increasing function of the entropy of $\gamma_x(t)$. The second benefit, which is more application-dependent, is the simplification of the variable subset selection task: in our application for a IOHMM, is extremely large, but typically sparse. The SimIOHMM

[1]The SimIOHMM can rely on multiple representative inputs per each state. To simplify the discussion, we assume a single representative input per state.

[2]A version of the SimIOHMM uses a distance function between input-output pairs, in which case each state has a representative input-output pair, and the structure of the model is appropriately modified. Its description is beyond the scope of this paper.

[3]Kernel functions with infinite support can also be used, but require small changes to the algorithms, that we do not discuss here for sake of clarity.

associates similar inputs with the same states, and this allows us to efficiently perform aggressive adaptive dimensionality reduction with no impact on accuracy.

3 Initial model selection

The structure of the SimIOHMM supports a straightforward way of selecting a minimal initial model. Start with a state space $\mathbb{X}$ containing a single state, interpreted as the source state of the HMM. The initial state has no associated representative input. Incrementally grow the state space by iterating over the training sequences, as follows. For each input-output pair (u,y), for each x in $\mathbb{X}$, compute $K(d(u,u_x))$. If all these values are equal to zero, add a new state to $\mathbb{X}$ and let u be its representative input; otherwise analyze the next input-output pair. Terminate when all the training sequences have been analyzed.
A second method, for selecting a somewhat larger initial model, consists of adding a new state x_{new} if $d(u,u_x) > 0$ for every $x \in \mathbb{X}$, and setting u as the representative input of x_{new}.

Once the initial number of states has been selected, and the corresponding representative inputs assigned, the E-M algorithm for the SimIOHMM is seeded by constructing initial values of $\gamma_x(u_t)$ and of the transition probabilities $a_{i,j}(u_t) = P(X_{t+1} = j \mid X_t = i, U_t = u_t)$ for each step of each training sequence, as follows. Let $\{(u_t^{(k)}, y_t^{(k)})\}_{t=1}^{T_k}$ be the kth training sequence. For each pair (u_t, y_t), compute $P_\Theta(X = x \mid U_t = u_t)$ using the estimator of Equation 1 in conjunction with the initial representative inputs. Set the initial value of $\gamma_x^{(k)}(u_t^{(k)})$ to $P_\Theta\left(X_t = x \mid U_t = u_t^{(k)}\right)$, and the initial value of $a_{i,j}^{(k)}(u_{t+1}^{(k)})$ to $P_\Theta\left(X_{t+1} = j \mid U_{t+1} = u_{t+1}^{(k)}\right)$. Note that these choices of $\gamma_x^{(k)}(u_t^{(k)})$ and $a_{i,j}^{(k)}(u_t^{(k)})$ are consistent, namely,

$$\begin{aligned}\sum_{x\in\mathbb{X}} \gamma_x^{(k)}(u_t^{(k)}) a_{x,j}^{(k)}(u_{t+1}^{(k)}) &= \sum_{x\in\mathbb{X}} \gamma_x(u_t^{(k)}) P_\Theta\left(X_{t+1} = j \mid U_{t+1} = u_{t+1}^{(k)}\right) \\ &= P_\Theta\left(X_{t+1} = j \mid U_{t+1} = u_{t+1}^{(k)}\right) = \gamma_j^{(k)}(u_{t+1}^{(k)}).\end{aligned}$$

This initialization algorithm can also be used to selectively add new states to a SimIOHMM that is incrementally constructed by providing additional training sequences.

In our experience, the SimIOHMM with this initialization regularly produces a model that has the same prediction accuracy as as a IOHMM constructed, say, with the Bayesian Information Criterion. This is the case for the experiments reported in the paper. However, on occasion the initialization produces a suboptimal model that can be improved by allowing the SimIOHMM to dynamically add and remove states.

4 Dynamically Adding States to the SimIOHMM

As mentioned in the introduction, the SimIOHMM gracefully accommodates the method described in [5] for pruning states from the state space, a slight variant of which is used in our implementation. The SimIOHMM also supports a method for dynamically adding states to the state space during the Baum-Welch iterations. Unlike other methods that perform state splitting, the SimIOHMM adds states in a way that cannot be reduced to splitting an existing state.

There are two components to the algorithm: the decision to add a new node, and the efficient recomputation of the E-step after the node is added. The SimIOHMM recomputes the representative inputs at the beginning of the M-step. The SimIOHMM then checks, for each input u in the training set, the collection of values $\{K\big(d(u,u_x)\big), x \in \mathbb{X}\}$. If all these values are equal to zero for an input u^*, the model is too small to explain the input sequence to which u^* belongs. This can happen if either the initial model is too small, or if

the set of recomputed representative inputs does not contain any input sufficiently similar to u_x (this could occur, for example, when the training sequence is an outlier, namely, has low probability under the true distribution). Hence, the SimIOHMM learning algorithm decides to add a new node only when the current model is insufficient to explain the data. The representative input of the new state is set to u^*.

Once a new state is added, the learning algorithm efficiently recomputes the alignment (the $\gamma_x^{(k)}(t)$ and $\xi_{i,j}^{(k)}(t) = P_\Theta\left(X_t = i, X_{t+1} = j \mid \{(u_l^{(k)}, y_l^{(k)})\}_{l=1}^{T_k}\right)$ quantities produced by the E-step). The brute-force approach consists of recomputing the entire alignment after the new state is added. However, typically only a small fraction of the training samples are affected by the addition of the new state, and the following efficient solution is adopted instead. It is always necessary to recompute $\gamma_x^{(k^*)}(t^*)$, $\xi_{i,j}^{(k^*)}(t^*-1)$ and $\xi_{i,j}^{(k^*)}(t^*)$, where k^* is the index of the training sequence to which u^* belongs, and t^* is the index of u^* within its sequence. The quantity $\gamma_x^{(k^*)}(t^*)$ is recomputed by setting it to 0 for all the states except the new one, say n, for which it is set to 1. The quantity $\xi_{i,j}^{(k^*)}(t^*-1)$ is set to 0 for all $j \neq n$, and to $\gamma_i^{(k^*)}(t^*-1)$ for all i and $j = n$, while $\xi_{i,j}^{(k^*)}(t^*)$ is set to 0 for all $i \neq n$ and to $\gamma_j^{(k^*)}(t^*+1)$ for $i = n$ and all j. It might also be necessary to recompute γ and ξ for other input-output pairs. When a new state is added, all the input-output pairs $(u_t^{(k)}, y_t^{(k)})$ in the training set are scanned (k is the index of the training sequence to which $(u_t^{(k)}, y_t^{(k)})$ belongs), and those for which $K(d(u_t, u^*)) = 0$ are discarded,[4] because the value of $\gamma_x^{(k)}(t)$ is unaffected by the addition of the new state. However, if an input $u_t^{(k)}$ is sufficiently similar to u^* that $K(d(u_t^{(k)}, u^*)) > 0$, the kth sequence is further analyzed by comparing the distributions $\gamma_X^{(k)}(t-1)$ and $\gamma_X^{(k^*)}(t^*-1)$ over the state space for the input-output pairs preceding $(u_t^{(k)}, y_t^{(k)})$ and (u^*, y^*) respectively. If these distributions have disjoint support sets, then the conclusion is that different parts of the state space are used to explain the $(u_t^{(k)}, y_t^{(k)})$ and (u^*, y^*), hence the alignment of $(u_t^{(k)}, y_t^{(k)})$ is not influenced by the newly created state, and therefore u_t is ignored. If this is not the case γ and ξ for u_t are updated as described later. Empirical evidence has shown that this approach can produce slightly larger models than a method that always updates γ and ξ for every $u_t^{(k)}$ such that $K(d(u_t^{(k)}, u^*)) > 0$, at the expense of a statistically significant reduction in the cost of induction, and with no impact on accuracy. Recomputing γ and ξ for an input-output pair $(u_t^{(k)}, y_t^{(k)})$ described above is more complex. Let $\mathcal{A}$ be the intersection of the support sets of $\gamma_X^{(k)}(t-1)$ and $\gamma_X^{(k^*)}(t^*-1)$. Let $\tilde{\gamma}_j^{(k)}(t) = \delta\left(K(d(u_t^{(k)}, u^*)) > 0\right) \sum_{i \in \mathcal{A}} \xi_{i,j}^{(k)}(t)$ be the "contribution" to $\gamma_j^{(k)}(t)$ of the nodes in $\mathcal{A}$, where δ is the Dirac $0-1$ indicator function. We update $\gamma_j^{(k)}(t)$ using the equations

$$\gamma_j^{(k)}(t) = \tilde{\gamma}_j^{(k)}(t)\left(1 - \frac{K(d(u_t^{(k)}, u^*))}{\sum_{x \in \mathbb{X}} K(d(u_t^{(k)}, u_x))\delta(\tilde{\gamma}_j^{(k)}(t) > 0)}\right) \quad \text{for } j = 1 \ldots, n-1$$

$$\gamma_n^{(k)}(t) = 1 - \sum_{j=1}^{n-1} \gamma_j^{(k)}(t),$$

that is, we distribute part of the probability of the $\mathcal{A}$ to the new node, in proportion to the probabilities estimated by the Kernel density estimator. Finally, for every $j \in \mathcal{A} \bigcup \{n\}$ the quantities $\xi_{i,j}^{(k)}(t-1)$ are recomputed to be consistent with the new values $\{\gamma_j^{(k)}(t)\}$.

This approach selectively adds new states while trying to minimize the computational cost,

[4] The SimIOHMM can be implemented so that $K(d(u_t, u^*))$ is not recomputed during the scan.

and for this reason we believe it can be successfully used for incremental learning.

5 Experimental Results

The IOHMM algorithm was implemented in Java, with instances of C4.5 as the transition and output classifiers, and the SimIOHMM was implemented by extending the IOHMM code. All experiments were run on an IBM computer with 2 Intel Xeon Processors at 2.4GHz, 512K of L2 cache per processor, and 8 Gb of RAM, running Linux. The dataset contains 11 training sequences, recorded while an expert performed a procedure for configuring the DNS settings of a system. Each sequence corresponded to a different initial configuration of the computer system and contains between 17 and 38 steps. The paths taken by the experts had substantial overlap, but only four sequences can be inferred from the remaining ones.

Figure 1 shows the training time of the standard IOHMM, and of four versions of the SimIOHMM characterized by the initialization (random initialization or adaptive) and by the variable subset selection (enabled or disabled). The number of nodes for the IOHMM and the SimIOHMM with random initialization was set to 24, the number selected by the SimIOHMM with adaptive initialization. The figure clearly shows a two-order of magni-

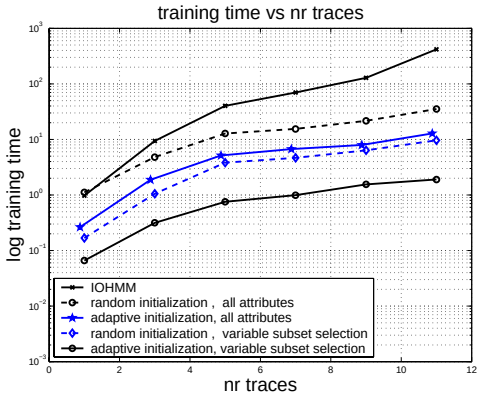

Figure 1: Training time vs. number of training sequences for IOHMM, SimIOHMM with random or adaptive initialization, and with or without variable subset selection.

tude reduction in training time achieved by the SimIOHMM wth adaptive initialization and variable subset selection over the standard IOHMM. Additionally, the slope of the curve for the IOHMM is larger than that for the SimIOMM, which indicates superlinear gains in training time as a function of the number of training sequences.

The next results are the median values of 7, 4-way cross-validation experiments. They compare the performance of the IOHMM, of the SimIOHMM with adaptive initialization, and of the SimIOHMM with dynamic state addition and random initialization (for which the results are reported as a function of the initial number of states). Figure 2 shows that the number of iterations to convergence for the SimIOHMM with adaptive initialization is slightly smaller than that of the SimIOHMM with dynamic state addition. The training time is substantially higher (but still at least one order of magnitude smaller than that of the IOHMM), and we attribute this to the fact that when a new state is added, the entropy of γ temporarily increases, and the cost of each iteration is an increasing function of the entropy. When the starting number of states grows, both node addition and node deletion mechanisms come into play, and the cost of the induction appears to grow accordingly.

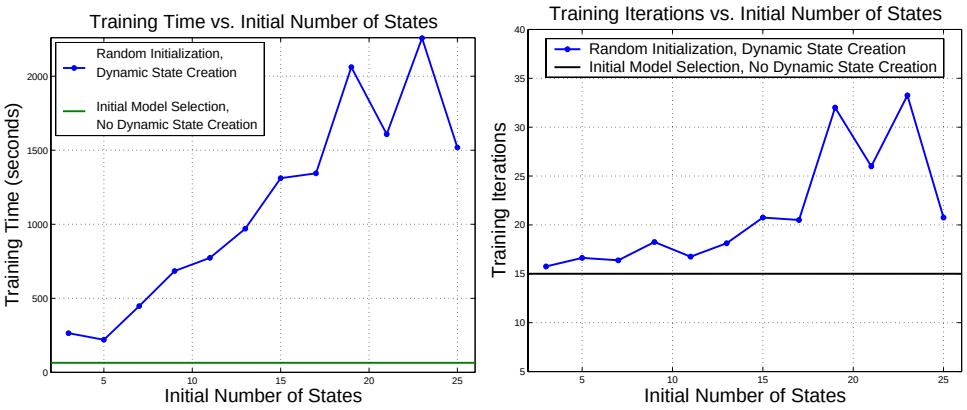

Figure 2: Training Times and Training Iterations. The IOHMM required 100 iterations. The SimIOHMM with Initial Model Selection has 14 states.

Figure 3 shows the overall prediction accuracy, and the prediction accuracy of pairs that

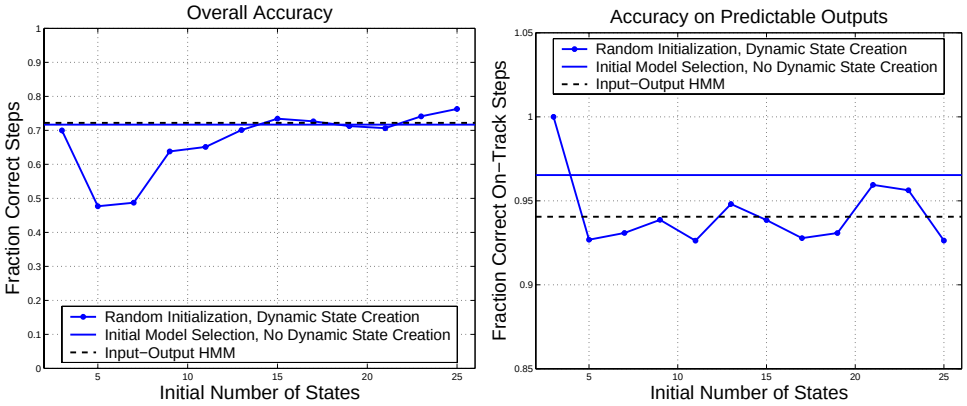

Figure 3: Overall accuracy, and accuracy on predictable input-output pairs. The SimIOHMM with Initial Model Selection has 14 states, and the IOHMM has 14 states.

the HMMs declares as being *unpredictable*, namely, not learnable from the training set. The IOHMM declares a pair to be unpredictable only if the output was not observed in the training set. The SimIOHMM also decides that a pair is unpredictable if the conditional probability of the input given the observed input-output pairs is equal to 0—the IOHMM has no provision for such detection. The SimIOHMM with adaptive initialization identifies more pairs as being unpredictable than the IOHMM, and its accuracy on predictable outputs is 3% higher. The overall accuracy of the two algorithms is identical.

The t-test (at the 0.05% significance level) shows that, on predictable pairs, the SimIOHMM with dynamic state addition and random initialization is at least as accurate than the IOHMM for any initial number of states. The t-test without the Bonferroni correction identifies the accuracy for 3, 5, 7, 9, 11, 13, and 21 initial states to be lower than that of the IOHMM. With the Bonferroni correction, the t-test shows that the SimIOHMM is at least as accurate as the IOHMM when the initial number of states is > 9.

6 Discussion and Conclusions

We have described dynamic adaptive model selection strategies for the SimIOHMM, a variation of the IOHMM that models the joint distribution of input-output sequences. We have shown that the SimIOHMM provides a substantial improvement in training time over the IOHMM, due to the the fact that the entropy of γ is quickly reduced during the first iterations of the Baum-Welch, which in turn substantially reduces the cost of training the output and especially the transition classifiers.

We have discussed two methods for selecting the initial number of states, a method for deciding when to add a new state, and a corresponding method for adding a new state. When using the adaptive initialization strategy, the current method for deciding to dynamically add a state usually does not trigger a node addition. Therefore, we have explored separately the contributions of the adaptive initialization strategy and of the dynamic state addition strategy. We have shown that the adaptive initialization strategy alone can substantially reduce the cost of induction without negatively affecting the accuracy, while the algorithm for dynamically adding states has a behavior that depends on its initialization. It appears that the best tradeoff between training time and prediction accuracy for this last algorithm consists of starting with a small number of initial states.

Future work includes the development of additional algorithms for deciding when to add new states, tailored towards maximizing predictive ability; the investigation of the scalability of the SimIOHMM to large datasets; and the construction and evaluation of incremental learning strategies that rely on the adaptive model selection algorithms presented herein.

Acknowledgments

We would like to thank John Turek and Alex Morrow for insightful discussions.

References

[1] Stolcke Andreas and Stephen Omohundro. Hidden markov model induction by bayesian model merging. In J. D. Cowan S. J. Hanson and C. L. Giles, editors, *In Advances in Neural Information Processing Systems 5*, pages 11–18. Morgan Kaufman, 1992.

[2] Andrew Barron, Jorma Rissanen, and Bin Yu. The minimum description length principle in coding and modeling. *IEEE Trans. Information Theory*, 44(6):2743–2760, 1998.

[3] Yoshua Bengio and Paolo Frasconi. Input-Output HMM's for sequence processing. *IEEE Trans. Neural Networks*, 7(5):1231–1249, September 1996.

[4] Alain Biem. A model selection criterion for classification: Application to HMM topology optimization. In *Proc. 7th Int. Conf. Document Analysis and Recognition, ICDAR2003*, pages 104–108, Edinburgh, Scotland, 2003.

[5] Mario A. T. Figueiredo and Anil K. Jain. Unsupervised learning of finite mixture models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(3):381–396, 2002.

[6] Tessa A. Lau, Lawrence D. Bergman, Vittorio Castelli, and Daniel Oblinger. Sheepdog: Learning procedures for technical support. In *Proc. of 2004 Int. Conf. on Intelligent User Interfaces (IUI 2004)*, pages 106–116, 2004.

[7] Daniel A. Oblinger, Vittorio Castelli, Tessa A. Lau, and Lawrence D. Bergman. Similarity-based alignment and generalization: A new paradigm for programming by demonstration. Submitted to NIPS 2004.

[8] Lawrence R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proc. IEEE*, 77(2):257–286, 1989.

[9] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464, 1978.