Monte Carlo Hidden Markov Models: Learning Non-Parametric Models of Partially Observable Stochastic Processes

Sebastian Thrun **John C. Langford** **Dieter Fox**

Robot Learning Laboratory, School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
{thrun,jcl,dfox}@cs.cmu.edu

Abstract

We present a learning algorithm for non-parametric hidden Markov models with continuous state and observation spaces. All necessary probability densities are approximated using samples, along with density trees generated from such samples. A Monte Carlo version of Baum-Welch (EM) is employed to learn models from data. Regularization during learning is achieved using an exponential shrinking technique. The shrinkage factor, which determines the effective capacity of the learning algorithm, is annealed down over multiple iterations of Baum-Welch, and early stopping is applied to select the right model. Once trained, Monte Carlo HMMs can be run in an any-time fashion. We prove that under mild assumptions, Monte Carlo Hidden Markov Models converge to a local maximum in likelihood space, just like conventional HMMs. In addition, we provide empirical results obtained in a gesture recognition domain.

1 Introduction

Hidden Markov models (HMMs) [27] have been applied successfully to a large range of temporal signal processing problems, for tasks such as modeling, filtering, classification and prediction of partially observable time sequences, stochastic environments. Hidden Markov models are often the method of choice in areas such as speech recognition [27, 33], natural language processing [3], robotics [21, 29], biological sequence analysis [11, 16], and time series analysis [10].

We present a new hidden Markov model, called *Monte-Carlo Hidden Markov Models (MCHMMs)*. MCHMMs use samples and non-parametric density trees to represent probability distributions. This makes it possible to learn non-parametric hidden Markov models with continuous state and observation spaces. Since continuous state spaces are sufficiently rich to overfit any data set, our approach uses shrinkage as a mechanism for regularization. The shrinkage factor, which determines the effective capacity of the HMM, is annealed down over multiple iterations of EM, and early stopping is applied for model selection.

MCHMMs possess the following four characteristics, which distinguishes them from the traditional HMM approach [27]:

1. **Real-valued spaces.** Both the state space and the observation space of MCHMMs are continuous. This is important in domains where the true state and observation space of the environment is continuous. The importance of continuous-valued spaces has been recognized by several authors, which have proposed real-valued extensions using parametric models such as Gaussians and neural networks [2, 8, 9, 13, 17].
2. **Non-parametric.** Most existing HMM models rely on parametric densities, defined by a small set of parameters (discrete distributions included). This is clearly appropriate when one has reason to believe that the parametric model is correct. In practice, however, one might lack such strong prior knowledge. MCHMMs learn non-parametric models of partially observable stochastic systems. Such non-parametric methods can fit much larger sets of functions, hence provide additional flexibility.
3. **Built-in complexity control.** MCHMMs make it possible to control the complexity of the state space during learning. In contrast, the complexity of discrete HMMs is determined by the number of states, which has to be determined in advance, before training. Multiple training runs may thus be necessary to determine the right number of states.
4. **Resource adaptive.** Finally, after training MCHMMs can be used in an *any-time* fashion. Any-time algorithms can generate an estimate at *any* time; the more time available, however, the better the answer [4, 34]. The any-time property is directly inherited from sampling-based methods, as the number of samples can be controlled on-line.

This paper presents MCHMMs. It describes a sample-based learning algorithm that extends Baum-Welch to MCHMMs.

We also give theoretical results that prove (with probability one) convergence of the learning algorithm to a local maximum in likelihood space. This result, whose proof is only outlined here (see [32] for all proofs), extends the well-known convergence result for discrete HMMs to the real-valued, non-parametric case. We also present empirical results obtained in a gesture recognition domain, which seek to provide the first evidence of the usefulness of MCHMMs in practice.

2 Generalized Hidden Markov Models

This section introduces the necessary notation to extend HMMs to continuous state and observation spaces. It follows the excellent deposition of [27], with continuous density replacing discrete distributions throughout. For convergence, we assume all event spaces and random variables (e.g., state, observations) are measurable. Unless otherwise specified, all probability density functions are Lipschitz (implying continuity) and non-zero over some compact interval.

A *generalized HMM* (in short: GHMM) is a partially observable, time-invariant Markov chain with continuous state and observation spaces and discrete time. At each time $t \geq 1$, the HMM's state is denoted x_t. Initially, at time $t = 1$, the state of the HMM is selected randomly according to the density π. State transitions are governed by a conditional probability density, denoted $\mu(x' \mid x)$ and called *state transition density*.

In HMMs (and thus in GHMMs), the state cannot be observed directly. Instead, only a probabilistic projection of the state is observable. Let b_t denote a measurable random variable that models the observation at time t. Observations are generated according to a probability density conditioned on the state of the HMM (called the *observation density*), denoted $\nu(b \mid x)$. A generalized HMM is uniquely defined trough three densities: $\lambda = \{\pi, \mu, \nu\}$.

Putting computational considerations aside for the moment, knowledge of λ is sufficient to tackle a variety of interesting practical problems, such as computing distributions over states and observations at arbitrary points in time, generating representative example trajectories in state and observation space, determining the likelihood of example trajectories under an HMM, and classifying data sequences generated by mixtures of labeled HMMs. Algorithms for these problems are described in detail in [27]; they are easily transferred from the finite to the continuous case.

In practice, the densities λ are often unknown and have to be estimated from data. The data, denoted d, is a sequence of observations[1], denoted

$$d \quad = \quad \{O_1, O_2, \ldots, O_T\}. \tag{1}$$

[1] For simplicity of the presentation, we only present the case in which the data consist of a single sequence. The extension to multiple sequences is straightforward but requires additional notation.

Here O_t denotes the observation at time t. The total number of observations in d is T.

The well-known Baum-Welch algorithm [1, 20, 27] provides a computationally efficient and elegant approach for learning π, μ, and ν. Baum-Welch begins with an initial model, denoted $\lambda^{(0)}$. It iterates two steps, an E-step and an M-step (see also [6]). In the n-th E-step, distributions for the various state variables x_t are computed under a fixed model $\lambda^{(n)}$ (with $n \geq 0$). The n-th M-step uses these distributions to derive a new, improved model $\lambda^{(n+1)}$.

In the E-step, distributions are computed for the state variables x_t conditioned on a fixed model λ and the data d. Recall from [27] that in the discrete case,

$$\alpha_t^{(n)}(x) \quad = \quad Pr(x_t = x \mid O_1, \ldots, O_t, \lambda^{(n)}) \tag{2}$$

$$\beta_t^{(n)}(x) \quad = \quad Pr(O_{t+1}, \ldots, O_T \mid x_t = x, \lambda^{(n)}) \tag{3}$$

$$\gamma_t^{(n)}(x) \quad = \quad Pr(x_t = x \mid d, \lambda^{(n)}) \tag{4}$$

$$\xi_t^{(n)}(x, x') \quad = \quad Pr(x_t = x, x_{t+1} = x' \mid d, \lambda^{(n)}) \tag{5}$$

The continuous case is analogous; however, here α and β are densities (and thus may be larger than 1). Following [27], these densities are computed incrementally; α is computed forward in time, and β backwards in time (for which reason this algorithm is often referred to as the *forward-backward algorithm*). Initially,

$$\alpha_0^{(n)}(x) \quad = \quad \pi^{(n)}(x) \qquad \text{and} \qquad \beta_T^{(n)}(x) \quad = \quad 1 \tag{6}$$

and for all other α_t and β_t:

$$\alpha_t^{(n)}(x) = \int \alpha_{t-1}^{(n)}(x')\, \mu^{(n)}(x \mid x')\, \nu^{(n)}(O_t \mid x)\, dx' \tag{7}$$

$$\beta_t^{(n)}(x) = \int \beta_{t+1}^{(n)}(x')\, \mu^{(n)}(x' \mid x)\, \nu^{(n)}(O_{t+1} \mid x')\, dx' \tag{8}$$

Bayes rule governs the conditional density over the state space at time t:

$$\gamma_t^{(n)}(x) \quad = \quad \frac{\alpha_t^{(n)}(x)\, \beta_t^{(n)}(x)}{\displaystyle\int \alpha_t^{(n)}(x')\, \beta_t^{(n)}(x')\, dx'} \tag{9}$$

Similarly, the state transition densities $\xi^{(n)}$ are computed as

$$\xi_t^{(n)}(x, x') \quad = \quad \frac{\alpha_t^{(n)}(x)\, \mu^{(n)}(x' \mid x)\, \nu^{(n)}(O_{t+1} \mid x')\, \beta_{t+1}^{(n)}(x')}{\displaystyle\int\!\!\int \alpha_t^{(n)}(\bar{x})\, \mu^{(n)}(\bar{x}' \mid \bar{x})\, \nu^{(n)}(O_{t+1} \mid \bar{x}')\, \beta_{t+1}^{(n)}(\bar{x}')\, d\bar{x}\, d\bar{x}'} \tag{10}$$

This computation is completely analogous to the finite case, replacing conditional probabilities by conditional densities.

The M-step uses $\gamma_t^{(n)}(x)$ and $\xi_t^{(n)}(x, x')$ to compute a new model $\lambda^{(n+1)}$, using the maximum likelihood estimator:

$$\pi^{(n+1)}(x) \quad = \quad \gamma_0^{(n)}(x) \tag{11}$$

$$\mu^{(n+1)}(x' \mid x) = \frac{\sum_{t=1}^{T-1} \xi_t^{(n)}(x, x')}{\sum_{t=1}^{T-1} \gamma_t^{(n)}(x)} \quad (12)$$

$$\nu^{(n+1)}(b \mid x) = \frac{\sum_{t=1}^{T} I_{O_t=b}\ \gamma_t^{(n)}(x)}{\sum_{t=1}^{T} \gamma_t^{(n)}(x)} \quad (13)$$

Here I_{cond} denotes an indicator variable that is 1 if the condition *cond* is true, and 0 otherwise. A straightforward result is the convergence of GHMMs.

Theorem 1. (GHMM Convergence Theorem) *Under the assumptions above, the GHMM learning algorithm does not decrease the likelihood of the observation sequence with probability 1. With probability 1, it does not improve the likelihood if, and only if a local maximum or a saddle point has been reached.*

The full proof can be found in [32]. Our proof is a direct extension of a result by Juang [13], who has shown convergence to local maxima for HMMs with a finite number of states and continuous observations, where the observation densities are a mixture of log concave or ellipsoidal symmetrical densities. The convergence of GHMMs follows from the fact that any continuous and Lipschitz density can be approximated by mixtures of Gaussians, hence by a sequence of HMMs that meet Juang's assumptions.

3 Sampling and Density Trees

3.1 Sampling

MCHMMs use samples to approximate all densities (models, posteriors, conditional and joint densities). Sample sets are (finite) sets of values, annotated by numerical probabilities [18, 23]. More specifically, let f be a probability density function, and let N denote a positive number (the cardinality of a sample set). A *sample set* is a set

$$X = \{\langle x_1, p_{x_1}\rangle \ldots, \langle x_N, p_{x_N}\rangle\} \quad (14)$$

where for all n with $1 \leq n \leq N$: $x_n \in \text{dom}(f)$, $p_{x_n} \in [0, 1]$, and $\sum_{n=1}^{N} p_{x_n} = 1$. Sample sets can be thought of as discrete probability distributions over the event space $\{x_1, \ldots, x_N\}$, defined through the p-values [23, 26] For reasons that will become clear below, p-values are often referred to as *importance factors* [28].

A *sampling method* is a method that approximates f through samples. We will call a sampling method *asymptotically consistent* if for $N \to \infty$, X converges to f with probability 1 when integrated over the system of half-open Borel sets:

$$\lim_{N\to\infty} \sum_{\langle x,p_x\rangle \in X} I_{x_0 \leq x < x_1}\ p_x = \int_{x_0}^{x_1} f(x)\ dx \quad (15)$$

Sampling can be applied to approximate any probability density f [31].

We will distinguish two methods for sampling, both of which are asymptotically consistent:

1. **Likelihood-weighted sampling.** When f is available explicitly, we will directly sample from f (e.g., using rejection sampling [31]). In likelihood-weighted sampling, all p-values are $1/N$. The concentration of the resulting sample set will be proportional to the density f, i.e., the larger the measure of f over a region A, the more samples will be in A (in expectation).
2. **Importance sampling.** In some cases, however, likelihood-weighted sampling cannot be applied. This is the case, for example, when approximating $\alpha_t^{(n)}(x)$, which is a recursive product of convolved densities and other densities conditioned on observations (c.f., (7)). Instead of sampling directly from f, importance sampling [28] samples from a different distribution g that has the property $f(x) > 0 \Longrightarrow g(x) > 0$. The difference between f and g is accounted by the p-values, which are defined as follows:
$$\bar{p}_{x_n} = f(x_n)/g(x_n) \quad (16)$$
$$p_{x_n} = \frac{\bar{p}_{x_n}}{\sum_{n=1}^{N} \bar{p}_{x_n}} \quad (17)$$
Importance sampling [28] has been studied extensively in the statistics literature (see for example [31]). The reader may notice that likelihood-weighted sampling is a special case of importance sampling (with $f \equiv g$).

Figure 1a shows a sample set drawn by likelihood-weighted sampling from a distribution that resembles the shape of a sine wave. All probabilities p_x of the sample set shown there are the same, and the samples are concentrated in a small region of the $\Re^2$.

Both sampling methods can equally be applied to *sample from a sample set* (often referred to as *resampling* [28]). Let X be a sample set. Likelihood-weighted sampling draws samples from X according to the (discrete) probability distribution induced by their p_x-values. Each sample is then assigned the same probability ($1/N$). Importance sampling draws samples x from X according to some other density g, and assigns $\frac{p_x}{g(x)}$ to the new sample's p-value. Sampling from sample sets plays an important role in the Monte Carlo HMM algorithm described below.

A straightforward but important result is the following:

Theorem 2. *Likelihood-weighted sampling converges (in the sense of (15) with probability one to f as $N \longrightarrow \infty$, at a rate of $1/\sqrt{N}$. Importance sampling also converges with probability one to f at the same asymptotic rate if $\frac{f(x)}{g(x)} < \infty$.*

The proof, which is given in detail for likelihood-weighted sampling in [32], follows directly from a result in [31], pg. 33. For the importance sampler, the variance of the error depends on the "mismatch" between f and g.

3.2 Particle Filters

Armed with the definition of sample sets and procedures for generating them, we can now devise a sample-based algo-

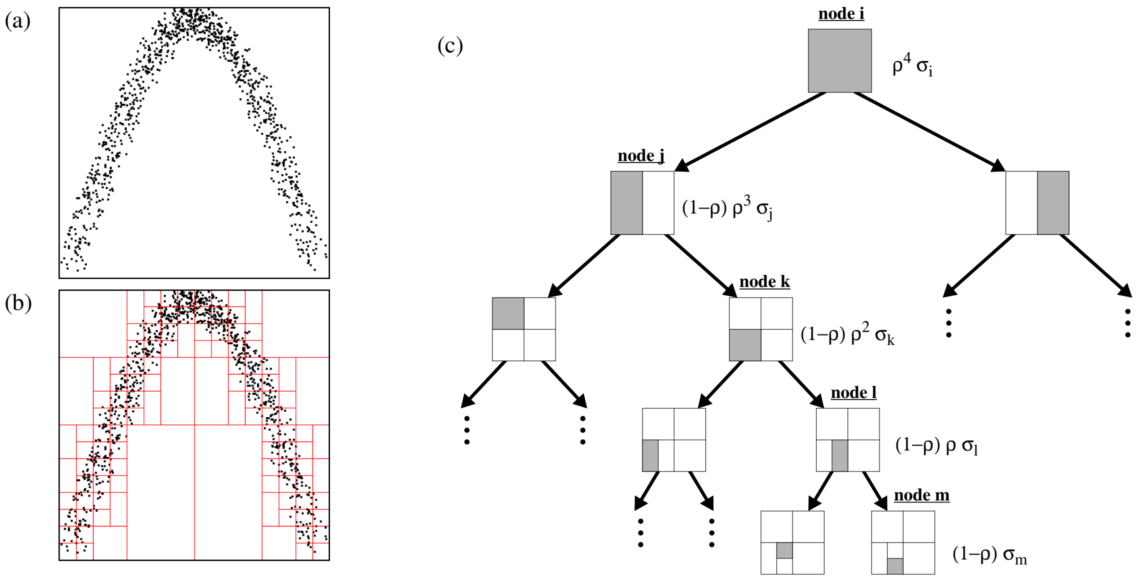

Figure 1: (a) Data set (b) partitioned by a density tree. (c) Density tree with shrinkage for complexity control. Each branch cuts the area under a node into two equally-sized, rectangular halves, shown in grey. The density of each node is an exponentially weighted mixture of densities of the nodes along the path to the leaf. The ρ-terms describe the mixture coefficients for this example.

rithm for tracking the state of a dynamical system: *particle filters* [26]. Variants of this algorithm, which have become popular in recent years, can be found in the literature under names such as *survival of the fittest* [14], *condensation algorithm* [12], or, in the context of mobile robotics, *Monte Carlo localization* [5, 7].

Particle filters assume that the model λ is known. They approximate the distributions of α_t using sampling/importance resampling:

1. **Initialization:** Generate N samples of π (e.g., using likelihood-weighted sampling).
2. **For each** $t = 1, \ldots$ **do:**
   (a) Generate N samples $\langle x', p_{x'} \rangle$ from the sample set representing α_{t-1} using likelihood-weighted sampling.
   (b) For each sample $\langle x', p_{x'} \rangle$, generate a random x from $\mu(x \mid x')$ using likelihood-weighted sampling.
   (c) Set p_x to a value proportional to $\nu(O_t \mid x)$, where O_t is the t-th observation in the data set.

Particle filters have successfully been applied to state estimation problems in computer vision [12] and robotics [5, 7]. In MCHMMs, this algorithm is used to approximate the distributions of α_t and β_t.

3.3 Density Trees

While sample sets are sufficient to approximate continuous-valued distributions, they differ from those in that they are *discrete*. This is problematic if one wants to combine densities represented through different sample sets. For example, according to Equation (9), γ_t is the normalized product of two densities, α_t and β_t, each of which represented by its own sample set. Unfortunately, with probability one, none of the samples in α_t and β_t are identical, and thus it is *not* straightforward how to obtain an approximation of their product.

MCHMMs solve this problem by transforming sample sets into (non-parametric) density trees [15, 22, 25], which effectively "generalize" discrete distributions (samples) to continuous distributions. Each node in a density tree is annotated with a hyper-rectangular subspace of dom(f), denoted V (or V_i for the i-th node). Initially, all samples are assigned to the root node, which covers the entire domain of f. The tree is grown by splitting each node i whenever the following two conditions are fulfilled:

1. At least $\sqrt{N}$ samples $\langle x, p_x \rangle \in X$ fall into V_i.
2. Its depth, i.e., its distance from the root node, does not exceed $\lfloor \frac{1}{4} \log_2 N \rfloor$.

If a node is split, its hyper-rectangle v is divided into two equally sized hyper-rectangles along its longest dimension, as illustrated in Figure 1c. Figure 1b shows a density tree generated for the sample set shown in Figure 1a.

Let f be a density function, X a sample drawn from this density, and for all $x \in$ dom(f) let $i(x)$ denote the leaf whose region V_i contains x. Furthermore, let $\hat{\sigma}_i$ denote the relative frequency of samples in the i-th node weighted by their respective p_x values:

$$\hat{\sigma}_i \quad = \sum_{\langle x, p_x \rangle \in X} I_{x \in V_i}\, p_x \tag{18}$$

Hence the density function of a tree, denoted $\hat{f}$, is defined as follows:

$$\hat{f}(x) \quad = \quad \frac{\hat{\sigma}_{i(x)}}{|V_{i(x)}|} \qquad (\forall x \in \text{dom}(f)) \tag{19}$$

The numerator of (19) describes the weighted relative frequency that a sample falls into the interval of the node $i(x)$. The denominator is the size of the interval, which is necessary to transform a relative frequency into a density over the interval $V_{i(x)}$. The density function $\hat{f}(x)$ can be equally defined for *internal nodes* (non-leaf nodes). This will be important below, where estimates at different levels of the tree are mixed for regularization.

In [32] we prove the following important convergence result:

Theorem 3. *With probability one, density trees converge to f as $N \to \infty$.*

The proof exploits the fact that our splitting rule ensures that (1) the depth of the tree grows without bounds and (2) the number of samples in each leaf also grows without bound as $N \to \infty$. The convergence then follows from the fact that f is Lipschitz.[2]

3.4 Regularization Through Shrinkage and Annealing

The continuous nature of the state space in GHMMs (and hence in MCHMMs), makes it possible to overfit *any* data set, no matter how large. This is because GHMMs are rich enough to assign a different state to each observation in the training data (of which there are only finitely many), making it essentially impossible to generalize beyond sequences other than the ones presented during training. A similar problem arises in conventional HMMs, if they are given more states than there are training examples. In GHMMs the problem is inherent, due to the topology of continuous spaces. Thus, some kind of regularization is needed to prevent overfitting from happening.

Our approach to regularization is based on *shrinkage* [30]. Shrinkage is a well-known statistical technique for "lumping together" different estimates from different data sources. In a remarkable result by Stein [30], shrinking estimators were proven to yield uniformly better solutions over unbiased maximum-likelihood estimators in multivariate Gaussian estimation problems. *Shrinkage trees* were introduced by McCallum [19]. Instead of using the density estimates at the leafs of a tree, shrinkage trees mix those densities with densities obtained further up in the tree. These internal densities are less specific to the region covered by a leaf node; however, their estimates are usually obtained from more data, making them less susceptible to variance in the data.

Figure 1c shows an example using shrinkage with an exponential factor, parameterized by ρ (with $0 \leq \rho \leq 1$). Here each node in the tree, with the exception of the root node, weighs its own density estimate by $(1 - \rho)$, and mixes it with the density estimate from its parent using the weighting factor ρ. As a result, every node along the path contributes to the density estimate at the leaf; however, its influence decays exponentially with the distance from the leaf node. Obviously, the value of ρ determines the amount of shrinkage. If $\rho = 1$, only the root node is consulted, hence, the probability density induced by the tree is uniform over its domain. If $\rho = 0$, on the other hand, there is no shrinkage and only the estimates in the leaf nodes determine the shape of the density function. For intermediate values of ρ, estimates along the entire path are combined.

Since the optimal value of ρ is problem-specific—de facto it depends on the nature of the (unobservable) state space—our approach uses annealing and cross validation to determine the best value for ρ. More specifically,

$$\rho^{(n)} \quad = \quad \bar{\rho}^{\,n-1} \tag{20}$$

where $\bar{\rho} < 1$ is a constant (e.g., 0.9) and n denotes the iteration counter of the Baum-Welch algorithm (starting at 1). Thus, ρ starts with $\rho^{(0)} = 1$, for which nothing can be learned, since every density is uniform. The parameter ρ is then annealed towards zero in an exponential fashion. Cross validation (early stopping) is applied to determine when to stop training.

4 Monte Carlo HMMs

We are now ready to present the main algorithm of this paper, along with the main theoretical result: The Monte Carlo algorithm for GHMMs, called *Monte Carlo hidden Markov models* (in short *MCHMM*). An MCHMM is a computational instantiation of a GHMM that represents all densities through samples and trees.

The learning algorithm for MCHMMs is depicted in Table 1. MCHMMs use both sample-based and tree representations during learning. In a nutshell,

- **E-step.** Particle filters are applied for forward and backward projection (c.f., Section 3.2 and Equations (7) and (8)), using the tree approximations of π, μ and ν.

  Forward and backward densities are multiplied (c.f., Equation (9)) by multiplying the importance factors of one of the sample sets (e.g., α_t) with their density values under the tree induced from the other sample set (e.g., β_t). This "trick" overcomes the problem discussed in

[2]The termination conditions for growing trees where chosen to facilitate the derivation of Theorem 3. In practice, these conditions can be overly restrictive, as they often require large sample sets to grow reasonable-sized trees. Our actual implementation sidesteps these conditions, and trees are grown all the way to the end. While the convergence results reported here are not applicable any longer, our implementation yielded much better performance specifically when small sample sets were used (e.g., 100 samples).

Model Initialization: Initialize $\lambda = \{\pi, \mu, \nu\}$ by three randomly drawn sets of samples of the appropriate dimension. Generate density trees from these samples. Set $\rho = 1$, and chose an initial sample set size $N > 0$.

E-step:

1. Computation of α_0 (c.f., (6)). Generate N samples from the tree representing π using likelihood-weighted sampling.
2. Computation of α_t (c.f., (7)). For each t with $1 < t \leq T$ do:
   (a) Generate N samples $\langle x', p_{x'} \rangle$ from the sample set representing α_{t-1} using likelihood-weighted sampling.
   (b) For each sample $\langle x', p_{x'} \rangle$, generate the conditional density $\mu(x \mid x')$ using the tree-version of μ. Sample a single x from this tree, using likelihood-weighted sampling.
   (c) Set p_x to a value proportional to $\nu(O_t \mid x)$, where O_t is the t-th observation in the data set. This density value is obtained using the tree representing ν.
   (d) Generate a tree from the new sample set.
3. Computation of β_T (c.f., (6)). Generate N uniformly distributed samples.
4. Computation of β_t (c.f., (8)). For each t with $T > t \geq 1$ do:
   (a) Generate N samples $\langle x', p_{x'} \rangle$ from the sample set representing β_{t+1} using likelihood-weighted sampling.
   (b) For each sample $\langle x', p_{x'} \rangle$, generate the conditional density $\mu(x' \mid x)$ using the tree-version of μ. Sample a single x from this tree, using likelihood-weighted sampling.
   (c) Set p_x to a value proportional $\nu(O_{t+1} \mid x')$, where O_{t+1} is the $t+1$-th observation in the data set. This density value is obtained using the tree representing ν.
   (d) Generate a tree from the new sample set.
5. Computation of γ_t (c.f., (9)). For each t with $1 \leq t \leq T$ do:
   (a) Generate $N/2$ sample from α_t by likelihood weighted sampling, and assign to each sample x an importance factor p_x proportional to $\beta_t(x)$, using the tree approximation of β_t.
   (b) Generate $N/2$ sample from β_t by likelihood weighted sampling and assign to each sample x an importance factor p_x proportional to $\alpha_t(x)$, using the tree approximation of α_t.

M-step:

1. Estimation of the new state transition density μ (c.f., (12)): Pick N random times $t \in \{1, \ldots, T-1\}$ and generate samples $\langle x, p_x \rangle$ and $\langle x', p_{x'} \rangle$ from γ_t, and γ_{t+1}, respectively, by likelihood-weighted sampling. Add $\langle (x, x'), N^{-1} \rangle$ into the sample set representing μ. Generate a tree from the sample set. When using $\mu(x' \mid x)$, condition the tree on x.
2. Estimation of the new observation density ν (c.f., (13)): Pick N random $t \in \{1, \ldots, T\}$ and generate a sample $\langle x, p_x \rangle$ from γ_t by likelihood-weighted sampling. Add $\langle (x, O_t), N^{-1} \rangle$ into the sample set representing ν. Generate a tree from the sample set. When using $\nu(b \mid x)$, condition the tree on x.
3. Estimation of the new initial state distribution π (c.f., (11)): Copy the sample set γ_0 into π. Generate a tree from the sample set.

Annealing: Set $\rho \leftarrow \rho\bar{\rho}$. Stop when the likelihood of an independent cross-validation set is at its maximum.

Sample set size: $N \longleftarrow \eta N$, for some $\eta > 1$.

Table 1: The MCHMM algorithm.

the beginning of Section 3.3, namely that with probability 1 no two samples in the sets representing α_t and β_t are the same. Its asymptotic correctness follows from the convergence results for sampling and density trees, and the convergence of importance sampling [28, 31].

- **M-step.** The new model $\lambda = \{\pi, \mu, \nu\}$ is obtained by generating samples from the initial state distribution (in the case of π), distributions of pairs of consecutive states (in the case of μ), and distributions of states paired with the corresponding observations (in the case of ν). Trees are then induced from these sample sets.

  The trees of μ and ν represent joint distributions, not conditional distribution. Hence, when using these trees they are conditioned on a state x (c.f., Equations (12) and (13)).

For regularization, MCHMMs use annealing and cross-validation to determine the best shrinkage factor. To ensure convergence, the number of samples N is increased over time.

The convergence of MCHMMs is guaranteed by the following important theorem:

Theorem 4 (MCHMM Convergence Theorem). *Under the assumptions stated above, any strictly monotonically increasing schedule of N's will cause an MCHMM to converge to a local maximum of the corresponding GHMM with probability 1.*

We omit the lengthy proof, which can be found in [32].

Once an MCHMM has been trained, it suffices to memorize only the tree-based version of the model $\lambda = \{\pi, \mu, \nu\}$; all sample sets can be discarded. Problems such as state tracking, prediction, analysis and discrimination can then typically be solved using particle filters (or variants/extensions thereof). By adapting the sample set size on-line, these methods can be implemented as *any-time* algorithms [4, 34] that adapt their computational complexity to the available resources. In addition, sampling in a likelihood-weighted manner places computation where needed: at regions with high likelihood. These properties make MCHMMs attrac-

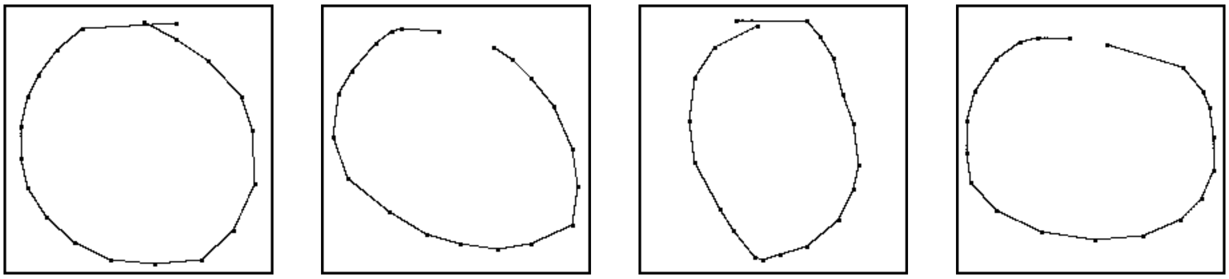

Figure 2: Examples of gestures. The first two gestures were drawn counterclockwise; whereas the other two were drawn clockwise. The MCHMM learning task is to differentiate them.

tive for real-time tracking and control [7].

5 Experimental Results

This section reports experimental results, obtained for MCHMMs applied to two problems: an artificial one which was chosen for demonstration purposes, and a gesture recognition problem. The results are primarily intended to illustrate MCHMMs in the finite sample case. Comparisons with existing HMM methods is beyond the scope of this paper.

The artificial data set consists of multiple oscillatory sequences. Observations are 10-dimensional and governed by

$$O_t = \begin{cases} \langle 0.25+\varepsilon_{t,1}, 0.25+\varepsilon_{t,2}, 0.25+\varepsilon_{t,3}, \\ \quad \ldots, 0.25+\varepsilon_{t,10}\rangle & \text{if } t \text{ odd} \\ \langle 0.75+\varepsilon_{t,1}, 0.75+\varepsilon_{t,2}, 0.75+\varepsilon_{t,3}, \\ \quad \ldots, 0.75+\varepsilon_{t,10}\rangle & \text{if } t \text{ even} \end{cases}$$

where $\varepsilon_{t,i}$ (with $1 \leq t \leq 20$ and $1 \leq i \leq 10$) are independent and identically distributed noise variables with zero-centered triangular distribution over $[-0.15; 0.15]$. To test the discriminatory accuracy of the learned model, we also generated a second, similar data set (using other independent noise variables ε):

$$O_t = \begin{cases} \langle 0.25+\varepsilon_{t,1}, 0.75+\varepsilon_{t,2}, 0.25+\varepsilon_{t,3}, \\ \quad \ldots, 0.75+\varepsilon_{t,10}\rangle & \text{if } t \text{ odd} \\ \langle 0.75+\varepsilon_{t,1}, 0.25+\varepsilon_{t,25}, 0.75+\varepsilon_{t,3}, \\ \quad \ldots, 0.25+\varepsilon_{t,10}\rangle & \text{if } t \text{ even} \end{cases}$$

The second data set consisted of a collection of hand-drawn gestures. Figure 2 shows examples. Once drawn, all gestures in our data base look quite similar. However, some of the gestures were drawn clockwise, whereas others were drawn counterclockwise. Here we are interested in discriminating clockwise from counterclockwise gestures. Notice that this problem is more difficult then the artificial one, as the observations alone (stripped of their temporal order) are insufficient for discrimination; instead, the MCHMM has to learn a meaningful model of the internal state.

Figures 3 shows results obtained for the first dataset. Shown in both figures are curves that characterize the log likelihood (notice that the log likelihood is unbounded for continuous distributions). Figure 3a illustrates that the log likelihood of the *training set* increases monotonically over multiple iterations of EM (Baum-Welch), for different experimental conditions described in the figure caption. Figure 3b shows testing results. The upper curve depicts the log likelihood for a set of random sequences that are generated from the same model as the training sequences. The bottom curve depicts the log likelihood for independently generated sequences using the other model, for which the MCHMM was not trained. In both cases, only a single data set (of length 20) was used for training, and testing was performed using 20 data sets generated independently. The initial shrinkage factor was $\rho = 1$, which was annealed down at the rate $\bar{\rho} = 0.9$. Figure 3b shows the effect of overfitting: the testing likelihood increases first, but then levels off (approx. iteration 6) and decreases. Using early stopping, we obtained 100% generalization accuracy in all experiments, even if as few as $N = 10$ samples were used.

Figure 4a shows the corresponding testing result for the more difficult gesture recognition problem. These results were obtained using a single gesture for training only, and using 50 gestures for testing. The top curve depicts the log likelihood for gestures drawn in the same direction as the training gesture, whereas the bottom curve shows the log likelihood for gestures drawn in the opposite direction. As easily noticed in Figure 4a, the difference between classes is smaller than in Figure 3b—which comes at no surprise—yet the likelihood of the "correct" class is still a factor of 4 to 5 larger than that of the opposite class. Figure 4a also shows the effect of annealing. The best shrinkage value is obtained after 12 iterations, where $\rho = 0.28$. As in the previous case, cross-validation using a single gesture performs well. On average, the classification accuracy when using cross-validation for early stopping is 86.0%. This rate is remarkably high, given that only a single gesture per class was used for training.

Figure 4b illustrates the trade-off between computation and accuracy empirically for the gesture data base. Shown there is the log likelihood of the testing data as a function of the sample set size N. Notice that the horizontal axis is logarithmic. All of these points are generated using a model λ obtained using early stopping. The sample set size was generated *after* training, to investigate the effect of com-

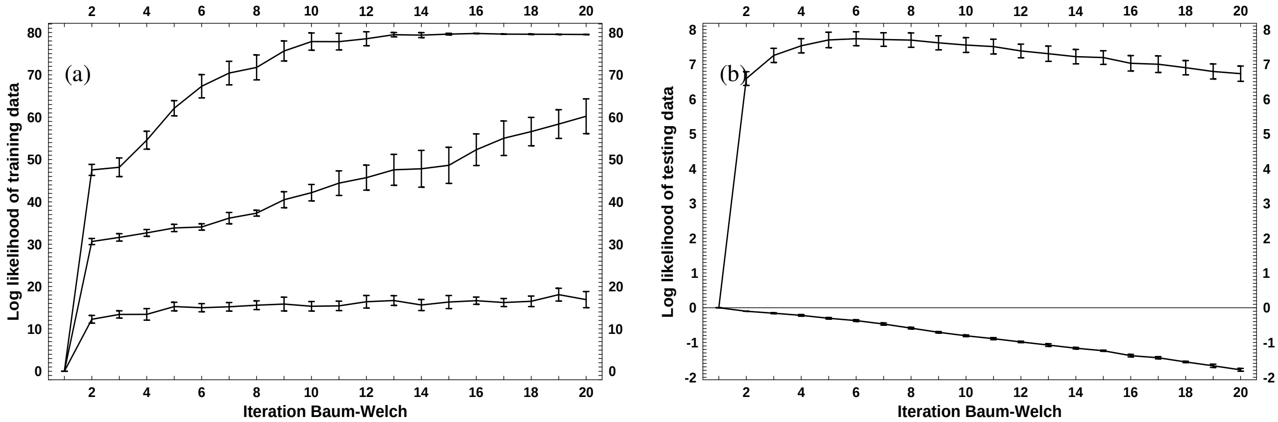

Figure 3: Log likelihood as a function of the iteration of EM, for the synthetic data set. **(a) Training**: Top curve: 1000 samples for all densities. Middle curve: 1000 samples for μ and ν, 100 samples for α, β, γ, and π. Bottom curve: 100 samples for μ and ν, 10 samples for α, β, γ, and π. **(b) Testing**: The top curve shows the log likelihood of "positive" training data (same orientation as used for training), whereas the bottom curve shows the log likelihood for the "negative" class (opposite orientation as used for training). Each result is averaged over 10 independent experiments; 95% confidence bars are also shown.

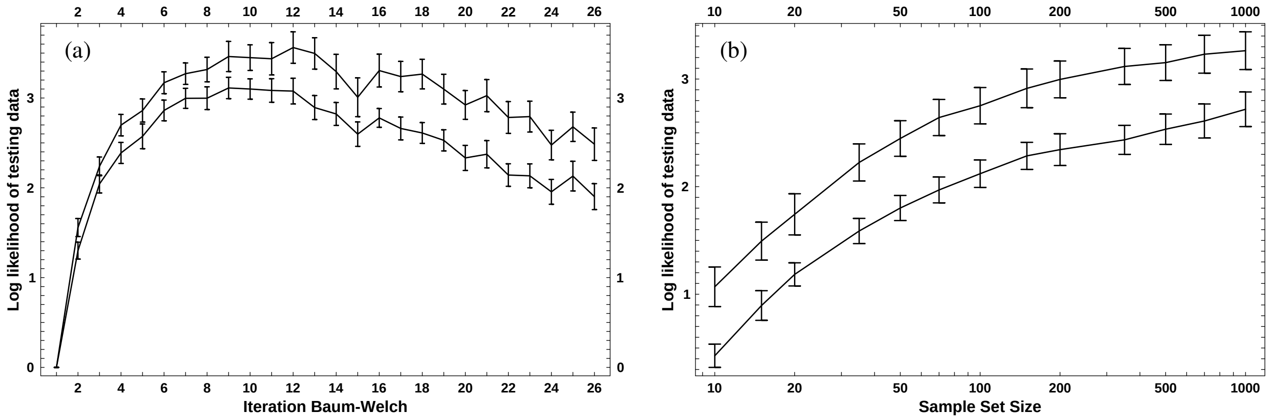

Figure 4: (a) Log likelihood for the gesture data base obtained for independent testing data. The top curve shows the log likelihood of gestures drawn in same same way as the training gesture, whereas the bottom curve shows the log likelihood of gestures drawn in opposite direction. (b) Trade-off between sample set size and log likelihood, for the gesture data set. The more samples are available, the higher the likelihood, and hence the more accurate the results. However, even extremely small sample sets yield reasonable discrimination.

putational limitations on the on-line performance of the MCHMM. As in Figure 4a, the top curve in Figure 4b depicts the log likelihood of gestures drawn in the same direction as the training data, whereas the bottom curve shows the log likelihood of gestures drawn in opposite direction. The result in Figure 4b illustrates that the likelihood (and hence the accuracy) of both data sets increases with the sample set size. This is not surprising, as the accuracy of the approximations increases with the sample set size. In addition, Figure 4b suggests that the distance between clockwise and counterclockwise gestures is approximately constant in log likelihood space (hence it grows in likelihood space). This illustrates that better results are achieved with larger sample sets. In our experiments, however, even small sample sets yielded good discrimination (after training with larger sample sets). For example, we observed on average 16% classification error with $N = 10$ samples.

For the gesture data base, one iteration of EM (using the settings described in the caption of Figure 3a) took the following time:

sample set size for μ, ν	sample set size for $\alpha, \beta, \gamma, \pi$	training time (in seconds)
100	100	0.852
1000	100	1.53
100	1000	10.36
1000	1000	18.38

After training, processing a single gesture (which involves running the particle filter on two different MCHMMs) took the following time:

sample set size for α	running time (in seconds)
10	0.00242
100	0.0242
1000	0.260

These numbers were obtained on a low-end PC (equipped with a 266Mhz Pentium II). The latter numbers illustrate that after learning, the processing of a gesture is several orders of magnitude faster than the hand motion involved in gesture generation.

6 Conclusion

We have presented Monte Carlo Hidden Markov Models (MCHMMs), a new algorithm for hidden Markov model learning. MCHMMs extend HMMs to real-valued state and observation spaces and non-parametric models. They represent all densities by samples, which are transformed into probability density functions using density trees. Because the continuous state spaces are rich enough to fit (and overfit!) arbitrary data sets, our approach uses shrinkage to reduce its complexity. The shrinkage parameter is gradually annealed down over time, and cross-validation (early stopping) is used for model selection (regularization).

In this paper, MCHMMs have been shown to converge to local maxima in likelihood space for a large non-parametric class of probability density functions. Empirical results, carried out in an artificial and a gesture recognition domain, demonstrate that our approach generalizes well even when trained with extremely scarce data. Additional experiments characterize the natural trade-off between sample set size and accuracy, illustrating that good results may be achieved even from extremely small sample sets. Additional results described in [32] illustrate the effect of *smoothing* in the forward/backward project step.

The choice of density trees for representing densities in MCHMMs is somewhat ad hoc. In principle, a range of methods could be applied, such as Gaussian kernels. The trade-offs involved in choosing the right density model are currently poorly understood and therefore warrant future research.

Acknowledgement

The authors gratefully acknowledge inspiring discussions with Frank Dellaert, Nir Friedman, Daphne Koller, and Larry Wasserman.

This research is sponsored in part by NSF (CAREER Award IIS-9876136) and DARPA via TACOM (contract number DAAE07-98-C-L032), and Rome Labs (contract number F30602-98-2-0137), which is gratefully acknowledged. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of NSF, DARPA, TACOM, Rome Labs, or the United States Government.

References

[1] L.E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *Annals of Mathematical Statistics*, 37, 1966.

[2] T. Briegel and V. Tresp. A Monte Carlo generalized EM-type algorithm for state and parameter estimation in nonlinear state space models. *NIPS-98*.

[3] E. Charniak. *Statistical Language Learning*. MIT Press, 1993.

[4] T. L. Dean and M. Boddy. An analysis of time-dependent planning. *AAAI-92*.

[5] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo Localization For Mobile Robots *ICRA-99*.

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

[7] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte Carlo Localization: Efficient position estimation for mobile robots. *AAAI-99*.

[8] Z. Ghahramani and S. Roweis. Learning nonlinear stochastic dynamics using the generalized EM algorithm. *NIPS-98*.

[9] N. Gordon. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings (F)*, 140(2), 1993.

[10] B. Hannaford and P. Lee. Hidden markov model analysis of force torque information in telemanipulation. *International Journal of Robotics Research*, 10(5), 1991.

[11] J. Henderson, S. Salzberg, and K. Fasman. Finding genes in human DNA with a hidden Markov model. *ISMB-96*.

[12] M. Isard and A. Blake. Condensation: conditional density propagation for visual tracking. *International Journal of Computer Vision*, 1998.

[13] B.H. Juang. Maximum likelihood estimation for mixture multivariate stochastic observations of markov chains. *AT&T Technical Journal*, 64(6), 1985.

[14] K. Kanazawa, D. Koller, and S.J. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. *UAI-95*.

[15] D. Koller and R. Fratkina. Using learning for approximation in stochastic processes. *ICML-98*.

[16] A. Krogh, I.S. Mian, and D. Haussler. A hidden markov model that finds genes in e. coli dna. *Nucleic Acids Research*, 22, 1994.

[17] L.A. Liporace. Maximum likelihood estimation for multivariate observations of markov sources. *IEEE Transactions in Information Theory*, 28(5), 1982.

[18] J. Liu and R. Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93, 1998.

[19] A. McCallum, R. Rosenfeld, T. Mitchell, and A.Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. *ICML-98*.

[20] G.J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley, 1997.

[21] M. Meilă and M.I. Jordan. Learning fine motion by markov mixture of experts. *NIPS-96*.

[22] A.W. Moore, J. Schneider, and K. Deng. Efficient locally weighted polynomial regression predictions. *ICML-97*.

[23] R.M. Neal. Probabilistic inference using markov chain monte carlo methods. TR CRG-TR-93-1, University of Toronto, 1993.

[24] R.M. Neal and G.E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*. Kluwer, 1998.

[25] S. M. Omohundro. Bumptrees for Efficient Function, Constraint, and Classification Learning. *NIPS-91*.

[26] M. Pitt and N. Shephard. Filtering via simulation: auxiliary particle filter. *Journal of the American Statistical Association*, 1999.

[27] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, 1989.

[28] D.B. Rubin. Using the SIR algorithm to simulate posterior distributions. In *Bayesian Statistics 3*. Oxford University Press, 1988.

[29] H Shatkay and L. Kaelbling. Learning topological maps with weak local odometric information. *IJCAI-97*.

[30] C. Stein. Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability 1*, 1995.

[31] M.A. Tanner. *Tools for Statistical Inference*. 2nd ed., Springer, 1993.

[32] S. Thrun and J. Langford. Monte carlo hidden markov models. TR CMU-CS-98-179, Carnegie Mellon University, 1997.

[33] A. Waibel and K.-F. Lee, editors. *Readings in Speech Recognition*. Morgan Kaufmann, 1990.

[34] S. Zilberstein and S. Russell. Approximate reasoning using anytime algorithms. In *Imprecise and Approximate Computation*. Kluwer, 1995.