Fast Inference and Learning in Large-State-Space HMMs

Sajid M. Siddiqi SIDDIQI@CS.CMU.EDU
Andrew W. Moore AWM@CS.CMU.EDU
The Robotics Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213

Abstract

For Hidden Markov Models (HMMs) with fully connected transition models, the three fundamental problems of evaluating the likelihood of an observation sequence, estimating an optimal state sequence for the observations, and learning the model parameters, all have quadratic time complexity in the number of states. We introduce a novel class of non-sparse Markov transition matrices called Dense-Mostly-Constant (DMC) transition matrices that allow us to derive new algorithms for solving the basic HMM problems in sub-quadratic time. We describe the DMC HMM model and algorithms and attempt to convey some intuition for their usage. Empirical results for these algorithms show dramatic speedups for all three problems. In terms of accuracy, the DMC model yields strong results and outperforms the baseline algorithms even in domains known to violate the DMC assumption.

1. Introduction

Hidden Markov Models are a popular tool for modeling the statistical properties of observation sequences in domains where the observations can be assumed to be indicative of an underlying hidden state sequence. Introduced in the late 1960s, HMMs have been used most extensively in speech recognition (Rabiner, 1989; Bahl et al., 1983) and bioinformatics (El-Difrawy & Ehrlich, 2002) but also in diverse application areas such as computer vision and information extraction (Brand et al., 1997; Seymore et al., 1999). An excellent tutorial on HMMs can be found in the work of Rabiner (1989).

With ever-increasing amounts of data becoming available in highly varied domains, it is important for HMM algorithms to be able to scale accordingly for large state spaces. For example, Felzenszwalb et al. (2003) note in their work on internet usage data that there were brief but significant events that could only be captured by large HMMs of a hundred or more states. This is hindered by the fact that the three fundamental algorithms for HMMs are quadratic in the number of states.

We propose a paradigm for HMM inference and learning that yields sub-quadratic algorithms for all three basic HMM operations without resorting to sparse transition models. We contend that for large-state-space HMMs, it is usually reasonable to focus on a few high-probability transitions per state and keep track of their exact values while approximating the rest with a constant. We call this a Dense-Mostly-Constant (DMC) transition matrix. For brevity, we shall use the term 'DMC HMM' to refer to an HMM whose transition matrix obeys the DMC condition. The number of transitions per state that are modeled exactly is represented by K. This set of 'important' transitions is chosen dynamically and need not be specified *a priori*. Indeed, the algorithm is designed to automatically detect and choose the K highest transition probabilities per state. Here is an example of a regular 3×3 transition matrix transformed to a DMC transition matrix with $K = 1$:

$$\begin{pmatrix} 0.5 & 0.3 & 0.2 \\ 0.1 & 0.3 & 0.6 \\ 0.05 & 0.8 & 0.15 \end{pmatrix} \Rightarrow \begin{pmatrix} \mathbf{0.5} & 0.25 & 0.25 \\ 0.2 & 0.2 & \mathbf{0.6} \\ 0.1 & \mathbf{0.8} & 0.1 \end{pmatrix}$$

The highest entry in each row is preserved exactly, and the remaining probability mass is evenly distributed among the remaining states. Each row of the matrix may have a different constant. This idea generalizes for higher values of N and K. We will also show how our modified learning algorithm for DMC HMMs can automatically discover the K highest transition probabilities for each state without computing them all.

The layout of this paper is as follows. We discuss some related work in Section 2. In Section 3 we briefly

Appearing in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.

review the three primary operations on HMMs and go on to see how the DMC condition is incorporated into the HMM model. Section 4 covers our new algorithms for the three HMM operations of estimation, inference and learning under the DMC condition. In Section 5 we present experimental results on both synthetic and real-world data sets to investigate the speedup achieved by our new algorithms and the accuracy tradeoff that is made with these models compared to regular HMMs. We also explore the interaction between the number of states (N) and the number of exact transition probabilities per state (K). Finally, in Section 6 we summarize the ideas and results presented in this paper and discuss how they may be extended in the future.

2. Related Work

Felzenszwalb et al. (2003) recently proposed fast algorithms for a class of HMMs where the states can be embedded in an underlying parameter space and the transition probabilities can be expressed in terms of distances in this space. In comparison, our new algorithms are made possible by introducing a class of transition models that are applicable to arbitrary state spaces. Assuming a sparse transition matrix is another way to speed up these algorithms, and is an underlying assumption in many cases. This has two drawbacks. Firstly, this is an overly restrictive assumption for many problem domains. Secondly, it requires the sparse structure to be known or extracted in advance. Other approaches for fast HMM algorithms include the work by Murphy and Paskin (2002), which treats HMMs as a special kind of Dynamic Bayesian Network and proposes faster inference algorithms for hierarchical HMMs, as well as Salakhutdinov et al. (2003) which derives an alternative learning algorithm for HMM parameters under conditions when EM is slow.

3. HMMs with DMC Transition Matrices

3.1. Basic Operations on HMMs

We will use the notation and terminology of Rabiner (1989), as well as his three canonical problems for HMMs.

Given an observation sequence O of length T and an HMM model $\lambda = (A, B, \pi)$ with N states, the three basic problems of HMMs are:

1. Computing the probability $P(O|\lambda)$ of an observation sequence O given the model λ.

2. Inferring an optimal state sequence $Q = q_1 q_2 ... q_T$ given the observation sequence O and model λ, i.e. a state sequence that maximizes $P(Q|O, \lambda)$.

3. Updating the model parameters λ to best suit the observation sequence O, i.e. to maximize $P(O|\lambda)$.

All these computations can be carried out in $O(TN^2)$ time using well-known methods. The forward part of the forward-backward procedure computes $P(O|\lambda)$. The Viterbi algorithm (Viterbi, 1967) yields an optimal state sequence Q to maximize $P(Q|O, \lambda)$, and Baum-Welch (Baum, 1972) is used to re-estimate the model to maximize $P(O|\lambda)$.

3.2. Incorporating the DMC Condition into HMMs

Before we discuss how the DMC condition allows us to carry out the three basic operations more efficiently, we will see how HMMs are modified under the DMC condition. We add a new parameter K, the number of transitions per row whose values are calculated exactly. Therefore, under the DMC condition with a particular value of K, the transition matrix A is constrained so as to contain K non-constant transition probabilities per row, with the rest of the $N - K$ transition probabilities being equal to a per-row constant value such that each row of A still sums to 1. We also define $NC = \{NC_i\}$ to be a collection of N K-sized lists where $NC_i = \{j : a_{ij}$ is a non-constant transition probability$\}$.An example form of a 5×5 DMC transition matrix with 2 non-constant probabilities (a_{ij}) and 3 constant (c_i) probabilities per row is

$$A_{DMC} = \begin{pmatrix} c_1 & \mathbf{a_{12}} & c_1 & \mathbf{a_{14}} & c_1 \\ \mathbf{a_{21}} & \mathbf{a_{22}} & c_2 & c_2 & c_2 \\ c_3 & c_3 & \mathbf{a_{33}} & c_3 & \mathbf{a_{35}} \\ c_4 & \mathbf{a_{42}} & \mathbf{a_{43}} & c_4 & c_4 \\ c_5 & c_5 & c_5 & \mathbf{a_{54}} & \mathbf{a_{55}} \end{pmatrix}$$

where the constants c_i are related to the non-constants as:

$$c_i = (1 - \sum_{j \in NC_i} a_{ij})/(N - K) \tag{1}$$

The other HMM parameters (B, π) are unaffected by the DMC condition. Applying the DMC condition on the transition matrix can be considered a form of parameter tying (Bahl et al., 1983). However, conventional parameter tying methods focus on tying observation model parameters of different states rather than transition probabilities.

4. Algorithms

We now outline the standard algorithms for the most common operations on HMMs, and describe our faster equivalents for DMC HMMs. Problems 1 and 2 from Section 3 are the same for DMC HMMs as they are for regular HMMs. In Problem 3, an additional constraint for us is that the resulting HMM should also conform to the DMC condition.

4.1. Calculating $P(O|\lambda)$

$P(O|\lambda)$ is computed using the forward part of the forward-backward procedure. Let $\alpha_t(i)$, the forward variable, be defined as $\alpha_t(i) = P(O_1 O_2 \cdots O_t, q_t = S_i|\lambda)$ The regular algorithm for constructing the $T \times N$ α matrix is, inductively,

$$\alpha_1(i) = \pi_i b_i(O_1) \tag{2}$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \tag{3}$$

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{4}$$

Since there is an N-term summation carried out for each state per timestep, and there are $N \times T$ $\alpha_t(i)$ values to be calculated, the total running time of the forward procedure is $O(TN^2)$.

For DMC HMMs, we can take advantage of the structure of the transition matrix and calculate the α values, and thus $P(O|\lambda)$, more efficiently. The following inductive step is possible when the transition matrix is DMC:

$$\begin{aligned}\alpha_{t+1}(j) &= \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \qquad (5)\\ &= \left[\sum_{i:j \in NC_i} \alpha_t(i) a_{ij} + \sum_{i:j \notin NC_i} \alpha_t(i) c_i\right] b_j(O_{t+1})\\ &= \left[\sum_{i=1}^{N} \alpha_t(i) c_i + \sum_{i:j \in NC_i} \alpha_t(i)(a_{ij} - c_i)\right] b_j(O_{t+1})\end{aligned}$$

Instead of computing the N-term summation for each state per timestep, we break it up into two summations. The first is an N-term summation without any mixed j terms, allowing us to compute it just once for each of the T rows of the α matrix. The second summation sums over an average of K terms for each $\alpha_t(j)$ value. The first summation adds an amortized cost of $O(1)$ over the N states, and the second one adds a $O(K)$ cost. Overall, this results in a time complexity of $O(TNK)$ for calculating the α matrix, and thus for $P(O|\lambda)$. Note that calculating the α matrix also allows fast computation of state occupation probabilities $P(q_t = S_i|O_1, \cdots, O_t, \lambda)$. The backward variables $\beta_t(i) = P(O_{t+1} O_{t+2} \cdots O_T|q_t = S_i, \lambda)$ are calculated similarly.

4.2. Finding an Optimal State Sequence

Given an observation sequence O, the Viterbi algorithm is used to calculate the state sequence Q that maximizes $P(Q|O, \lambda)$. Define $\delta_t(i)$ as

$$\delta_t(i) = \max_{q_1, \cdots, q_{t-1}} P[q_1 q_2 \cdots q_t = i, O_1 O_2 \cdots O_t|\lambda] \tag{6}$$

The inductive formula for $\delta_t(j)$ used in the Viterbi algorithm is $\delta_{t+1}(j) = [\max_i \delta_t(i) a_{ij}] b_j(O_{t+1})$. Since there is a maximization over N terms carried out for each state per timestep, and there are $N \times T$ $\delta_t(i)$ values to be calculated, the total running time of the Viterbi algorithm is $O(TN^2)$. Under the DMC condition, however, we can calculate this maximization more efficiently.

$$\begin{aligned}\delta_{t+1}(i) &= \left[\max_i \delta_t(i) a_{ij}\right] b_j(O_{t+1}) \qquad (7)\\ &= \left[\max\left\{\max_i \delta_t(i) c_i, \max_{i:j \in NC_i} \delta_t(i) a_{ij}\right\}\right] b_j(O_{t+1})\end{aligned}$$

Analogously to (5), we can split the $O(N)$ maximization into two terms: a maximization over N terms that is common to the entire timestep, and a maximization over an average of K terms per state.

4.3. Learning a Maximum Likelihood Model

Maximum likelihood estimates of the HMM parameters are obtained by repeatedly iterating the Baum-Welch algorithm until the likelihood of the data on the model converges. We present a new re-estimation procedure only for the transition matrix A since other parts of the HMM learning procedure are unaffected by the DMC condition. The main challenge is to ensure we pick the K largest transition probabilities per state without doing the $O(TN^2)$ work done by regular Baum-Welch.

First, we need a few more standard definitions. The set of forward variables $\alpha_t(i)$ and backward variables $\beta_t(j)$ were defined earlier. Two more sets of variables we need are the point-wise state variables and the point-wise transition variables:

$$\gamma_t(i) = P(q_t = S_i|O, \lambda) \tag{8}$$

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j|O, \lambda) \tag{9}$$

Once these variables have been calculated, the update equation for the transition probabilities is:

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{10}$$

We will see that (10) and its requisite variables can be computed in less than $O(TN^2)$ time while at the same time automatically choosing the correct K important transitions for every state. A sub-quadratic procedure for computing the forward and backward variables has already been described in Section 4.1. The γ matrix takes $O(TN)$ time and is easily carried out given the α and β matrices: $\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}$. We then move to the harder problem of calculating the numerator of (10) in less than $O(TN^2)$ time.

4.4. Computing the pointwise transition variables ξ

The T ξ_t matrices each contain N^2 terms to be calculated, resulting in a $O(TN^2)$ running time for computing all required ξ terms. In the regular Baum-Welch routine, each term is computed as follows:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} \beta_{t+1}(j) b_j(O_{t+1})}{P(O|\lambda)} \tag{11}$$

The DMC condition is insufficient for obtaining a $O(TNK)$ algorithm for the ξ terms, since the DMC condition does not imply any exploitable structure in the α and β matrices. Instead we outline a $\Omega(TNK + RN^2)$ algorithm with a worst case complexity of $O(TN^2)$, but which runs much faster in practice. R is a new parameter described later. Define S to be an $N \times N$ matrix such that $S(i,j) = \sum_{t=1}^{T-1} \xi_t(i,j)$. Towards this end, for a particular row i, it is useful to think of the $\bar{a}_{ij}$ update equation (10) as $\bar{a}_{ij} \propto S(i,j)$ since the denominator in (10) is constant with respect to j, and the numerator is the main computational challenge. Because of the DMC condition, only K exact $S(i,j)$ terms will be kept in each row of the S matrix, but at the same time we would like to ensure that our algorithm picks the $S(i,j)$ terms with the largest values.

4.4.1. Lazy dot-product evaluation for the S matrix

Define the κ matrix as $\kappa_t(j) = \beta_{t+1}(j) b_j(O_{t+1})$. From (11), we can consider each S_{ij} term to be the dot product of column i of the α matrix and column j of the κ matrix. Calculating the S matrix involves computing $\frac{N^2}{2}$ dot-products between vectors of length T, giving us an overall $O(TN^2)$ running time. However, even if we computed the exact S matrix we would then average out the smaller $N-K$ entries in each row in order to maintain the DMC property, thus wasting much of our work. To avoid this situation, we adopt a lazy evaluation approach where we first *partially* compute each dot-product in the S matrix along with an upper bound on each S_{ij} term. A new parameter $R \in \{1, \ldots, T\}$ is defined to determine how much of the S matrix we partially compute in the beginning. We first compute some intermediate variables:

for $i = 1$ to N **do**
  $\alpha^*(i) \leftarrow$ R'th largest $\alpha_t(i)$ value
  $T_{\alpha(i)} \leftarrow \{t : \alpha_t(i) > \alpha^*(i)\}$
  $\alpha^s(i) \leftarrow \sum_{t \notin T_{\alpha(i)}} \alpha_t(i)$ // sum of small $\alpha(i)$'s
end for
for $j = 1$ to N **do**
  $\kappa^*(j) \leftarrow$ R'th largest $\kappa_t(j)$ value
  $T_{\kappa(j)} \leftarrow \{t : \kappa_t(j) > \kappa^*(j)\}$
  $\kappa^s(j) \leftarrow \sum_{t \notin T_{\kappa(j)}} \kappa_t(j)$ // sum of small $\kappa(j)$'s
end for
for $i = 1$ to N **do**
  for $j = 1$ to N **do**
    $L_{ij} \leftarrow T_{\alpha(i)} \cup T_{\kappa(j)}$ // set of indices t where $\alpha_t(i)$ or $\kappa_t(j)$ are large
  end for
end for

We can now compute a matrix of partial $S(i,j)$ calculations S_p, and an upper bound matrix U, by explicitly computing summation terms that have a large $\alpha_t(i)$ or $\kappa_t(j)$ term[1] and bounding the rest of the summation.

$$\begin{aligned} S(i,j) &= \sum_{t \in L_{ij}} \alpha_t(i)\kappa_t(j) + \sum_{t \notin L_{ij}} \alpha_t(i)\kappa_t(j) \\ &\leq \sum_{t \in L_{ij}} \alpha_t(i)\kappa_t(j) + \min\{\alpha^*(i)\kappa^s(j), \alpha^s(i)\kappa^*(j)\} \end{aligned}$$

This inequality allows us to define S_p and U as:

$$\begin{aligned} S_p(i,j) &\leftarrow \sum_{t \in L_{ij}} \alpha_t(i)\kappa_t(j) \\ U(i,j) &\leftarrow S_p(i,j) + \min\{\alpha^*(i)\kappa^s(j), \alpha^s(i)\kappa^*(j)\} \end{aligned}$$

The extent to which we partially compute each dot-product, i.e. the size of L_{ij}, is determined by the parameter R. When partially computing the $S(i,j)$ terms, we carry out $O(RN^2)$ amount of work. The larger an R we choose, the costlier will be our partial computation. However, this will also make $U(i,j)$ a tighter bound for $S(i,j)$, saving us more work later on. Once we have the upper bound matrix U, we fully

[1] To be precise, we work with the scaled variables $\alpha'_t(i) = \frac{\alpha_t(i)}{P(O_1,\cdots,O_t)}$ and $\kappa'_t(j) = \frac{\kappa_t(j)}{P(O_{t+1},\cdots,O_T|O_1,\cdots,O_t)}$ since the raw α and κ values vary monotonically with time. Conveniently, $\alpha'_t(i)\kappa'_t(j) = \frac{1}{P(O|\lambda)}\alpha_t(i)\kappa_t(j)$.

compute as few dot-products as possible in order to obtain the desired DMC S matrix:

```
1: for i = 1 to N do
2:    Ũ_i ← row i of U sorted in decreasing order
3:    j ← 1
4:    repeat
5:       S̃_i(j) ← compute full dot-product to get exact
         S value bounded by Ũ_i(j)
6:       j ← j + 1
7:    until at least K exact S̃_i terms have been com-
      puted, and the smallest of them is larger than
      all remaining Ũ_i terms
8:    Rearrange K largest S̃_i(j) terms to get the re-
      quired K non-constant terms in row i of S
9: end for
```

Each exact $S(i,j)$ computation in step 5 above is a $O(T)$ operation. The termination condition of the inner loop is designed to ensure that we compute as few exact terms as possible while still detecting the largest K terms. This means that the inner loop could iterate as few as K times or as many as N times, depending on the tightness of the upper bounds in U, which in turn depends on R. As a result, the overall running time for calculating the S matrix (and by implication, the transition matrix) under the DMC condition can range from $O(TNK + RN^2)$ to $O(TN^2)$ in the theoretical worst case, though empirically the average running time for this procedure was always observed to be much faster than $O(TN^2)$.

5. Experiments

We implemented the regular and DMC HMM algorithms in ANSI C with parameter scaling measures in place to avoid numerical underflow problems (Rabiner, 1989). The inputs are real-valued, multi-attribute observations modeled as a multi-dimensional axis-aligned Gaussian for every state. We also implemented routines to generate regular and DMC HMMs of arbitrary sizes and simulate data from these HMMs. These routines are used in generating the synthetic datasets on which we show our results. Another dataset we use to show results contains 14,720 rows of two-dimensional, real-valued time series data obtained from accelerometers worn at two positions on a person's body over the course of his daily routine, with 1 datapoint being obtained every minute. We believe the activity monitoring domain lends itself naturally to the DMC assumption since people tend to persist in the same activity for long durations, and then transition with a high probability to a small set of related activities, though sometimes they make unusual transitions. We refer to this as the Motionlogger dataset[2]. The R parameter for DMC HMMs is heuristically chosen to be $T/20$ in these experiments.

In conducting experiments, we would like to investigate the following:

1. Running times of algorithms for basic HMM operations, on regular and DMC HMMs.

2. The accuracy tradeoff incurred by assuming a DMC HMM model on different datasets where the DMC assumption is obeyed or violated, and how this tradeoff varies with respect to K. Accuracy is measured by comparing the test set likelihoods of the regular and DMC HMMs that were learned with Baum-Welch on training data.

3. Tradeoffs between N and K in the DMC HMM model, and how different choices of N and K affect the data likelihood in different domains.

5.1. Speed

We study the running times of DMC and regular HMM algorithms on synthetic data. Since both sets of algorithms are linear in the number of observations, we focus on varying the number of states to highlight differences between the regular and DMC HMM algorithms while holding the number of observations constant. The speedup brought about by the DMC assumption is appreciable for HMMs of 10-15 states or more. However, we show results for much larger HMMs here to demonstrate how the algorithm scales.

Figure 1(A) illustrates the time taken for the Viterbi algorithm as it varies with the number of states in the HMM for a regular HMM, and the times taken for our DMC version of Viterbi for DMC HMMs with $K = 20$, $K = 50$ and $K = 100$. The dataset being used here consists of 2000 rows of two-dimensional observations. The difference is very clear here since the regular Viterbi algorithm is quadratic in N and the DMC version is linear in N for fixed K.

Evaluating the likelihood of an observation sequence on an HMM is equivalent to computing the α matrix. As discussed in Section 4.1, this procedure is also $O(TNK)$ under the DMC condition, and the running time improvement is exactly the same as shown in Figure 1(A) for the Viterbi algorithm.

Figure 1(B) shows how the time taken per iteration of Baum-Welch varies with the number of states in the HMM. As N increases, the speedup is apparent. The

[2] http://www-2.cs.cmu.edu/∼siddiqi/data/motionlogger

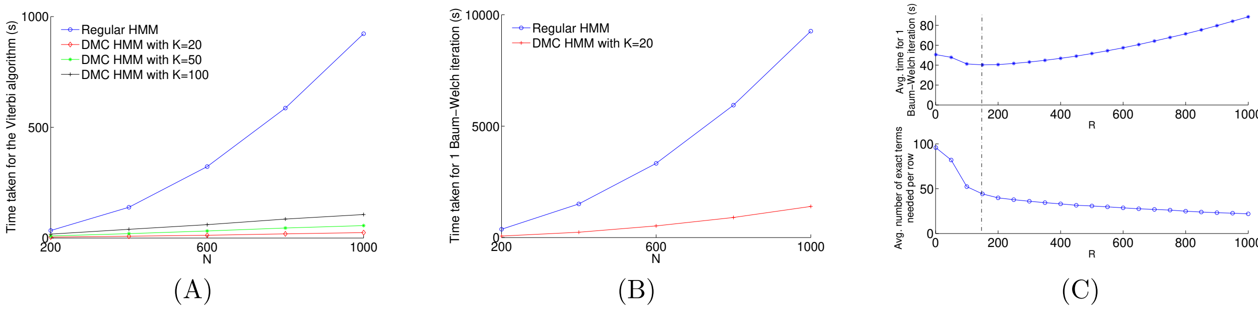


Figure 1. A. Running times (in seconds) for the Viterbi algorithm in regular and DMC HMMs. B. Running times for one Baum-Welch iteration in regular and DMC HMMs. C. Baum-Welch running time as a function of R (top panel), along with the average number of transition matrix terms needed to be exactly computed at each R value (bottom panel). The dotted line indicates the optimal value of R.

regular Baum-Welch algorithm is clearly quadratic, whereas the DMC version, though not quite linear, is still much faster.

As we discussed in Section 4.4.1, R is a tuning parameter whose optimal value depends on the particular dataset. In the upper panel of Figure 1(C) we see how the average running time for one iteration of Baum-Welch varies with R while learning a 100-state DMC HMM with $K = 10$. The dataset consists of 10,000 observations generated from a 100-state DMC HMM with $K = 10$. In the lower panel we plot the average number of entries per row of the transition matrix needed to be calculated exactly during the procedure to calculate the S matrix. This is equivalent to the average number of inner loop iterations during S matrix calculation in Section 4.4.1. Ideally this value should be as close to K as possible at the value of R which yields the fastest average running time, in order to achieve a running time of $O(TNK + RN^2)$. In the case of Figure 1(C) it is much less than 100 but still larger than 10. Therefore, though we do not achieve the lower bound running time, the DMC transition matrix re-estimation procedure is much faster than the worst case $O(TN^2)$ running time.

5.2. Accuracy

We would like to measure how well DMC HMMs model different data sets compared to their regular HMM counterparts. We do this by learning regular and DMC HMMs with different values of K using Baum-Welch on a training data set until the likelihood stops improving on a test set, and examining the test set log-likelihood scores. The three data sets we use include two synthetic and one real-world data set. For each set of experiments we include as baseline experiments a 20-state HMM with a uniform transition matrix (es-

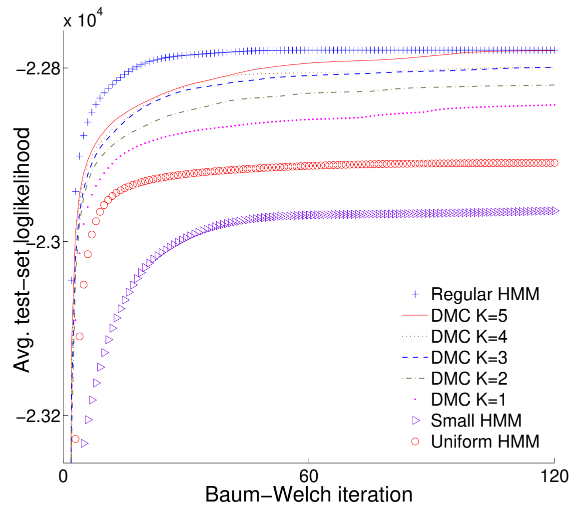
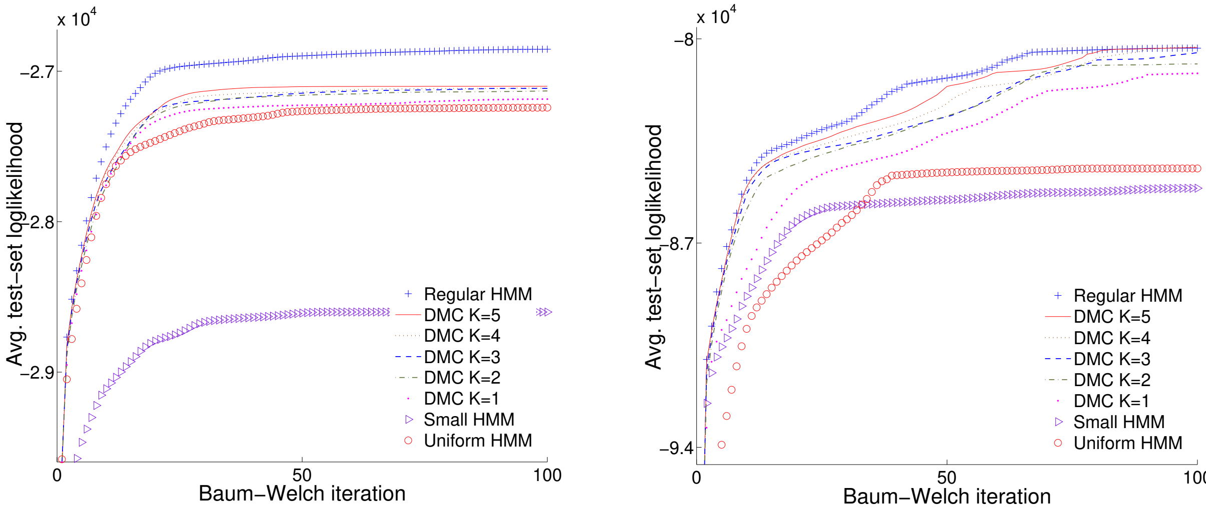


Figure 2. DMC-friendly data set learning curves for 20-state DMC HMMs with different K values, compared to a 20-state regular HMM and two control models.

sentially a gaussian mixture model) to see whether the transition matrix is essential, and a 5-state regular HMM to see whether a large number of states are needed. The synthetic data sets are generated from HMMs with highly overlapping states, since gaussian HMM data with well-separated states can be adequately modeled with a gaussian mixture model. The Anti-DMC data sets were generated from a 20-state HMM whose transition model is designed to break the DMC assumption; the transition probabilities are proportional to the sequence $1, 2, 3, ..., 20$ in each state. The DMC-friendly data set is generated from an HMM with a DMC transition matrix, where the top 5 transition probabilities for each state capture 60% of the probability mass. Both synthetic data sets consist of 25,000 datapoints, with 20,000 used for training and the rest as a test set. From the Motionlogger data,



(A) (B)

Figure 3. Learning curves for 20-state DMC HMMs with different K values on the (A) Anti-DMC and (B) Motionlogger data sets, compared to a 20-state regular HMM and two control models.

10,000 rows are used for training and the rest for testing. Each of the learning curves is calculated by averaging 5 runs of Baum-Welch at every iteration.

5.2.1. Comparing Regular and DMC HMMs

Figures 2 and 3 show learning curves for regular and DMC HMMs for the different data sets with different parameter settings, along with a regular 5-state HMM and uniform 20-state HMM as the baseline. Figure 2 compares a regular 20-state HMM to 20-state DMC HMMs with K ranging from 5 to 1, on the DMC-friendly data set. The DMC model with K=5 achieves the likelihood achieved by the full model, while the other DMC models still outperform the baselines by a large margin. Keeping track of even a single transition probability parameter per state (as the DMC model with K=1 does) makes a large difference as compared to the uniform HMM (which can be thought of as a DMC model with K=0). Figure 3(A) shows similar learning curves for the Anti-DMC data set. Here the DMC models beat the uniform HMM by a smaller margin, and the DMC HMM with K=1 is almost the same as the uniform HMM. The Anti-DMC data set is designed to confound the DMC assumption, meaning that the 15 transition probabilities being treated as constant for each state actually vary by a large amount, and they contain a large portion of the probability mass for each state. Figure 3(B) shows similar learning curves on the Motionlogger data set. Here the DMC models do much better, with the K=5 model achieving the likelihood of the full HMM, and all the DMC models outperforming the baselines by a large margin. This indicates that the data is well-suited to the DMC assumption.

Table 1. Average test set log-likelihoods at convergence for regular and DMC HMMs with 100 transition parameters on the (A) DMC-friendly, (B) Anti-DMC, and (C) Motionlogger data sets.

HMM type	A	B	C
Regular $N = 10$	-22829	-27681	-81948
DMC $N = 20, K = 5$	**-22780**	-27098	-80101
DMC $N = 25, K = 4$	-22783	-27076	-80170
DMC $N = 50, K = 2$	-22807	**-27054**	-79616
DMC $N = 100, K = 1$	-22847	-27070	**-79283**

5.2.2. Tradeoffs between N and K

We would like to study the tradeoff between the DMC parameter K and the number of states in the HMM. For all three data sets considered above, we train a number of different models and compare their average test set log-likelihoods at convergence. We keep the number of transition parameters constant as we decrease K and increase the number of states, and see how the resulting models fare on test sets. In Table 1, we see that, among HMMs with 100 transition parameters, our three data sets are best modeled by DMC HMMs with different pairs of (N, K) values. When varying N and K in this manner, we are giving the HMM more parameters to fit features in the data, but fewer parameters per state to model its transition patterns. Different (N, K) pairs may be explored to find settings that best explain the data, where the space of (N, K) values considered is bounded by running time considerations or domain knowledge.

6. Discussion

The DMC assumption induces just enough structure in the transition matrix to allow faster algorithms for the three canonical HMM problems, without imposing so much structure as to be overly restrictive as in the case of sparse transition models. Empirical results show considerable speedups for large HMMs in both inference and learning. The resulting class of approximations can be viewed as a parameterized mechanism for partially ignoring the sequential aspect of the data. $K = N-1$ gives us the regular HMM which models all possible transition probabilities, while the DMC HMM with K=1 models only one distinct transition probability per state, the largest one, and approximates the remaining sequential information in return for computational gains. Figures 2 and 3 show how, for a gaussian-based HMM, the DMC model can segue from a full HMM ($K = N-1$) to a gaussian mixture model ($K = 0$).

It seems that the accuracy of DMC models is determined by the extent to which the approximated probabilities conform to the DMC assumption, as well as the probability mass encapsulated by the K largest transition probabilities in each state on average. Domains where each underlying state transitions mostly to a small subset of states, but may in principle transit to any other state at times, are well-suited for DMC HMMs in terms of accuracy. Alternatively, in scenarios with large state-spaces where regular HMM algorithms would take too long, the speed of DMC HMM algorithms is the motivating factor for their usage rather than the exact suitability of the DMC assumption for the data. This is well justified since, as we demonstrated, explicitly modeling even a small fraction of transition probabilities in a large state-space HMM results in a more powerful and accurate model than a much smaller HMM, or an equally large state-space model that doesn't model transitions at all.

Aside from being a useful paradigm for efficiently working with large HMMs, the DMC HMM model is highly promising as a regularization mechanism to prevent overfitting when working with data that is well-modeled by large HMMs. A regular 100-state HMM has 10,000 transition probabilities that need to be learned; a DMC HMM of equal size with K=5 has only 500. Another potential application of DMC HMMs is fast sparse structure discovery; that is, a method for efficiently to discovering the transition structure in data that is well explained by a large HMM with a sparse transition matrix.

Acknowledgements

We would like to acknowledge the reviewers' comments which were extremely helpful, as well as discussions with Brigham Anderson. This work was partially funded by DARPA grant #NBCHD030010 and NSF ITR grant #CCF-0121671.

References

Bahl, L. R., Jelinek, F., & Mercer, R. L. (1983). A maximum likelihood approach to continouous speech recognition. *IEEE Trans Pattern Anal Machine Intell.* (pp. 179–190).

Baum, L. (1972). An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process. *Inequalities*, *3*, 1–8.

Brand, M., Oliver, N., & Pentland, A. (1997). Coupled hidden Markov models for complex action recognition. *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR).*

El-Difrawy, P. B. S., & Ehrlich, D. (2002). Hidden Markov Models for DNA Sequencing. *Proceedings of Workshop on Genomic Signal Processing and Statistics (GENSIPS).*

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).*

Murphy, K., & Paskin, M. (2002). Linear Time Inference in Hierarchical HMMs. *Advances in Neural Information Processing Systems (NIPS).*

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

Salakhutdinov, R., Roweis, S., & Ghahramani, Z. (2003). Optimization with EM and Expectation-Conjugate-Gradient. *Proceedings, Intl. Conf. on Machine Learning (ICML).*

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

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