DATA-DRIVEN RECOMPOSITION USING THE HIERARCHICAL DIRICHLET PROCESS HIDDEN MARKOV MODEL

Matthew D. Hoffman[†], *Perry R. Cook*[†‡], *David M. Blei*[†]
Princeton University
†Dept. of Computer Science
‡Dept. of Music
35 Olden St.
Princeton, NJ, USA 08540

ABSTRACT

Hidden Markov Models (HMMs) have been widely used in various audio analysis tasks such as speech recognition and genre classification. In this paper we show how HMMs can be used to *synthesize* new audio clips of unlimited length inspired by the temporal structure and perceptual content of a training recording or set of such recordings. We use Markov chain techniques similar to those that have long been used to generate symbolic data such as text and musical scores to instead generate sequences of continuous audio feature data that can then be transformed into audio using feature-based and concatenative synthesis techniques. Additionally, we explore the use of the Hierarchical Dirichlet Process HMM (HDP-HMM) for music, which sidesteps some difficulties with traditional HMMs, and extend the HDP-HMM to allow multiple song models to be trained simultaneously in a way that allows the blending of different models to produce output that is a hybrid of multiple input recordings.

1. INTRODUCTION

Generative probabilistic models, and in particular graphical models such as the hidden Markov model, have been successfully used for numerous audio analysis problems such as speech recognition, genre classification, source separation, automatic transcription, etc. These models attempt to explain input data by assuming that they were generated by an underlying process with some set of parameters that can be inferred from the data. This is in contrast to discriminative learning techniques such as boosting [13] or support vector machines [17], which try to minimize generalization error while making as few assumptions as possible. (For a good overview of graphical models, see [10].)

An interesting property of generative models is that, once trained, they can be used to synthesize new data. In most domains, there is little demand for this kind of artificial data, since the objective is to make predictions about data from the real world. But musical data may be an exception – many listeners care very little whether the musical "data" they listen to are "artificial" or "natural." Indeed, it could be argued that most music qualifies as artificial data manipulated for aesthetic purposes, in contrast to naturally occurring environmental sound.

In this paper we explore a novel application of the Hidden Markov Model (HMM) and its nonparametric cousin the HDP-HMM to data-driven music generation. By training an HDP-HMM on a sequence of feature vectors extracted from sliding windows of a recorded song, generating a new sequence of feature vectors from the trained model, and then using feature-based or concatenative synthesis to transform the feature vectors into audio, we can synthesize unlimited amounts of new audio based on a finite amount of training audio.

The rest of the paper is organized as follows. We begin by reviewing some background on Markov chains, HMMs, feature-based synthesis, and concatenative synthesis. We then discuss the more recently developed hierarchical Dirichlet process HMM, which is better suited than the traditional HMM to the long and complex input sequences produced by analyzing some musical audio. Finally, we present some experimental results that demonstrate the viability of our approach and discuss directions for future work.

2. THE MARKOV CHAIN

A first-order Markov chain is defined by a set of states, the transition probabilities between those states, and the probability of starting in each state. The transition matrix contains a vector of probabilities for each state describing how likely it is that the process will transition from that state to each other state. For example, in the model in figure 1, if at time t the model is in state A, at time $t+1$ there is a 70% chance of being in state A, a 10% chance of being in state B, and a 20% chance of being in state C. The probability of generating the sequence $ABAC$ would be $0.024 = 0.3 \cdot 0.1 \cdot 0.4 \cdot 0.2$, that is, the probability of starting in state A multiplied by the probabilities of the three transitions AB, BA, and AC.

The assumption that the future state of a process depends only on its single most recent state is often unreasonable. In such cases, it makes sense to take more than one previous state into account using a higher-order

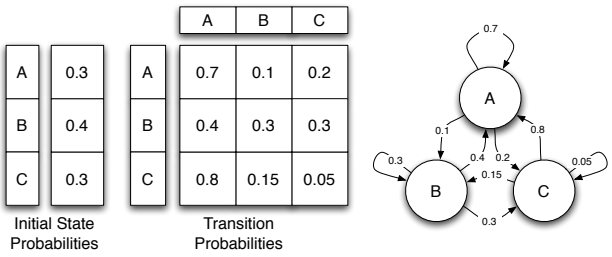

Figure 1. Initial state probabilities, transition matrix and graphical representation of a simple first-order three-state Markov chain.

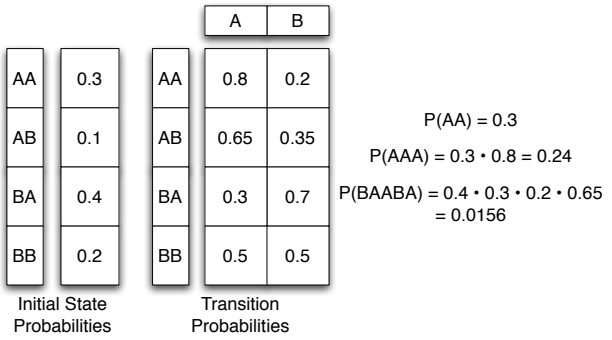

Figure 2. Initial state probabilities and transition matrix of a second-order two-state Markov chain, as well as some example sequences and their likelihoods.

Markov chain. Higher-order Markov chains can be defined by making transition probabilities depend on the last N states in addition to the most recent state. Figure 2 shows a simple second-order Markov chain's starting probabilities vector and transition matrix, as well as a few example state sequences and their likelihoods.

Training Markov chains (of any order) from data is fairly straightforward if state values are directly observable, as they are for symbolic data such as text and musical scores. To obtain the Markov chain under which the observed data are most likely to have occurred, one simply sets the transition probability vector from each state (or sequence of N states for an order-N model) to match the relative frequencies of each observed transition. For example, in the sequence $AABBABBA$ we find one AA transition and two AB transitions, so the likelihood of going from A to A would be $1/3$ and the likelihood of going from A to B would be $2/3$; there are two BA transitions and two BB transitions, so the likelihood of going from B to A would be $2/4$ and the likelihood of going from B to B would also be $2/4$.

Once trained, Markov chains can be used to generate new state sequences of arbitrary length that bear some resemblance to the data on which they were trained, a property that has been used to creative ends in interesting ways. The poet Jeff Harrison has generated poems using Markov chains trained on text documents (treating each unique word as a state); a few Bell Labs researchers trained a Markov chain on posts from the net.singles

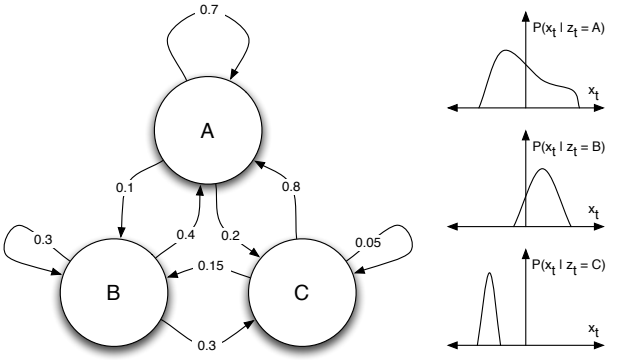

Figure 3. One perspective on the hidden Markov model. Left: Graphical representation of transition probabilities between the hidden states A, B, and C. Right: Probability density functions (PDFs) for the observed data given the underlying state of the model x_t is the observation at time t, and z_t is the underlying state label at time t.

Usenet group and posted the random posts it generated back to the group under the punning pseudonym "Mark V. Shaney." [1] Markov chains have also been used since at least the 1950's to produce musical scores [12, 1].

3. HIDDEN MARKOV MODELS

Although simple Markov chains lend themselves well to applications involving symbolic data, to model continuous data such as feature vector sequences describing audio we need to add another layer of complexity. The hidden Markov model assumes that there is still a set of states generating our data, and that the identity of each successive state still depends only on the state(s) before it, but now we cannot observe these states directly. Instead, each state is associated with an emission probability density function (PDF) that generates our observed data. Training HMMs is more complicated than training simple Markov chains, and is usually done either by attempting to maximize the likelihood of the training data (using the Baum-Welch expectation maximization algorithm [11]) or by placing priors on the HMM's parameters and trying to find parameters that maximize the posterior likelihood of the data [6].

3.1. Higher-order HMMs

Although it is possible to define higher-order HMMs as well as first-order HMMs, as the order grows even moderately high the transition matrix grows exponentially and becomes increasingly difficult to learn. A suitable workaround for our purposes is to build higher-order Markov models based on the state sequences obtained by training low-order HMMs. If we take these state sequences

[1] Those interested in playing with letter-based Markov chain text generation can open a text document in GNU Emacs and type "<META>-x dissociated-press."

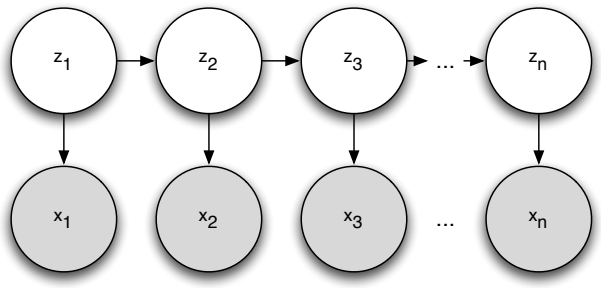

Figure 4. Graphical model representation of a first-order HMM. $z_{1...n}$ are the underlying states associated with the n observations $x_{1...n}$. The x_i's are shaded to show that they are observed directly. Each observation x_i is drawn from the distribution associated with state z_i, and therefore depends on z_i. Each state z_i depends on the previous state z_{i-1}.

as observed "true" data, then we can build Markov models of arbitrarily high order as described for symbolic data above. This is much the same approach as that taken by Casey [3] to build 2nd- to 8th-order lexemes for audio retrieval.

3.2. Generating new feature vector sequences

Once an HMM has been trained, it can be used to generate new sequences of observations in much the same way as a Markov chain with no hidden layer. First we generate a sequence of state assignments, then for each time step we draw an "observation" from the PDF associated with the state for that time step.

For audio, we might train an HMM on a sequence of vectors of 20 Mel-Frequency Cepstral Coefficients (MFCCs) extracted from a recorded song, assuming that each state generates a 20-dimensional MFCC vector from its own multivariate normal distribution. The training process estimates the initial state probability vector, transition probability matrix, and the means and covariance matrices associated with each state. These estimated parameters describe a model that could have generated the song on which we trained. Once the model is trained, we can then generate a new sequence of MFCC vectors as described above, and that sequence of MFCC vectors will describe another "song" that could have been generated by our model. This sequence of MFCC vectors is of little interest, however, unless we can turn it into something audible.

4. FEATURE-BASED AND CONCATENATIVE SYNTHESIS

The question of how to use feature vectors describing short windows of audio to drive sound synthesis has received some attention. We will consider several approaches to bridging the gap between HMM-generated state sequences and audio.

4.1. Feature-based synthesis

Some of the authors' previous research has focused on the problem of feature-based synthesis, which involves finding ways of synthesizing audio characterized by a desired vector of features. For some feature sets, efficient closed-form solutions exist to this problem. The MFCC extraction process, for example, can be reversed step by step to produce noisy audio with the appropriate broad spectral character.

It is not necessarily as easy to reverse arbitrary feature vectors consisting of multiple concatenated feature sets, however. In such circumstances we resort to combinatorial optimization algorithms such as simulated annealing or genetic algorithms to find synthesizer parameters that will produce audio described by a feature vector as close as possible to the desired feature vector. This optimization is performed using the FeatSynth framework [8, 9].

4.2. Concatenative synthesis

Another approach, often referred to as concatenative synthesis, attempts to find previously recorded grains of audio that closely match the desired feature values [14]. These short grains are retrieved from a large, efficiently indexed database. This approach is potentially more flexible than feature-based synthesis (since the database is not limited to synthetic sounds) and can have much less computational overhead than repeatedly synthesizing and testing new short audio segments. However, it requires that a sufficiently large database be compiled, analyzed, and indexed ahead of time, and leaves no recourse if no clips described by an adequately similar feature vector are in the database.

4.3. Cluster mosaicing

We also consider another concatenative approach, this one leveraging information that comes directly out of training the HMM. Assuming that we still have the audio from which the feature vector sequence(s) used to train the HMM were extracted, we can use the maximum-likelihood state sequence obtained using the Viterbi algorithm during training to associate each window from the training audio with a state of the HMM.

To generate audio for a new state sequence, for each time step t we can simply choose a window uniformly at random from those windows whose state labelling is the same as the current state z_t, add it to the end of the current audio stream (with appropriate crossfading), and move on to the next time step $t + 1$. This is in lieu of drawing from the emission density associated with z_t. This density should nonetheless be reasonably well approximated, since the empirical distribution of observations is what the emission density is supposed to be modelling in the first place.

Note that the database of windows available to this technique can be expanded beyond those provided by the audio used for training. The Viterbi algorithm can also be

used to provide maximum-likelihood labellings for new audio recordings, and each window of a new recording can be associated with its appropriate state. Doing so does, however, make it more likely that the empirical distribution of available windows for each state will become skewed. In this case, it may be best to actually draw a feature vector from the emission density and choose the cluster member with the smallest Euclidean distance, or to use some other heuristic.

Elements of this approach resemble the audio oracle, which deterministically creates a Markov chain in which each window of audio is a state [4]. Another approach suggested by Casey also used HMMs to cluster windows of audio, with a focus on indexing large databases for audio mosaicing [3].

5. THE HDP-HMM

When using traditional HMMs one must decide ahead of time how many states are necessary to capture the complexity of one's data. Choosing too few states results in an inadequately rich model, while choosing too many may result in overfitting or difficulties in training. Recent research in Dirichlet Process (DP) modelling has resulted in a nonparametric version of the HMM that can make a principled choice of how many states it needs based on the complexity of its training data. This model, the Hierarchical Dirichlet Process Hidden Markov Model (HDP-HMM) exploits the structure of the Hierarchical DP (HDP) developed by Teh et al. [16], and was extended by Fox et al. [5]. The HDP-HMM consists of a theoretically infinite number of states, only a few of which have a likelihood significantly greater than zero of being visited. A cursory overview of this model follows, but for details on training methods please refer to Fox et al.'s paper.

5.1. The basic model

In the standard generative HDP-HMM model, an HDP-HMM is created by first drawing a vector beta of infinite length (whose elements nonetheless sum to one) from Sethuraman's stick-breaking construction [15] with hyperparameter γ, denoted GEM(γ). This vector β describes the relative frequencies with which states are expected to be visited for the whole model. It defines a multinomial distribution over states, and for notational convenience we treat the vector and the distribution it defines interchangeably. Next, for each state j in $[1 \dots \infty]$ a vector π_j (also summing to one) is drawn from a DP with hyperparameter α and base distribution β. This vector π_j defines a multinomial distribution over transitions from state j, that is, it is the transition vector associated with state j. The higher the value of α, the less likely it becomes that π_j will deviate significantly from β. Next, parameters θ_k for the emission distributions $F(\theta)$ associated with each state k are drawn from their previously specified prior distributions H. This is summarized below:

$$\begin{aligned} \beta &\sim \text{GEM}(\gamma) \\ \pi_j &\sim \text{DP}(\alpha, \beta) \\ \theta_k &\sim H \end{aligned} \tag{1}$$

Then, it is assumed that such a model generated our training data $Y = y_{1\dots N}$, as well as a state sequence $Z = z_{1\dots N}$. Assume for simplicity that the starting state z_1 is drawn from β. Then z_{t+1} was drawn from π_{z_t} for $t = 1 \dots N-1$. Once each z_t was drawn, each observation y_t (our actual data) was drawn from $F(\theta_{z_t})$. To summarize:

$$\begin{aligned} z_t &\sim \pi_{z_{t-1}} \\ y_t &\sim F(\theta_{z_t}) \end{aligned} \tag{2}$$

Given Y, and having defined H and chosen values for α and γ, we can use the Gibbs sampling method described by Teh et al. to infer the state assignments Z. Given Z, we can then infer those parts of β, π, and θ that we care about – that is, those associated with states that are associated with observations. There are still theoretically an infinite number of states in the model, but those states that are not associated with observations can be dealt with in the aggregate. This algorithm requires that H be a conjugate prior distribution on $F(\theta)$, however, which can be a problematic restriction.

Another Gibbs sampling method can be used to train the HDP-HMM, which Fox et al. call the blocked-z Gibbs sampler. This method operates on a finite approximation to the full HDP-HMM that uses L states instead of an infinite number of states, and converges to the HDP-HMM as $L \to \infty$:

$$\begin{aligned} \beta &\sim \text{Dir}(\gamma/L, \dots, \gamma/L) \\ \pi_j &\sim \text{Dir}(\alpha\beta_1, \dots, \alpha\beta_L); \theta_k \sim H \\ z_t &\sim \pi_{z_{t-1}}; y_t \sim F(\theta_{z_t}) \end{aligned} \tag{3}$$

The blocked-z Gibbs sampler fully instantiates and samples β, π, z, and θ for this finite model, and can therefore exploit the relatively simple structure of the HMM to sample Z jointly using a variant of the forward-backward procedure [5], which may speed convergence. Another important advantage of this algorithm is that it does not require that H be a conjugate prior. Note that although the number of states L to use when training the model is specified, as long as L is sufficiently large not all L states will be used, and as L becomes very large the model converges to the infinite HDP-HMM described previously, in which only a finite number of states are actually used. In practice, setting L to be more than 2–4 times the number of states that the model winds up using does not seem to significantly alter results.

Although the hyperparameters α and γ are assumed to be given, we can place vague prior distributions (we used Beta(1, 0.005)) on each of them and let the model choose them as well [5]. If we allow the model to control α and

γ then the only parameters we need to specify are those associated with the priors over emission densities.

5.2. Priors on emission density parameters

We choose our prior densities to require only two parameters to be manually set – one controlling the expected size of each cluster and one controlling how much to allow cluster sizes to vary. If we specify a preference for smaller clusters, then the training algorithm will respond by allocating more clusters and therefore a richer model to capture the complexity of the data. The stronger a preference we specify for clusters of a particular size, the less variation there will be in cluster size.

Our emission distributions are multivariate normal. The conjugate prior distribution for the mean and covariance parameters of a multivariate normal distribution is the normal-inverse-Wishart [7]. Determining the posterior of this distribution based on observed samples is computationally straightforward. But the normal-inverse-Wishart distribution makes it difficult to specify different levels of prior certainty about the shape of the distribution (as captured by the correlations between dimensions) and its spread (the standard deviations of the dimensions). Ideally we would like to be able to express the sort of preferences with respect to spread described in the previous paragraph while letting the clusters take any shape the data suggest.

Mathematically, we accomplish this by factorizing the multivariate normal distribution's covariance matrix parameter Σ into $\Sigma = SRS$, where S is a square matrix with the standard deviation for each dimension on its diagonal and zeros elsewhere and R is a square symmetric positive definite correlation matrix where $R_{i,j}$ is the correlation between dimensions i and j. For each dimension we place an independent scaled inverse-$\chi^2(\nu, \sigma^2)$ distribution on the square of its standard deviation (i.e. its variance), where σ^2 is our desired or expected average cluster variance in that dimension and ν is a degrees of freedom parameter that controls how much weight to give σ^2. Depending on the data, it may make sense to specify different values of σ^2 for each dimension. One way to avoid having to do this manually is to look at the empirical standard deviations of the entire sequence (or the average standard deviations of smaller clusters) of feature vectors, scale them all by the same constant, and square them. This only requires that one parameter be specified a priori, while hopefully keeping the proportions of the emission distributions reasonable with respect to the data.

Unfortunately, few off-the-shelf distributions over symmetric positive definite matrices exist, so we use a variant on a parameter expansion technique described by Boscardin and Zhang [2]. We define an auxiliary diagonal standard deviation matrix Q and place a vague inverse-Wishart prior I-W$(I, d+1)$ on the covariance matrix QRQ, where d is the dimensionality of the feature vectors and I is the d-dimensional identity matrix. The marginal prior distribution of each correlation R_i, j under this distribution is uniform from -1 to 1 [7]. Although we no longer have a conjugate prior for the covariance matrix, we can use Gibbs sampling and the Metropolis-Hastings algorithm to sample from the posterior distributions of S, R, and Q, and therefore from the posterior distribution of $\Sigma = SRS$.

To simplify computation, we choose a conjugate multivariate normal prior on the mean of our emission distribution [7].

5.3. Adding another layer of hierarchy

We add another layer of hierarchy to the standard HDP-HMM in order to allow ourselves to train models for multiple songs simultaneously in a way that allows these models to share a common vocabulary of states. Sharing state vocabularies across models allows us to use the technique described in section 3.1 to build Markov chains that combine the transition characteristics of multiple songs.

The new generative model is:

$$\begin{aligned} \beta_0 &\sim \text{GEM}(\delta) \\ \beta_i &\sim \text{DP}(\gamma, \beta_0) \\ \pi_{i,j} &\sim \text{DP}(\alpha, \beta_i); z_{i,t} \sim \pi_{z_{t-1}} \\ \theta_k &\sim H; y_{i,t} \sim F(\theta_{z_{I,t}}) \end{aligned} \tag{4}$$

Where each song i has its own song-level state likelihood vector β_i, transition matrix π_i, state sequence Z_i, and observation sequence Y_i, and the emission density parameters θ are shared across all models. β_0 is an infinitely long vector that defines the global likelihood of being in a particular state across all songs, while each β_i defines the likelihood of being in a particular state for song i. The hyperparameter δ determines how much each β_i is likely to deviate from β_0, much like γ determines how much each $\pi_{i,j}$ is likely to deviate from each β_i. The same vague prior can be placed on δ as on α and γ, so it can also be inferred during training.

6. EXPERIMENTS

We ran experiments on the dance-pop song "Chewing Gum" by the Norwegian recording artist Annie. The song was selected because of the prominence of its strong, repeating beat. Sound examples and a link to a stream of the original song are at: http://www.cs.princeton.edu/∼mdhoffma/icmc2008. All algorithms were implemented in MATLAB and C++.

We began by breaking the song into non-overlapping 1024-sample windows and extracting each window's RMS power and first 20 Log-Frequency Cepstral Coefficients (LFCCs) [3], resulting in a 21-dimensional feature vector for each window, 10,086 feature vectors in all. We chose these features because they are simple to compute and, more importantly, simple and efficient to reverse. In the future, we plan on experimenting more with other features, particularly chroma vectors.

As a preprocessing stage, we ran Principal Component Analysis (PCA) on our 10,086 feature vectors and projected each onto the first 21 principal components. This ensures that no correlations exist between the 21 dimensions of the data we are trying to model, which can be

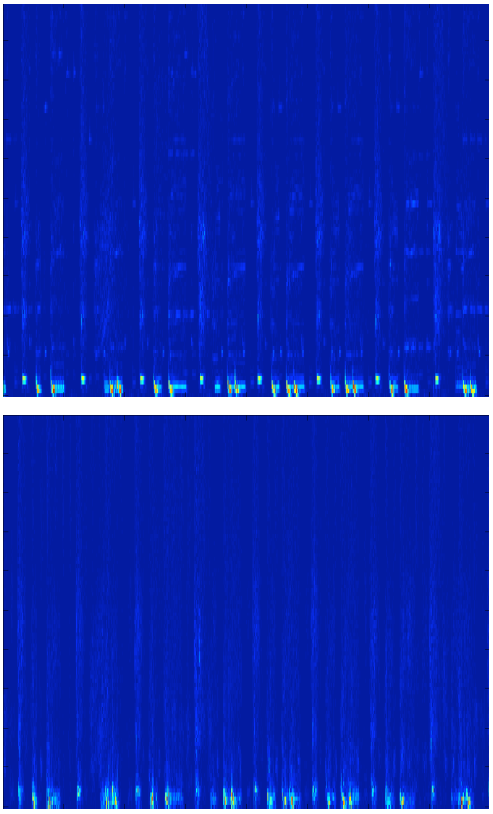

Figure 5. Top: spectrogram of a clip from "Chewing Gum." Bottom: spectrogram of resynthesized clip.

a desirable property. We transform our synthetic feature vectors back to their original basis before transforming them into audio. We then used Fox's blocked-z Gibbs sampler with $L = 175$ to train our model until the log-probability of the data under the model no longer increased. We chose the parameters for our priors as follows.

For the σ^2 parameters to the scaled-inverse-χ^2 priors on each variance, we chose the average variance in each respective dimension of 100 clusters of 70 points each. The clusters were selected at random by first choosing a random point and then finding the 70 closest points to that first point under the L^2 (Euclidean) norm. The goal was to push the model to expect to find clusters with a median size of about 70 points. We chose 50 for the degrees of freedom parameter to each scaled-inverse-χ^2 prior.

The mean and covariance parameters to the prior on the emission distribution's mean were chosen as the mean and the covariance matrix of the entire data set.

6.1. Results

Figure 5 shows a spectrogram of a roughly 10-second clip from "Chewing Gum." Below it is a spectrogram of a resynthesized version of the same clip generated by extracting the feature set described above and then converted the result back into audio. Notice that all fine detail in the spectrum has been washed out. This is because cepstral coefficients are designed to smooth away such fine detail.

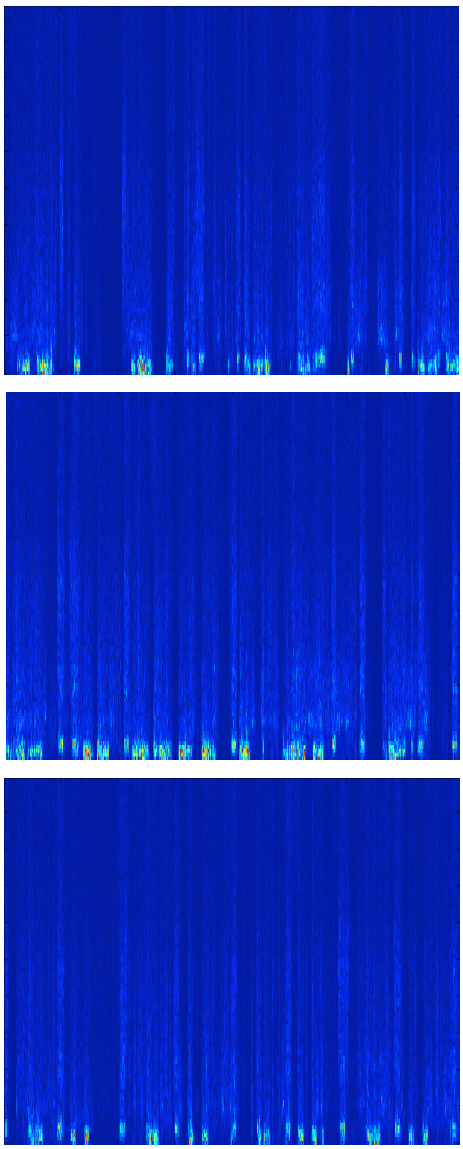

Figure 6. Spectrograms of audio produced by (from top to bottom) a 1st-order Markov chain, 4th-order Markov chain, and an 8th-order Markov chain.

Figure 6 compares spectrograms of 10-second audio clips generated by 1st, 4th, and 8th-order Markov chains created from our trained model. We manually reduced the variance of each emission density by 50% when generating new feature sequences to achieve a less noisy result. As information about more previous states is included, the model can produce audio with more structure, but also finds itself more constrained. If the order of the model N becomes too high, there will be almost no state sequences of length N that are not unique, and the model will be forced to reproduce the original state sequence.

In all, the model took about 15 iterations and 2300 seconds to converge on a MacBook Pro laptop with a 2.0 GHz Core Duo processor and 2 GB of RAM. Figure 7 plots the log-probability of the data under the model, the number

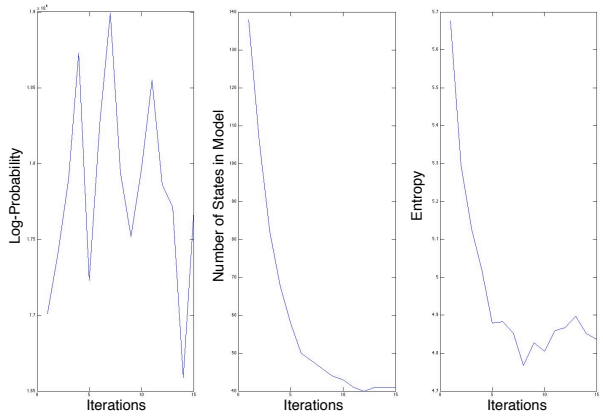

Figure 7. Log-probability of the data (left), number of states used (middle), and entropy of state histogram (right) at each iteration.

of states the model used, and the entropy of the histogram of states at each iteration (another measure of model complexity). Although the likelihood of the data converged fairly quickly, it took somewhat longer for the complexity of the model to stabilize.

7. DISCUSSION

Although our early results are promising, there remains much room for further exploration of generative modelling in conjunction with feature-driven synthesis methods. In the future we hope to extend the techniques discussed in this paper to include a more sophisticated treatment of rhythm, possibly using switching HMMs to model the transitions between measures and beats in a more hierarchical fashion. We also intend to apply these techniques to more pitch-savvy feature sets.

8. REFERENCES

[1] Ames, C. "The Markov process as a compositional model: a survey and tutorial," in Leonardo 22(2): pp. 175-188, 1989.

[2] Boscardin, W.J. and Zhang, X. "Modeling the covariance and correlation matrix of repeated measures," in Applied Bayesian Modeling and Causal Inference from Incomplete Data Perspectives, Gelman, A. and Meng, X. eds., Wiley, Hoboken, 2004.

[3] Casey, M. "Acoustic Lexemes for Organizing Internet Audio," in Contemporary Music Review, Vol. 24, No. 6, pp. 489-508, Dec. 2005.

[4] Dubnov, S., Assayag, G. and Cont, A. "Audio Oracle: a new algorithm for fast learning of audio structures," Proceedings of the International Computer Music Conference, Copenhagen, Denmark, 2007.

[5] Fox, E., Sudderth, E., Jordan, M., and Willsky, A. "Developing a tempered HDP-HMM for systems with state persistence," Technical report, MIT Laboratory for Information and Decision Systems, 2007.

[6] Gauvain, J. and Lee, C. "MAP estimation of continuous density HMM: theory and applications," Proceedings of the workshop on Speech and Natural Language, Harriman, New York, USA, 1992.

[7] Gelman, A., Carlin, J., Stern, H., and Rubin, D. Bayesian Data Analysis. CRC Press, Boca Raton, 2003.

[8] Hoffman, M., and Cook, P.R. "Feature-based synthesis: Mapping from Acoustic and Perceptual Features to Synthesis Parameters," Proceedings of the International Computer Music Conference, New Orleans, USA, 2006.

[9] Hoffman, M., and Cook, P.R. "The FeatSynth framework for feature-based synthesis: design and applications," Proceedings of the International Computer Music Conference, Copenhagen, Denmark, 2007.

[10] Jordan, M. "Graphical Models," in Statistical Science, Vol. 19, No. 1, pp. 140-155, 2004.

[11] Rabiner, L. "A tutorial on hidden Markov models and selected applications in speech recognition," Proceedings of the IEEE, vol. 77, no. 2, pp. 257-286, 1989.

[12] Roads, C. The Computer Music Tutorial. MIT Press, Cambridge, 1996.

[13] Schapire, R. "A brief introduction to boosting," Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, 1999.

[14] Schwarz, D. "An Overview of Current Research in Concatenative Sound Synthesis," Proceedings of the International Computer Music Conference, Barcelona, Spain, 2005.

[15] Sethuraman, J. "A constructive definition of Dirichlet priors," in Statistica Sinica, 4:639-650, 1994.

[16] Teh, Y., Jordan, M., Beal, M., and Blei, D. "Hierarchical Dirichlet processes," in Journal of the American Statistical Association, 101(476):1566-1581, 2007.

[17] Vapnik, V.N. The Nature of Statistical Learning Theory. Springer, New York, 2000.