Implementation of the Discrete Hidden Markov Model in Max / MSP Environment

Paul Kolesnik and **Marcelo M. Wanderley**
Sound Processing and Control Lab,
Centre for Interdisciplinary Research in Music Media and Technology,
McGill University
pkoles@music.mcgill.ca

Abstract

A set of discrete Hidden Markov Model (HMM) objects have been developed for Max/MSP software as part of the project that deals with the analysis of expressive movements of a conductor. The objects were tested with recognition of English alphabet symbols, and were applied toward analysis of isolated conducting gestures. Training and recognition procedures were applied toward both right hand beat- and amplitude- indicative gestures (beat and tempo indications), and left hand expressive gestures (articulation indications). Recognition of right-hand gestures was incorporated into a real-time gesture analysis and performance system that was implemented in Eyesweb and Max/MSP/Jitter environments.

Introduction

Analysis of expressive conducting gestures has been a topic of interest to researchers. Conducting can be viewed as a crucial link in the path that expressive information travels during a musical performance—where articulation encoded on a musical score is conveyed by the conductor's gestures, and then perceived and transformed by musical performers into expressive sound, which is perceived by the audience and interpreted as feedback by the conductor.

First successful attempts to analyze conducting gestures with the help of a computer were made as early as 1980 with *A Microcomputer-based Conducting System* (Buxton *et al.* 1980) that was based on previous research in music synthesis carried out by Moore, Matthews and collaborators in *Groove* project and *The Conductor Program* (Mathews 1976). Following these works, a number of conducting systems have been developed (see (Kolesnik & Wanderley 2004) for a list of related references). Those systems experimented with a number of approaches towards beat tracking and conducting gesture analysis.

Design of identification and recognition procedures for expressive gestures has been one of the main issues in the field of computer-based conducting gesture recognition. One of the main goals of the described project was to develop and test a set of recognition tools that would provide such functionality. Discrete Hidden Markov Model (HMM) techniques had been successfully used in analysis of right-hand beat-indicating conducting gestures (Usa & Mochida 1998), as well as in other related gesture analysis areas, such as sign language recognition (Vogler & Metaxas 1999) and facial expression recognition (Lien 1998).

Structure and Functionality of the HMM Package Objects

The HMM gesture recognition functionality was desired for a Gesture Analysis, Recognition and Performance system (Kolesnik & Wanderley 2004), which was implemented in Max/MSP sound processing environment. Preliminary tests carried out with already existing external HMM objects for Max/MSP showed that the objects were goal-specific and did not provide the functionality required for this project. Therefore, it was decided to implement a set of new external HMM objects based on previous implementations and theoretical materials (Rabiner & Huang 1986), (Rabiner 1989).

Main HMM Package Object - *dishmm*

The main *dishmm* object was written as an implementation of the three principal features of a discrete HMM model—training, optimal sequence of states and recognition.

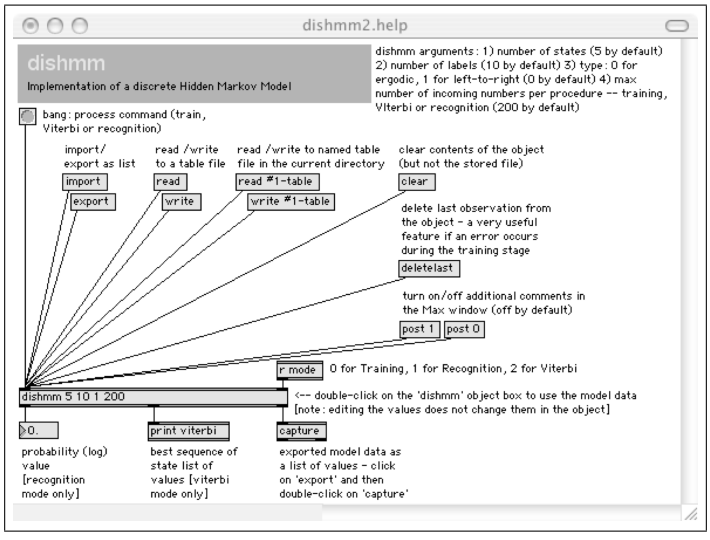

Figure 1: A 10-state 20-label left-to-right HMM object.

The number of states, labels and the type of the object (0 for ergodic, 1 for left-to-right) are specified as arguments to the object during the initialization stage in their respective order. The fourth argument accepted by the object box is the maximum number for the array that records the incoming labels the object can accept at a time. Those characteristics, which are directly responsible for the amount of memory allocation for the arrays used by the object, can be changed by reinitializing the object with new arguments, or by importing another recorded model into the current object using the *import* feature. If no arguments are supplied after the object name, it defaults to a 5-state 10-label ergodic model with 200 as the maximum size for the internal array of numbers recorded from the incoming stream.

The current mode of the object can be changed by sending an input to its right inlet—0 for training the model, 1 for recognition mode, and 2 for the Viterbi functionality. By default, the object is initialized in the training mode.

The object accepts an incoming stream of numbers in its left inlet and records them in an internal array in the order they are received. When a *bang* message is received in the left inlet, the labels stored in the internal array are passed to one of the object's algorithms—training, recognition or Viterbi, depending on what mode it is currently in, and the internal array is cleared in order to be ready to accept the subsequent stream of numbers. For the training algorithm, multiple scaled reestimation formulae are used to calculate the new model parameters, and the array of numbers is then stored in the object so that it can be used during the next training procedure together with all of the previously recorded and the new incoming observation arrays. For the recognition algorithm, probability is calculated using scaled forward and backward probability computations, and the result—which will always be in the range of $-\infty < logP(O|\lambda) \leq 0$ (where $P(O|\lambda)$ is the calculated probability of the model), since it is a logarithm of the actual probability—is passed on to the leftmost outlet of the object. For the Viterbi algorithm, the scaled Viterbi computations are used to generate the most probable sequence of states based on the incoming array, which is then passed as a list to the middle outlet of the object.

A *deletelast* message to the left inlet of the object deletes the most recent observation array from its memory, and retrains the model with all of the training observations that were recorded previously to the one that was deleted.

The object stores all of the model parameters which can be then viewed, imported or exported as a list, and written to or read from a text file. The information is stored as an array of values that follows the structure presented in Table 1.

Number of states, number of labels, type and number of trainings are represented by single integer values. State transition and state label output matrices are represented as arrays of floats, with each of the rows of the matrices placed in left-to-right order. There is no separation needed between the two matrices, since their dimensions are specified by the 'number of states' and 'number of labels' values—for example, a 5-state 10-label model will have a 5X5 state transition matrix represented by an array of 25 values, and a 5X10 output matrix represented by an array of 50 values. Training observations are stored after the two matrices, with each observation being in the format presented in Table 2.

Definition	*Length*
Number of states	1
Number of labels	1
Type	1
Number of trainings	1
Initial probability array	N
Transition matrix	N x N
State label output matrix	N x M
Training observation(s)	length of observation + 1 (per observation)

Table 1: Storing format of an HMM model (N is the number of states, and M is the number of output labels in the model).

Double-clicking on the object box opens a text editor which displays the current model information. A *read* message to the left inlet of the object opens a 'read file' dialogue so that a previously recorded model can be loaded into a current object. A *read* <*filename*> message results in reading the file from the same directory the current Max patch with the object is in. Similarly, *write* message opens a *write file* dialogue, and *write* <*filename*> writes the specified file in the same directory as the Max patch. The model information can be also imported and exported as a list within the patch—an *export* message sends out of the current model data through the rightmost outlet of the object in a list format, and an *import* <*list*> message (where <*list*>) is a list containing model information) loads the new model information into the object. Therefore, it is possible to exchange model information between several objects in the same patch by sending an *export* message to one object, appending the word *import* to the list that gets generated as the output, and sending it to another HMM object in the patch, which could be useful in some applications.

Number of labels in current observation
Array of observation output labels
-1(to indicate the end of the current observation)

Table 2: Storing format of an individual training observation.

The *post 1* and *post 0* messages turn on and off the additional information about the training, recognition and Viterbi computations to be printed in the Max window.

Appendix A provides an additional overview of the specifics of implementing the *dishmm* object in Max/MSP.

Other objects of the HMM Package

Two supporting external objects were written in addition to the main HMM object for the current system. The *orient2d* is responsible for calculation of positional orientation—it accepts a relative change in horizontal positional data in its left input and relative change in vertical positional data in the right output, calculates the positional orientation based

on those values, and outputs the result from the right output in degrees (in the range of 0-359) and out the left output in radians ($0 - 2\pi$).

The *code2d* object is responsible for the elementary vector quantization of the positional data stream. It divides the incoming positional data stream with a larger number of possible values (0-359 in this case) in a number of sectors, the number being determined by the desired number of labels for the HMM model object, and assigns that label to each incoming number within that sector. For example, a *code2d* object with 20 and 360 as its respective label size and maximum incoming data size, divides the range of 0-360 in 20 sectors, assigns the labels of 1 to 20 to each respective sector, and outputs a corresponding label for each incoming orientation value.

For future projects using the HMM object in Max/MSP (such as audio data recognition), or for advancements in the current project that would require a more complex quantization technique, other external objects will have to be written to produce the desired label stream that will serve as the input to HMM external objects.

Preliminary Tests with the HMM Objects

In order to verify the ability of the designed HMM package objects to correctly identify input gestures based on HMM training and recognition techniques, several preliminary tests were carried out. Left-to-right 5-state 10-label HMM models were used in all of the testing examples.

Symbol Recognition with a Mouse/Wacom Tablet

Recognition of English Alphabet symbols was the initial task for the system developed with the external HMM objects in order to test its performance. Absolute 2-D positional coordinates extracted from the movement of the mouse or a Wacom tablet were used to calculate the orientation values with a *orient2d* object. Resulting data stream was then passed to the *code2d* object that mapped the observation stream to a label data stream. Each of the HMM objects that were implemented in the system represented an isolated symbol to be recognized. At the learning stage, HMM objects were individually trained with 10 symbol examples. At the recognition stage, an observation stream representing a symbol was passed to all of the HMM objects, and the one producing the highest probability was considered as the recognized symbol. There were five observation examples of each symbol provided for recognition, and the system performed with a 92.5% recognition rate.

Symbol Recognition with USB Cameras and Eyesweb

The procedure used by the symbol recognition system was then replicated using a single webcam to capture a 2-D positional user input. Eyesweb video processing software (Camurri *et al.* 2000) was used to extract the positional information from the video input, and OSC network objects (Wright 1998) were used to transfer the positional information to the recognition patch in Max/MSP. Five English alphabet symbols (A,B,C,D,E) were used for training and recognition.

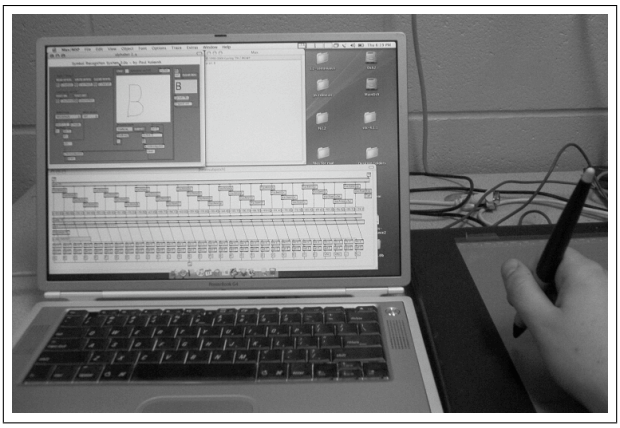

Figure 2: HMM symbol patch with a Wacom tablet input—recognition process.

As in the previous experiment, there were 10 training and 5 recognition sets per gesture.

The resulting recognition rates were lower than those obtained in the previous experiment. In particular, the capital symbol ‘D’ was repeatedly mistaken for a ‘B’, whereas all of the other symbols (that did not share positional similarities, as in the case of those two symbols) were correctly identified. This can be explained by the fact that Eyesweb has a faster recognition rate than the one used by the mouse tracking object, and the visual gesture symbolizing the symbol was performed during a longer period of time than writing it in with a mouse. Therefore, the left-to-right object did not contain enough states to represent all of the positional transitions, and considered the symbol ‘D’ as the upper part of the symbol ’B‘,whereas it did not contain enough available states to represent the lower part. On the basis of this observation, it was decided to use 10-state models for all of the HMM models during the actual conducting gesture recognition experiments.

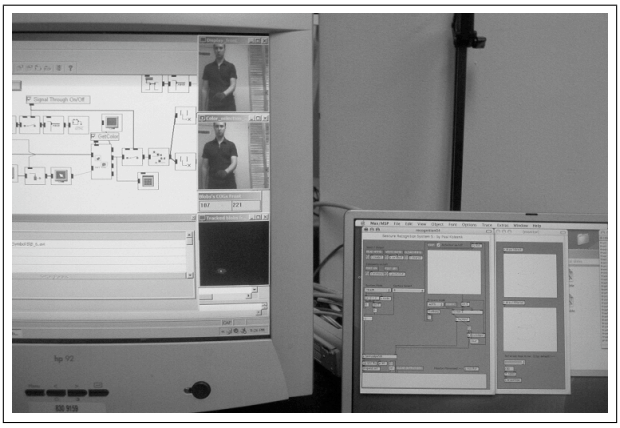

Figure 3: HMM symbol patch with Eyesweb input—training process.

Testing the HMM Objects with Conducting Gesture Recognition

A number of tests with conducting gestures were carried out using the HMM objects. All of the conducting gestures were performed by a doctoral conducting student at the Music Faculty of McGill University. The conducting gesture recordings were done using the Eyesweb software with two USB cameras that were placed in front and profile view of the conductor. The recorded session video files were later edited in order to prepare them for use with the recognition patches in Max/MSP (editing involved deleting unnecessary beginnings/endings of the files, and splitting larger session files into training and recognition parts). The HMM objects that were used in Max for all of the gesture recognition experiments were based on a 10-state 20-label left-to-right model.

Left hand Expressive Gestures

Five left-hand isolated expressive gestures were selected to be recognized—*crescendo*-cutoff, *diminuendo*-cutoff, *fermata*-click gesture, *accent* indication and expansion gesture. The set of gestures was intentionally chosen to contain both simple (accent, expansion) and complex(crescendo+cutoff, diminuendo+cutoff and fermata+click) gestures in order to test the system's ability to cope with both kinds of gestures simultaneously.

Figure 4: Front and profile camera view of the user training the recognition system with a left-hand *crescendo*-cutoff expressive gesture.

For each of the five gestures, 20 training sets and 10 recognition sets were recorded as two synchronized movie files for front and profile views, and then split into 30 individual file pairs using video editing software. In the recognition component of the system, five HMM object pairs were assigned to correspond to the gestures. Each HMM object pair was then individually trained with the 20 training video segments. Upon completion of the training process, 50 examples (10 examples per gesture) were presented for recognition to the entire set of the HMM objects. The recognition scores of the pairs of HMMs were combined and compared to find the maximum logarithm value, which indicated the gesture that was considered as the most likely to correspond to the incoming positional data stream.

The gestures were recognized with a 98 $\%$ degree of accuracy. In fact, those gestures presented a real challenge to the HMM models, since they shared many of the positional characteristics. For example, the complex *fermata*-click gesture (which is described in detail in (Rudolph 1994)) is a combination of a *fermata* indication, followed a short 'click' or 'breath', followed by an entrance indication. The positional information received from the middle part of the *fermata*-click gesture is very similar to the simple accent gesture. Nonetheless, the HMM objects were able to distinguish the differences between the gestures.

Expressive Styles of Right Hand Beat Indicating Gestures

For right-hand beat indicating gestures, three sets of beat patterns were chosen—the first set containing a four-beat expressive legato and four-beat light staccato patterns, the second with a three-beat expressive legato and three-beat light staccato, and the third set with a two-beat legato, two-beat marcato and two-beat staccato patterns. A separate HMM object pair was used to represent each beat gesture of the described patterns. For each beat pattern, 20 measures of continuous conducting was recorded for gesture training purposes and 10 measure of contunuous conducting for gesture recognition. In this case, instead of manually splitting the video files into individual segments for each beat of each pattern, the temporal segmentation process was performed by the beat identification component of the system. A loop was performed during the execution where every time a beat was detected, the incoming positional data stream was rerouted to the HMM pair representing the next beat, and when the last beat of the beat pattern was reached, the first beat HMM pair was chosen for the next training stage.

After all of the three sets of beat patterns (representing 7 different beat patterns and 20 different beat model pairs) were trained, 10 measures for each beat pattern were presented to the system for gesture recognition. The recognition for each beat pattern was done twice—at first, by comparing the scores of the HMM pairs corresponding to different beats of the current beat pattern only, and then by comparing the scores of the models representing the beats of the entire set.

The system provided a robust recognition of right hand beat-indicating gestures, both in terms of correctly identifying the individual beat information within a single beat pattern, and distinguishing between different styles of conducting using the same meter value. The overall recognition rate was equal to 99.5% for the gestures that were performed consecutively by the same conductor.

Embedded Right hand Expressive Gestures

Since the right hand is known to be mainly responsible for time-beating indications throughout the performance, the right hand expressive gestures have to be incorporated into the beating gestures—unlike the left hand that has more freedom of movement and is not constrained with the baton (if it is being used). Therefore, in order to extract right hand expressive gestures, either transitions from one type of beat to another or different forms of the same basic type of beat should be analyzed.

The gestures were studied on examples of gradual *crescendo* and *diminuendo* indicated through gradual in-

crease/decrease in the amplitude of individual beat indicating gestures. Three different cases were studied—a gradual crescendo (one measure span) on a four-beat legato, a gradual diminuendo (one measure span) on a four-beat legato, and no dynamics on a four-beat legato. In this case, each of the HMM model pairs represented the entire measure and not an individual beat in the measure as in the previous experiment, so that it would be possible to track changes in beat amplitude over the four beats. Therefore, this system variation contained three HMM pairs, one per beat pattern.

Since in this case only three long patterns produced by the same conductor were analyzed, and their positional relative paths were clearly different by comparison to each other, the system was able to distinguish the gesture transition patterns with a perfect recognition rate.

Combined Gesture Analysis, Recognition and Performance System

Recognition of right-hand beat-indicating gestures was used as one of the components of the Gesture Analysis, Recognition and Performance system (Kolesnik 2004), (Kolesnik & Wanderley 2004) during a realtime conducting performance. The system included realtime gesture analysis, recognition and performance of a prerecorded audio score using phase vocoder audio stretching techniques. Detection of conducting beats and calculation of audio stretching amount required to synchronize the conductor and the audio playback was done on the basis of previous works in the related field (Borchers, Samminger, & Muhlhauser 2002), (Murphy, Andersen, & Jensen 2003).

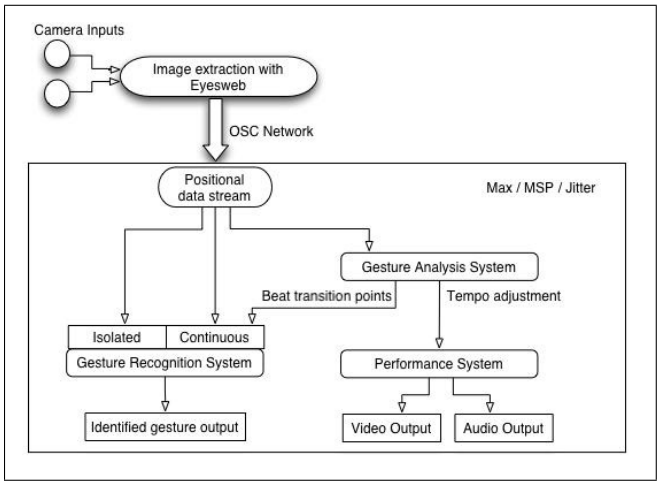

Figure 5: Schematic representation of the Gesture Analysis, Recognition and Performance system.

Whereas the input processing part of the patch was often unstable due to software/hardware limitations of processing several simultaneous video streams received from USB cameras, the HMMs were able to recognize the incoming gesture data with a high degree of accuracy during the cases when an acceptable number of positional frames was received—the system performed with a 94.6% recognition rate over the 73 measures, with comparison being done between the four possible gestures in the 4-beat legato pattern.

Analysis of Results

Through a set of isolated gesture recognition experiments, the preliminary results showed that the Hidden Markov Model package provides an efficient tool for recognition and analysis of both isolated indicative and expressive conducting gestures in Max/MSP environment. Very high or perfect recognition rates resulted from the fact that the compared gestures were produced by the same conductor in a uniform environment. Further research is needed on how the system would react to gestures produced by different conductors.

Conclusion and Future Work

The main achievement of the project was the development of a set of objects that were used for recognition of isolated conducting gestures in Max/MSP environment. This brings an improvement over existing systems, since whereas right hand movements had been analyzed with HMM (Usa & Mochida 1998) and Artificial Neural Net (Ilmonen & Takala 1999) (Garnett *et al.* 2001) techniques in the past, there has been no previous research involving high-level recognition and classification techniques applied to left hand expressive gestures.

The current system is able to provide recognition of isolated gestures—however, one of the future goals of the project is to design a gesture recognition process that can be implemented in a continuous conducting movement environment in combination with the developed gesture analysis and performance system.

Whereas the described experiments used low-cost USB cameras for gesure tracking, further research on conducting gestures that is currently carried out is using Vicon Motion Capture and Polhemus Motion Capture systems now available at the Sound Processing and Control Lab at McGill University.

The objects are available for free distribution and further research on positional gestures, audio content recognition or any other data stream recognition in Max/MSP environment.

References

Borchers, J.; Samminger, W.; and Muhlhauser, M. 2002. Engineering a realistic real-time conducting system for the audio/video rendering of a real orchestra. In *Proceedings of the 4th International Symposium on Multimedia Software Engineering*, 352–362.

Buxton, W.; Reeves, W.; Fedorkov, G.; Smith, K. C.; and Baecker, R. 1980. A microprocessor-based conducting system. *Computer Music Journal* 4(1):8–21.

Camurri, A.; Coletta, P.; Peri, M.; Ricchetti, M.; Ricci, A.; Trocca, R.; and Volpe, G. 2000. A real-time platform for interactive performance. In *Proceedings of the International Computer Music Conference*. International Computer Music Association.

Garnett, G. E.; Jonnalagadda, M.; Elezovic, I.; Johnson, T.; and Small, K. 2001. Technological advances for conducting a virtual ensemble. In *Proceedings of the International Computer Music Conference*, 167–169. International Computer Music Association.

Ilmonen, T., and Takala, T. 1999. Conductor following with Artificial Neural Networks. In *Proceedings of the International Computer Music Conference*, 367–370. International Computer Music Association.

Kolesnik, P., and Wanderley, M. 2004. Recognition, analysis and performance with expressive conducting gestures. In *Proceedings of the International Computer Music Conference*, (in print). International Computer Music Association.

Kolesnik, P. 2004. Conducting gesture recognition, analysis and performance system. Master's thesis, McGill University.

Lien, J. J. 1998. *Automatic Recognition of Facial Expressions Using Hidden Markov Models and Estimation of Expression Intensity*. Ph.D. Dissertation, The Robotics Institute, Carnegie Mellon University.

Mathews, M. V. 1976. The Conductor program. In *Proceedings of the International Computer Music Conference*.

Murphy, D.; Andersen, T. H.; and Jensen, K. 2003. Conducting audio files via Computer Vision. In *Proceedings of the 2003 International Gesture Workshop*.

Rabiner, L. R., and Huang, B. H. 1986. An introduction to Hidden Markov Models. *IEEE Acoustics, Speech and Signal Processing Magazine* 3(1):4–16.

Rabiner, L. R. 1989. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of IEEE* 77(2):257–285.

Rudolph, M. 1994. *The Grammar of Conducting: A comprehensive guide to baton technique and interpretation*. Toronto: Maxwell Macmillan Canada.

Usa, S., and Mochida, Y. 1998. A multi-modal conducting simulator. In *Proceedings of the International Computer Music Conference*, 25–32. International Computer Music Association.

Vogler, C., and Metaxas, D. 1999. Parallel Hidden Markov Models for american sign language recognition. In *In Proceedings of the International Conference on Computer Vision*, 116–122.

Wright. 1998. Implementation and performance issues with OpenSound Control. In *Proceedings of the International Computer Music Conference*, 224–227. International Computer Music Association.

Appendix A. Specifics of the *dishmm* Object Implementation

This section provides a brief overview of the *dishmm* object's implementation in Max/MSP. A complete set of formulae used by the object procedures is available (Kolesnik 2004). The formulae, together with additional description of the object and sample Max/MSP patches, are also available at *http://www.music.mcgill.ca/~pkoles/download.html*.

Initialization

During the initialization process, the object allocates memory for a number of internal arrays that are used for storing of intermediate and final values during the three main procedures. In order to avoid a situation where the model is given an observation label during the recognition process that it did not get during the training process, the initial model setting, which has random (for ergodic models) or equal (for left-to-right models) distribution of parameter weights, is considered as one of the settings derived from model observations. As the model gets more training, the initial observation gets less and less weight assigned to it.

Training Process

The training procedure retrains the initial probability, transition and output array values, according to the traditional HMM reestimation procedure (Rabiner & Huang 1986). However, to avoid computation range errors during the calculations, scaled forward and backward probabilities are used in the calculations. The scaled forward and backward probabilities are calculated on the basis of the original forward and backward probability arrays in combination with a scaling coefficient c_t.

By default, 50 reestimation procedures are done for each for each of the three model parameter arrays (i.e. initial probability array, transition matrix and label output matrix), using temporary storage arrays to store the intermediate reestimation parameter values. After the completion of the reestimation process, the parameters are recorded in the main model arrays.

Viterbi Process

Similarly to the reestimation process, the Viterbi process follows the traditional recursive routine for finding the optimal sequence of states with state sequence backtracking (Rabiner & Huang 1986), but scales the array values by taking logarithms of the original $\tilde{\delta}_t(i)$ values. The optimal state probability values are stored by the object in a $\tilde{\delta}$ array.

Recognition process

Probability of the model can be found as a product of individual probabilities calculated for all of the observation sequences. The probability of each individual sequence $P(O|\lambda)$ is calculated using scaling coefficients c_t. However, in practice there will be a possibilty that the resulting probability value will also be smaller than computational precision range. A solution to this is to calculate $\log P(O|\lambda)$ instead, which produces a negative value in an acceptable range.

The calculation of observation/model probability process uses the same forward and backward probability calculation that are used in the training process, but applies them directly to the model parameter arrays without using temporary array space, and only goes through the process once.

Additional Functions

Normalize Normalize function is used to normalize the arrays following the completion of the training process in order to avoid problems caused by precision errors during the calculation of intermediate probability values. It is also used in conjunction with the randomize functions.

Randomize Functions A set of randomize functions is implemented in order to provide initial values to the model's initial array, transition matrix and output label matrix values, in case the dishmm object is initialized as an ergodic HMM model. Random values in the range of 0 to 100 are given to the array and matrices, and they are then normalized to correspond to general HMM requirements. The random values are not used if the object is initialized as a left-to-right HMM model, since equal energy distribution in the arrays and matrices are an acceptable option in this case.

Observations The HMM object records all of the observation data as a part of its internal structure. When a *deletelast* message is received by the object, it drops the data of the last observation and retrains itself with all of the previous observations.

Input/Output The object allows to import and export its information internally in Max/MSP as a list or externally by writing to a text file. Double-clicking on the object box allows to view its current contents in a separate text window.