Hierarchical Sequential Memory for Music: A Cognitive Model

James B. Maxwell
Simon Fraser University
jbmaxwel@sfu.ca

Philippe Pasquier
Simon Fraser University
pasquier@sfu.ca

Arne Eigenfeldt
Simon Fraser University
arne_e@sfu.ca

ABSTRACT

We propose a new machine-learning framework called the Hierarchical Sequential Memory for Music, or HSMM. The HSMM is an adaptation of the Hierarchical Temporal Memory (HTM) framework, designed to make it better suited to musical applications. The HSMM is an online learner, capable of recognition, generation, continuation, and completion of musical structures.

1. INTRODUCTION

In our previous work on the MusicDB [10] we outlined a system inspired by David Cope's notion of "music recombinance" [1]. The design used Cope's "SPEAC" system of structural analysis [1] to build hierarchies of musical objects. It was similar to existing music representation models [7, 9, 13], in that it emphasized the construction of hierarchies in which the objects at each consecutively higher level demonstrated increasing "temporal invariance" [5]—i.e., an "S" phrase in SPEAC analysis, and a "head" in the Generative Theory of Tonal Music [9], both use singular names at higher levels to represent *sequences* of musical events at lower levels.

Other approaches to learning musical structure include neural network models [8], recurrent neural network models (RNNs) [11], RNNs with Long Short-Term Memory [3], Markov-based models [12, 14], and statistical models [2]. Many of these approaches have achieved high degrees of success, particularly in modeling melodic and/or homophonic music. With the HSMM we hope to extend such approaches by enabling a single system to represent melody, harmony, homophony, and various contrapuntal formations, with little or no explicit *a priori* modeling of musical "rules"—the HSMM will learn only by observing musical input. Further, because the HSMM is a cognitive model, it can be used to exploit musical knowledge, in real time, in a variety of interesting and interactive ways.

2. BACKGROUND: THE HTM FRAMEWORK

In his book "On Intelligence", Jeff Hawkins proposes a "top-down" model of the human neocortex, called the "Memory Prediction Framework" (MPF) [6]. The model is founded on the notion that intelligence arises through the interaction of perceptions and predictions; the perception of sensory phenomena leads to the formation of predictions, which in turn guide action. When predictions fail to match learned expectations, new predictions are formed, resulting in revised action. The MPF, as realized computationally in the HTM [4, 5], operates under the assumption of two fundamental ideas: 1) that memories are hierarchically structured, and 2) that higher levels of this structure show increasing temporal invariance.

The HTM is a type of Bayesian network, and is best described as a memory system that can be used to discover or infer "causes" in the world, to make predictions, and to direct action. Each node has two main processing modules, a Spatial Pooler (SP) for storing unique "spatial patterns" (discrete data representations expressed as single vectors) and a Temporal Pooler (TP) for storing temporal groupings of such patterns.

The processing in an HTM occurs in two phases: a "bottom-up" classification phase, and a "top-down" recognition, prediction, and/or generation phase. Learning is a bottom-up process, involving the storage of discrete vector representations in the SP, and the clustering of such vectors into "temporal groups" [4], or variable-order Markov chains, in the TP. A node's learned Markov chains thus represent temporal structure in the training data. As information flows up the hierarchy, beliefs about the identity of the discrete input representations are formed in each node's SP, and beliefs about the membership of those representations in each of the stored Markov chains are formed in the TP. Since the model is hierarchical, higher-level nodes store invariant representations of lower-level states, leading to the formation of high-level spatio-temporal abstractions, or "concepts."

A simplified representation of HTM processing is given in Figure 1. Here we see a 2-level hierarchy with two

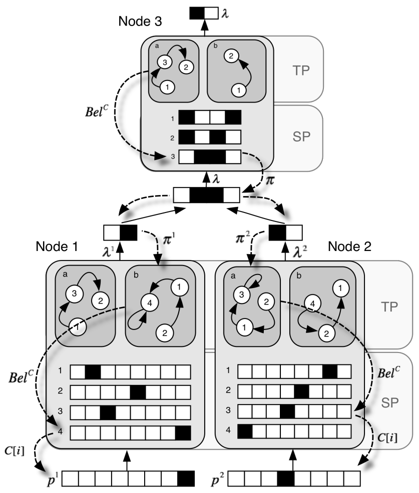

Figure 1. Simplified HTM processing.

nodes at L1 and one node at L2. This HTM has already received some training, so that each L1 node has stored four spatial patterns and two Markov chains, while the L2 node has stored three spatial patterns and two Markov chains. There are two input patterns, p^1 and p^2. It can be seen that p^1 corresponds to pattern 4 of Node 1, and that pattern 4 of Node 1 is a member of Markov chain *b*. When presented with p^1, the node identifies pattern 4 as the stored pattern most similar to p^1, calculates the membership of pattern 4 in each of the stored Markov chains, and outputs the vector [0, 1], indicating the absence of belief that p^1 is a member of Markov chain *a*, and the certainty that p^1 is a member of Markov chain *b*.

It can also be seen from Figure 1 that the outputs of the children in hierarchy are concatenated to form the inputs to the parent. The SP of node 3 thus treats the concatenated outputs of nodes 1 and 2 as a discrete representation of their temporal state at a given moment—i.e., time is 'frozen' by the parent node's SP. Node 3 then handles its internal processing in essentially the same manner as nodes 1 and 2.

The dotted lines indicate the top-down processes by which discrete state representations can be extracted or inferred from the stored Markov chains, and passed down the network. Top-down processing can be used to support the recognition of inputs patterns, to make predictions, or to generate output.

3. MOTIVATIONS BEHIND THE HSMM

Our interest in the HTM as a model for representing musical knowledge derives from its potential to build spatio-temporal hierarchies. The current HTM implementation from Numenta Inc., however, is focused primarily on visual pattern recognition [4, 5], and is currently incapable of learning the sort of high-level temporal structure found in music. This structure depends not only on the temporal proximity of input patterns, but also on the specific *sequential order* in which those patterns arrive. The HSMM treats sequential order explicitly, and can thus build detailed temporal hierarchies.

Another motivation behind the HSMM lies in the fact that the HTM is strictly an offline learner. For compositional applications, we are interested in a system that can acquire new knowledge during interaction, and exploit that knowledge in the compositional process. We have thus designed the HSMM with four primary functions in mind:

1) **Recognition:** The system should have a representation of the hierarchical structure of the music at any given time in a performance.
2) **Generation:** The system should be capable of generating stylistically integrated musical output.
3) **Continuation:** If a performance is stopped at a given point, the system should continue in a stylistically appropriate manner.
4) **Pattern Completion:** Given a degraded, or partial input representation, the system should provide a plausible completion of that input (i.e., by adding a missing note to a chord).

4. MUSIC REPRESENTATION

For the current study, we are working with standard MIDI files from which note data is extracted and formatted into three 10-member vectors: one for pitch data, one for rhythmic data, and one for velocity data. The incoming music is first pooled into structures similar to Cope's "Groups" [1]—vertical 'slices' of music containing the total set of unique pitches at a given moment. A new Group is created every time the harmonic structure changes, as shown in Figure 2. The Groups are preprocessed using a simple voice-separation algorithm, which divides polyphonic material across the 10 available voices in the vector representation. Group pitches are first sorted in ascending order, after which the voice separation routine follows one simple rule: tied (sustained) notes must not switch voices.

Pitch material is represented using inter-pitch ratio [16], calculated by converting the MIDI notes to hertz,

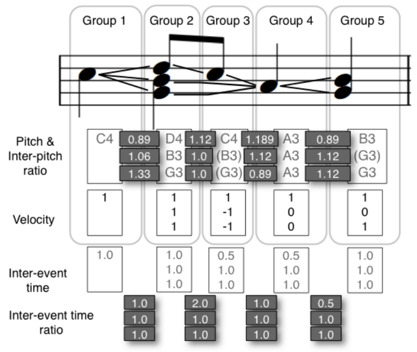

Figure 2. Music representation for the HSMM.

and dividing the pitch at time *t*-1 by the pitch at time *t*. In order to avoid misleading values, as a result of calculating the ratio between a rest (pitch value 0.0) and a pitch, rests are omitted from the pitch representation, and the velocity representation is used to indicate when notes are active or inactive (see Figure 2).

It will be noted that velocity is not given using conventional MIDI values, but is rather used as a flag to indicate the state of a given voice in the Group. Positive values indicate onsets, negative values indicate sustained notes, and zeros indicate offsets. We have simplified the non-zero values to 1 and -1 in order to avoid attributing too much weight to note velocity in the training and inference process.

The rhythmic values used represent the times at which each voice undergoes a transition either from one pitch to another, or from a note-on to a note-off. We use the inter-event time between such changes, and calculate the ratio between consecutive inter-event times, for each voice *n*, according to the following:

$$interEventRatio^{t}[n] = \frac{interEventTime^{t-1}[n]}{interEventTime^{t}[n]} \qquad (1)$$

The final representation for the HSMM will thus consist of one 10-member inter-pitch ratio vector, one 10-member inter-event time ratio vector, and one 10-member velocity flag vector.

5. HSMM LEARNING AND INFERENCE

Figure 3 shows a four-level HSMM hierarchy with inputs for pitch, rhythm, and velocity information, an "association" node (L2) for making correlations between the L1 outputs, and two upper-level nodes for learning higher-ordered temporal structure. The association node at L2 provides the necessary connectivity between pitch, rhythm, and velocity elements required for the identification of musical "motives." The upper-level nodes at L3 and L4 are used to learn high-order musical structure from the motives learned at L2.

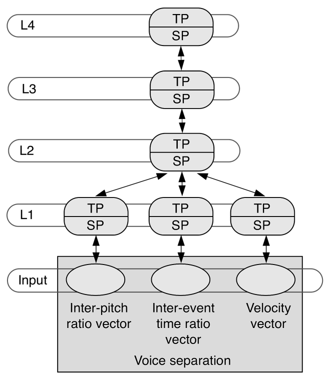

Figure 3. A four-level HSMM hierarchy.

5.1 Online Learning in the HSMM

Whereas the TP in the HTM builds Markov chains during its training phase, in the HSMM we focus simply on constructing discrete *sequences* from the series of patterns input to the node. As in the HTM, the patterns stored in the SP will be referred to as "coincidences." The sequences stored by the TP will be referred to simply as "sequences."

5.1.1 Learning and Inference in the Spatial Pooler

The objective of SP learning is to store unique input patterns as coincidences, while the objective of SP inference is to classify input patterns according to the stored coincidences. The algorithm used is given in Figure 4. As in the HTM, when a new input is received the SP checks the input against all stored coincidences, *C*. The result is an SP output vector y^t, calculated according to:

$$y^t[i] = e^{-d(p,C[i])^2/\sigma^2},\ for\ i=0\ to\ |C|-1 \qquad (2)$$

where $d(p,C[i])$ is the Euclidean distance from input *p* to coincidence *C*[*i*], and σ is a constant of the SP. The constant σ is used to account for noise in the input, and is useful for handling rhythm vectors, where frequent fluctuations of timing accuracy are expected. The output y^t is a belief distribution over coincidences, in which a higher value indicates greater similarity between input pattern *p* and stored coincidence *C*[*i*], and thus greater evidence that *p* should be classified as *C*[*i*]. If the similarity of *p* to all stored coincidences is less than the minumum allowed similarity, *simThreshold*, *p* is added as a new coincidence.

In the event that a new coincidence is added, the algorithm uses the value of *maxSimilarity*—i.e., the belief in the coincidence most similar to input *p*—as the initial belief value when adding the new coincidence. It then normalizes y^t in order to scale the new belief according to the belief over all stored coincidences. In order to decide whether a new coincidence is required at higher levels, we start by first determining whether the input pattern λ (see Figure 1) should be stored as a new coincidence. This is simply a matter of checking the length of the λ vector at time *t* against the length at time *t*-1. If the length has increased, we know that at least one of the children has learned a new representation in the current time step, and that a new coincidence must be added in order to account for the additional information. For each new coincidence stored by the SP, a histogram called the *counts* vector is updated. In the HTM, the update is an integer incrementation—a count of how many times the coincidence has been seen during training. However, because the HSMM is an online learner, an integer incrementation is not appropriate, as it would lead to counts of vanishingly small proportions being assigned to new coincidences if the system were left running for long periods of time. Thus, in the HSMM, the *counts* vector is updated according to the following:

$$inc = |C| \times 0.01 \qquad (3)$$

$$counts^t[topCoinc^t] = counts^{t-1}[topCoinc^t] + inc \qquad (4)$$

$$counts^t[i] = \frac{counts^t[i]}{1+inc},\ for\ i=0\ to\ |counts|-1 \qquad (5)$$

where *C* is the number of learned coincidences, *inc* is the incrementation value, and $topCoinc^t$ is the coincidence that rated as having the *maxSimilarity* (Figure 4) to the input. Because *counts* is regularly normalized, it represents a time-limited histogram in the HSMM.

SP inference above L1 is calculated as in the HTM, but we outline it here for the sake of clarity. At higher levels we want to calculate the *probability* that the new input λ should be classified as one of the stored coincidences. When the node has more than one child, we consider each child's contribution to the overall probability separately:

$$\begin{gathered} C = C^1 \cup \ldots \cup C^M \\ \lambda = \lambda^1 \cup \ldots \cup \lambda^M \\ y^t[i] = \prod_{j=1}^{M} \max_{k} (C^j[k,i]) \times \lambda^j[k],\ for\ i=0\ to\ |C|-1 \end{gathered} \qquad (6)$$

where *M* is the number of child nodes, C^j is the portion of coincidence vector *k* attributed[1] to child *j*, and λ^j is the portion of *λ* attributed to child *j*. Figure 5 shows an example calculation for a hypothetical SP with two stored coincidences.

p	The current input pattern
C	The table of stored coincidences
maxSimilarity	The maximum similarity value found
simThreshold	The minimum degree of similarity between input *p* and coincidence *C*[*i*] required for *p* to be classified as *C*[*i*]
unmatchedCoinc	A count of the number of times input *p* was found to be insufficiently similar to all coincidences in *C*

```
Set maxSimilarity to 0
For each stored coincidence C[i] in C
  Calculate $y^t[i]$, given input p, according to Equation 2
  If $y^t[i]$ > maxSimilarity
    Set maxSimilarity to $y^t[i]$
  If $y^t[i]$ < simThreshold
    Increment unmatchedCoinc count

If unmatchedCoinc count = size of C
  add input p to stored coincidences C
  append maxSimilarity to end of $y^t$ vector
  normalize $y^t$ vector
```

Figure 4. Online SP learning and inference.

[1]Recall that when the node has more than one child, each coincidence will be a concatenation of child outputs.

5.1.2 Learning in the Temporal Pooler

The objective of TP learning is to construct sequences from the series of belief vectors (y) received from the SP. When a new input to the TP is received, the TP first calculates the winning coincidence of y^t:

$$topCoinc^t = \underset{i}{\operatorname{argmax}}\,(y^t[i]) \tag{7}$$

It then determines whether this coincidence has changed since the previous time step—i.e., whether $topCoinc^t$ equals $topCoinc^{t-1}$—and stores the result in a flag called *change*.

The next step is to determine whether the transition from $topCoinc^{t-1} \rightarrow topCoinc^t$ exists among the TP's stored sequences. To do this, we depart from the HTM entirely, and use an algorithm we refer to as the Sequencer algorithm. In the Sequencer algorithm, we consider two aspects of the relationship between $topCoinc^t$ and a given stored sequence, Seq^n: 1) the *position* of $topCoinc^t$ in Seq^n (zero if $topCoinc^t \notin Seq^n$), referred to as the "sequencer state", and 2) the cumulative slope formed by the history of sequencer states for Seq^n. Thus, if Seq^n is four coincidences in length, and each successive $topCoinc^t$ matches each coincidence in Seq^n, then the history of sequencer states will be the series: {1, 2, 3, 4}, with each transition having a slope of 1.0. We use a vector called *seqState* to store the sequencer states, and a vector called *seqSlope* to store the cumulative slope for each sequence, formed by the history of sequencer states. The slope is calculated as follows:

$$seqSlope^t[i] = \frac{1}{seqState^t[i] - seqState^{t-1}[i]} \tag{8}$$

$$seqSlope^t[i] = \begin{cases} seqSlope^{t-1}[i] - seqSlope^t[i], & i = 1.0 \\ seqSlope^{t-1}[i] - |seqSlope^t[i]|, & i \neq 1.0 \end{cases} \tag{9}$$

$$seqSlope^t[i] = \frac{2}{1+e^{-seqSlope^t[i]}} - 1 \tag{10}$$

where $seqState^t[i]$ indicates the position of $topCoinc^t$ in sequence i (zero if non-member). The sigmoid scaling performed in Equation 10 helps to constrain the cumulative slope values. Figure 6 shows an example of using cumulative sequence slopes to reveal the best sequence.

At levels above L1, we only update the *seqSlope* vector when *change* = 1, in order to help the TP learn at a time scale appropriate to its level in the hierarchy. A node parameter, *slopeThresh*, is used to determine the minimum slope required for the TP to pass onto the inference stage *without* adding a new sequence or extending an existing sequence. If the maximum value in *seqSlope* does not exceed the value of *slopeThresh*, then either a new sequence is created, or an existing sequence extended.

Generally, we allow only one occurrence of any given coincidence in a single sequence at all levels above L1, though any number of sequences may share that coincidence. This is done to avoid building long sequences at the bottom of the hierarchy, thus dividing the construction of longer sequences across the different levels. We allow consecutive repetitions of coincidences at L1, but do not allow non-consecutive repetitions. This is a musical consideration, given the frequent use of repetitions in musical language.

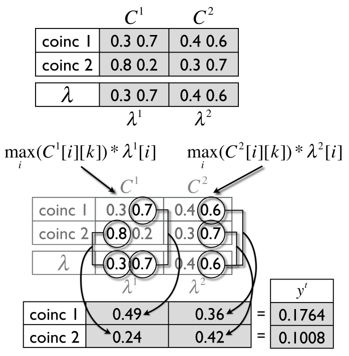

Figure 5. SP inference calculations above L1.

Input patterns

$topCoinc^{t-2}$ = 3	$topCoinc^{t-1}$ = 4	$topCoinc^t$ = 6

Stored sequences		$seqState^{t-2}$	$seqSlope^{t-2}$	$seqState^{t-1}$	$seqSlope^{t-1}$	$seqState^t$	$seqSlope^t$
Seq 1	2 4 5	0	0.0	2	-0.24	0	-0.35
Seq 2	1 3 4 6	2	0.24	3	0.55	4	0.95
Seq 3	3 4	1	0.46	2	0.62	0	0.06

Table assumes that $seqState^{t-3}$ = 0 for all sequences.

Figure 6. Using *seqSlope* to find the best sequence.

5.1.3 Inference in the Temporal Pooler

The objective of TP inference is to determine the likelihood that $topCoinc^t$ is a member of a given stored sequence. At each time step, the TP uses the *counts* vector, from the SP, to update a Conditional Probability Table, called the *weights* matrix, which indicates the probability of a specific coincidence occurring in a given sequence. The *weights* matrix is calculated as:

$$weights[i,j] = counts[j] \times I_{i,j} \,/\, \left(\sum_{i=1}^{k} counts[k] \times I_{i,j}\right) \tag{11}$$

$$I_{i,j} = \begin{cases} 1, & C[j] \in S[i] \\ 0, & C[j] \notin S[i] \end{cases}$$

where $C[j]$ is the j^{th} stored coincidence and $S[i]$ is the i^{th} stored sequence.

The probabilities stored by the *weights* matrix are used during TP inference, and also when forming top-down beliefs in the hierarchy, as introduced in Section 2. It is a row-normalized matrix where rows represent sequences and columns represent coincidences. Because the *counts* vector maintains its histogram of $topCoinc^t$ occurrences over a limited temporal window, the *weights* matrix in the HSMM is able to act as a form of short-term memory for the node.

The output of TP inference is the bottom-up belief vector z, which indicates the degree of membership of $topCoinc^t$ in each of the stored sequences. The argmax of z thus identifies the sequence most strongly believed to be active, given $topCoinc^t$. To calculate z, we use a variant of the "sumProp" and "maxProp" algorithms used in the HTM [6], which we refer to as *p*MaxProp. The algorithm uses the *weights* matrix to calculate a belief distribution over sequences, as follows:

$$z[i] = \max_{j=1}^{i}(weights[i,j] \times y[j]) \tag{12}$$

An example run of the *p*MaxProp algorithm is given in Figure 7, using the coincidences and sequences from Figure 6. Because the *weights* matrix in the HSMM is a

y^t	0.1	0.25	0.5	0	0.1	0.05

coincidences ↓ *

weights	1	2	3	4	5	6		z
Seq^1	0	0.2	0	0.6	0.2	0		0.05
Seq^2	0.14	0	0.29	0.43	0	0.14	max →	0.145
Seq^3	0	0	0.4	0.6	0	0		0.2

Figure 7. The *p*MaxProp algorithm calculations.

short-term memory, and the *p*MaxProp algorithm is a "one-shot" inference, with no consideration of the previous time step, we combine the results of *p*MaxProp with the results of the Sequencer algorithm, to yield the final bottom-up belief vector:

$$z^t[i] = \frac{z^t[i] + seqSlope^t[i]}{2} \tag{13}$$

5.2 Belief Formation in an HSMM Node

The final belief vector to be calculated, a belief distribution over coincidences called Bel^C, represents the combination of the node's classification of a given input, and its prediction regarding that input in the current temporal context. Thus, for every bottom-up input there is a top-down, feedback response. Bottom-up vector representations passing between nodes are denoted with λ, while top-down, feedback representations are denoted with π[2]. A schematic of node processing can be seen in Figure 8. The top-down, feedback calculations used in the HSMM are the same as those used in the HTM, but we outline them here for completeness.

The first step in processing the top-down message is to divide the top-down parent belief π by the node's bottom-up belief λ (at the top of the hierarchy, the bottom-up belief z is used for the top-down calculations):

$$\pi'[i] = \pi[i] / \lambda[i] \tag{14}$$

Next, the π' vector is used to calculate the top-down belief distribution over stored coincidences as:

$$y^{\downarrow}[i] = \max_{Seq_i \in S}(weights^T[i,j] \times \pi'[j]) \quad \text{for } i = 0 \text{ to } |C|-1 \tag{15}$$

where $weights^T[i,j]$ is the transposed *weights* matrix, and $y^{\downarrow}$ is the top-down belief over coincidences, and S is the table of stored sequences. Figure 9 gives an example, assuming the coincidences and sequences from Figure 7.

The Bel^C vector is then calculated as the product of the top-down ($y^{\downarrow}$) and bottom-up ($y^{\uparrow}$) belief distributions over coincidences:

$$Bel^C[i] = y^{\downarrow}[i] \times y^{\uparrow}[i] \tag{16}$$

This calculation ensures that the belief of the node is always based on the available evidence both from *above* and *below* the node's position in the hierarchy. At all levels above L1, the top-down output of the node (the message sent to the children) is calculated using the Bel^C vector and the table of stored coincidences C:

$$\pi[i] = \operatorname*{argmax}_{C[i] \in C}(C[i] \times Bel^C[j]) \quad \text{for } i = 0 \text{ to } |C|-1 \tag{17}$$

[2]At the node level, the λ and z vectors are equivalent. The naming is intended to distinguish the between-node processes from the within-node processes.

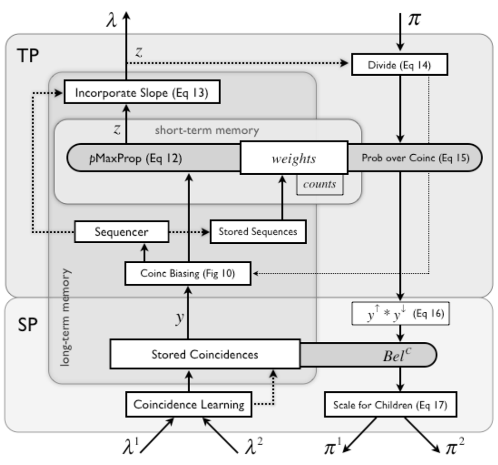

Figure 8. HSMM node processing.

This calculation ensures that each child portion of the top-down output is proportional to the belief in the node. In cases where the parent node has two or more children, the π vector is divided into segments of length equal to the length of each child's λ vector (i.e., reversing the concatenation of child messages used during bottom-up processing). The various stages of top-down processing are illustrated on the right side of Figure 8.

One extra step, in support of TP inference in the HSMM, is added that is not present in the HTM. In accordance with the ideas of the MPF, it seemed intuitively clear to us that predictions could be used locally in the TP to support the learning process by disambiguating SP inputs whenever possible. With this in mind we added a calculation to the TP inference algorithm that biases the SP belief vector, y, according to the state of the *change* flag, and the current top-down inference over sequences. In cases where the sequence inferred by top-down processing at time t-1 contains $topCoinc^{t-1}$, and *change* = 0, the belief value for $topCoinc^{t-1}$ is strengthened. However, when *change* = 1, belief in the *next* coincidence in the inferred sequence is strengthened. The algorithm is given in Figure 10. Thus, when the state of the node appears to be changing, belief is biased slightly toward what is most *likely* to occur, whereas when the state appears to be stable, the most recent belief is assumed to be correct.

π	0.55	0.4	0.05

↓ *

	$weights^T[i,j]$	Seq^1	Seq^2	Seq^3		$y^{\downarrow}$
coincidences	1	0	0.14	0		0.056
	2	0.2	0	0		0.11
	3	0	0.29	0.4	max →	0.116
	4	0.6	0.43	0.6		0.33
	5	0.2	0	0		0.11
	6	0	0.14	0		0.056

Figure 9. Using the *weights* matrix to calculate the top-down belief over coincidences.

6. DISCUSSION AND CONCLUSION

The strength of the HSMM lies in its balancing of hierarchical interdependence with node-level independence. Each node learns in the context of the hierarchy as a whole, but also forms its own representations and beliefs over a particular level of musical structure. At L1, simple motivic patterns can be recognized and/or generated, and at the higher levels, larger musical structures like phrases, melodies, and sections can also be learned, classified, and generated. Further, since nodes above L1 all process information in an identical manner, and only require a single child, additional high-level nodes could be added, enabling the learning of higher levels of formal structure —songs, movements, compositions. Each node can be monitored independently, and its state exploited for compositional, analytical, or musicological purposes. Composition tools could be developed, offering various levels of interactivity, while maintaining stylistic continuity with the user's musical language. In the area of classic Music Information Retrieval, low levels of the HSMM could be used to identify common motivic gestures among a given set of works, while higher levels could be used to recognize the music of individual composers, or to cluster a number of works by stylistic similarity.

$topSeq^{t-1}$	The sequence inferred by top-down processing
predCoinc	The predicted coincidence

```
For each coincidence $c$ in $topSeq^{t-1}$
  If $topSeq^{t-1}[c]$ equals $topCoinc^{t-1}$
    Set $predCoinc$ to $topSeq^{t-1}[c+1]$

If $change = 0$
  $y[topCoinc^{t-1}] = y[topCoinc^{t-1}] * 1.1$
else if $change = 1$
  $y[predCoinc] = y[predCoinc] * 1.1$
```

Figure 10. Biasing the predicted coincidence.

The HSMM exploits short-term and long-term memory structures, and uses an explicit sequencing model to build its temporal hierarchies, thus giving it the capacity to learn high-level temporal structure without the tree-like topologies required by HTM networks.

Tremendous progress has been made in the cognitive sciences and cognitive modeling, but such work has remained largely unexplored by the computer music community, which has focused more on pure computer science and signal processing. The HSMM offers a first step toward the development and exploitation of a realistic cognitive model for the representation of musical knowledge, and opens up a myriad of areas for exploration with regard to the associated cognitive behavior.

7. FUTURE WORK

A working prototype of the HSMM has been implemented, and initial tests have shown great promise. A future paper will cover the evaluation in detail, with an emphasis on exploiting the strengths offered by the cognitive model.

We are interested in exploring alternative distance metrics for the L1 nodes—particularly the pitch and rhythm nodes, where more musically-grounded metrics may be effective. We are also interested in exploring different topologies for the hierarchy, in particular, topologies that isolate individual voices and allow the system to learn both independent monophonic hierarchies and associative polyphonic hierarchies. Along similar lines, we would like to explore the possibilities offered by state-based gating of individual nodes in more complex hierarchies, in order to simulate the cognitive phenomenon of attention direction.

8. REFERENCES

[1] D. Cope: *Computer Models of Musical Creativity.* 87-123, 226-242, MIT Press, Cambridge, MA, 2005.

[2] S. Dubnov, G. Assayag, and O. Lartillot: “Using Machine-Learning Methods for Musical Style Modelling,” *Computer*, 36/10, 2003.

[3] D. Eck and J. Schmidhuber: “Learning the Long-Term Structure of the Blues,” *Lecture Notes in Computer Science*, Vol. 2415, Springer, Berlin, 2002.

[4] D. George. “How the Brain Might Work: A Hierarchical and Temporal Model for Learning and Recognition,” PhD Thesis, Stanford University, Palo Alto, CA, 2008.

[5] D. George and J. Hawkins: “A Hierarchical Bayesian Model of Invariant Pattern Recognition in the Visual Cortex,” Redwood Neuroscience Institute, Menlo Park, CA, 2004.

[6] J. Hawkins and S. Blakeslee: *On Intelligence*, Times Books, New York, NY, 2004.

[7] K. Hirata and T. Aoyagi: “Computational Music Representation Based on the Generative Theory of Tonal Music and the Deductive Object-Oriented Database,” *Computer Music Journal*, 27/3, 2003.

[8] D. Hörnel and W. Menzel: “Learning Music Structure and Style with Neural Networks,” *Computer Music Journal*, 22/4, 1998.

[9] F. Lerdhal and R. Jackendoff: *A Generative Theory of Tonal Music*, MIT Press, Cambridge, MA, 1983.

[10] J. Maxwell and A. Eigenfeldt: “The MusicDB: A Music Database Query System for Recombinance-based Composition in Max/MSP,” *Proceedings of the 2008 International Computer Music Conference*, Belfast, Ireland, 2008.

[11] M.C. Mozer: “Neural Network Music Composition by Prediction: Exploring the Benefits of Psychoacoustic Constraints and Multi-scale Processing,” *Connection Science*, 6/2-3, 1994.

[12] E. Pollastri and G. Simoncelli: “Classification of Melodies by Composer with Hidden Markov Models,” *Proceedings of the First International Conference on WEB Delivering of Music*, 2001.

[13] Y. Uwabu, H. Katayose and S. Inokuchi: “A Structural Analysis Tool for Expressive Performance,” *Proceedings of the International Computer Music Conference,* San Fransisco, 1997.

[14] K. Verburgt, M Dinolfo and M. Fayer: “Extracting Patterns in Music for Composition via Markov Chains,” *Lecture Notes in Computer Science*, Springer, Berlin, 2004

[15] G. Wilder: “Adaptive Melodic Segmentation and Motivic Identification,” *Proceedings of the International Computer Music Conference*, Belfast, Ireland, 2008.