

Hierarchical Sequential Memory for Music: A Cognitively-Inspired Approach to Generative Music

James B. Maxwell and Philippe Pasquier and Arne Eigenfeldt

Simon Fraser University: School for the Contemporary Arts, School for Interactive Art + Technology
jbmaxwel@sfu.ca, pasquier@sfu.ca, arne_e@sfu.ca

Abstract

In this paper we present a refinement of our Hierarchical Sequential Memory for Music framework (HSMM) and discuss preliminary results. The HSMM is an extension of the Hierarchical Temporal Memory (HTM) framework of Dileep George and Jeff Hawkins, intended to make it better suited to musical applications. The HSMM is a machine learning framework, designed to learn hierarchies of sequences, and to make inferences on those hierarchies in a cognitively-inspired process of “bottom-up” and “top-down” information propagation.

Introduction

In our previous work (Maxwell, Pasquier, and Eigenfeldt 2009) we outlined the basic components of the HTM, discussed how these components were adapted for the HSMM, and described any additional algorithms unique to the HSMM. In the current paper we will present a number of modifications and refinements to the model, and discuss the results of our initial tests.

Theoretical models of musical form, like Schenkerian analysis, Lerdhal and Jackendoff’s “Generative Theory of Tonal Music” (Lerdahl and Jackendoff 1984), and Cope’s “SPEAC” system of formal analysis (Cope 2001), all acknowledge the central role that hierarchy plays in generating meaningful musical structures. Indeed, as the product of a highly developed cognitive process, the hierarchical structure of music appears to reflect the underlying mechanisms of human memory (Snyder 2001). It was for this reason that the HTM—a type of Bayesian network, inspired by a high-level model of neocortical function called the Memory Prediction Framework (Hawkins and Blakeslee 2004)—presented itself as an immediately attractive model for the learning and generation of musical forms. The HSMM is an extension of the HTM, which adds online learning, and focuses more strictly on building hierarchies of *sequences*.

Other approaches to learning musical structure include generative grammars and transition networks (Cope 2005), neural network models (Menzel 1998), recurrent neural network models (RNNs) (Mozer 1991), RNNs with Long Short-Term Memory (LSTM) (Eck and Schmidhuber 2002),

Markov-based models (Conklin and Witten 1995) (Wiggins, Pearce, and Müllensiefen 2009), and statistical models (Assayag 2006). With the HSMM, we hope to extend such systems by drawing on the cognitive metaphors of “bottom-up” and “top-down” processing inherent to the HTM’s design.

In the current paper we assume a basic understanding of the ideas underlying the HTM, which can be found in (George, Jaros, and Inc 2007), (George 2009), and (Maxwell, Pasquier, and Eigenfeldt 2009). The following is a list of the main terms and ideas we will be using from this background work:

- 1) **Node**: The basic processing unit of an HSMM network.
- 2) **Coincidence**: A storage type used to represent a unique input to a node. In nodes with more than one child, the child outputs are concatenated together to form the input to the parent, and thus represent the “coincidence” of child states at a particular time.
- 3) **Spatial Pooler (SP)**: A module used to learn coincidences, and to make inferences based on the stored coincidences.
- 4) **Temporal Pooler (TP)**: A module used to build sequences by learning the temporal dependencies among coincidences, and to make inferences based on the stored sequences.
- 5) **BelC**: A vector indicating the final belief over coincidences of the node.
- 6) **Likelihood**: In keeping with the slightly modified terminology used in (George and Hawkins 2009), all inference calculations, other than that of the final *BelC* vector, are referred to as “likelihood” calculations.
- 7) **Feedforward/Feedback**: Messages that pass up the network, from child to parent, are referred to as feedforward messages, while messages passing from parent to child are referred to as feedback.

Motivations: Representing Musical Memory

Although recent research (George 2009) suggests the capacity for high-level sequence learning in the HTM, the current “NuPIC” implementation, available from Numenta Inc., is focused primarily on visual pattern recognition and lacks

the ability to build the sort of high-level sequential structures found in music. The HSMM treats sequential order explicitly, and can thus build detailed sequential hierarchies.

A strong motivation behind the HSMM is the desire for a system capable of learning the hierarchical form of music, but is at the same time capable of incorporating new musical materials, introduced during a given compositional process. That is, we need an online learner, capable of learning new musical fragments and contextualizing those fragments within the scope of a broader musical language. We are particularly interested in a system that can frame this notion of context in terms of the hierarchical structures discovered in the learned material. In consideration of these goals, the HSMM has been designed 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 pattern (i.e., by adding a missing note to a chord.)

Refinements to the Implementation

A number of refinements have been made to the implementation which either generalize the previous approach, or offer more efficient ways of performing the same basic functions.

The Updated Sequencer Algorithms

Because many aspects of musical form are highly sensitive to sequential ordering, the HSMM ensures that common sequential transitions are retained by partitioning the series of inputs into variable length n-grams. Thus, whereas the HTM exploits Markov chains explicitly, the HSMM uses a method with stronger connections to the “context models” of (Conklin and Witten 1995).

The process of testing our previous implementation (Maxwell, Pasquier, and Eigenfeldt 2009) revealed a number of areas that could be refined, simplified, or generalized. Most of these refinements concern the algorithm we refer to as the “Sequencer” algorithm. In its original form, this algorithm was used to determine the membership of an input coincidence in each of the stored sequences, given the history of inputs to the node. Although this remains the primary focus of the Sequencer, the process itself has been generalized, and the data structure used in its implementation has also been exploited for the feedback calculations.

The Sequencer’s role in calculating the feedforward likelihood over sequences remains essentially unchanged. However, in place of the previous “slope” calculation, we now use a probabilistic calculation. Each sequence represents an

Input Sequence {1, 3, 2, 6, 5}										
Sequence Table					state					
	s_1	s_2	s_3	s_4	s_1	s_2	s_3	s_4	s_5	
s_1	1	3	2	5	4	1	2	3	0	4
s_2	4	3	2	6	5	0	2	3	4	5
s_3	4	5	6	5		0	0	0	3	4
s_4	3	4	2	3	4	0	1	3	0	0

Likelihood over Sequences					
	s_1	s_2	s_3	s_4	s_5
s_1	0	1	.67	.33	.11
s_2	0	0	.33	.67	.56
s_3	0	0	0	0	.33
s_4	0	0	0	0	0

Figure 1: Feedforward TP Likelihood Calculations.

observed series of coincidences, and the Sequencer’s *state* vector indicates the position of the current input coincidence in each of the stored sequences. The *state* vector is updated as before, and indices are one-based (with zero indicating non-membership), as in the original model. The identification of sequences containing the current transition is done as follows:

$$trans^t[i] = \begin{cases} 1 & \text{if } state^{t-1}[i] > 0 \\ & \wedge state^t[i] - state^{t-1}[i] = 1 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where *trans* is a vector of length equal to the number of stored sequences, indicating which sequences contain the current transition. The first condition limits the assignment of transition values to within-sequence transitions, skipping transitions from non-membership ($state^{t-1}[i] = 0$) to sequence beginnings ($state^t[i] = 1$). All sequences containing the transition (i.e., $trans^t[i] = 1$) are thus assigned a 100% probability for the current time step. The $trans^t$ vector is then added to the feedforward likelihood over sequences from the previous time step (z^{t-1}) and the result is normalized to sum to 1.0:

$$z^t[i] = z^{t-1}[i] + trans^t[i] \quad (2)$$

$$z^t[i] = \frac{z^t[i]}{\sum_{k=1}^{|S|} z^t[k]} \quad (3)$$

where z^t is the feedforward likelihood over sequences given the current input coincidence, and S is the set of all stored sequences. In these calculations, the likelihood vector z is accumulated and normalized at each time step, ensuring that the likelihood distribution expressed by z^t is theoretically dependent upon all preceding values of z , from time $t = 0$ to the current time step. A sample run of the Sequencer’s feedforward algorithm is given in Figure 1, which shows the determination of the *state* for a series of five input coincidences, given four stored sequences, and indicates the feedforward likelihood vectors produced over the five time steps.

Perhaps the most significant change implemented in the revised HSMM, however, can be found in the modified TP feedback calculations. The function of TP feedback is to derive a likelihood distribution over coincidences, when presented with a likelihood distribution over sequences from above. That is, given a degree of likelihood that a certain sequence is active, to estimate how likely it is that a given coincidence in that sequence is active. The calculation used in the original HSMM followed the method documented for Numenta’s NuPIC HTM implementation (George, Jaros, and Inc 2007), which exploited a conditional probability table called the *weights* matrix. In the HTM, this table gives the probability of coincidences, over all sequences, based on a histogram of coincidences seen during training. Because the HSMM, unlike the HTM, is an online learner, the process of updating the *weights* matrix was modified for the HSMM, but the principle was essentially the same.

In the revised HSMM feedback calculations, however, we now exploit the *state* vector used by the Sequencer for the feedforward TP calculations. In the feedback case, however, we want to determine the set of possible coincidences that *might* be active at the current time step, and assign some likelihood value to each one. To do this, we first use the $state^{t-1}$ vector to derive the set of all possible coincidences that could *potentially* occur at the current time step:

$$pCoinc^t[i] = \begin{cases} S[i, state^{t-1}[i]] & \text{if } state^{t-1}[i] > 0 \\ & \wedge state^t[i] > 0 \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

where $pCoinc^t$ represents the set of coincidences that would occur as a result of stepping all sequences forward from the *previous* time step—that is, the *continuation* of all sequences from $state^{t-1}$. Note that, because the *state* vector uses one-based indexing, and the sequence table S uses zero-based indexing, $S[i, state^{t-1}[i]]$ will return the *next* coincidence in sequence i —that is, the coincidence that *might* have occurred, had the new input followed the transitions in sequence i . The requirement that $state^{t-1}[i] > 0 \wedge state^t[i] > 0$ serves to limit support for sequences which do not contain the transition, or have become inactive at the current time step.

With the set of possible coincidences determined, we then accumulate support for each coincidence, using the values provided by the feedback likelihood over sequences:

$$\gamma^t[pCoinc^t[i]] = \gamma^{t-1}[pCoinc^t[i]] + \varepsilon^t[i] \quad (5)$$

$$\gamma^t[i] = \frac{\gamma^t[i]}{\sum_{k=1}^{|S|} \gamma^t[k]} \quad (6)$$

where γ^t is the feedback likelihood over coincidences, and ε^t is the feedback likelihood over sequences. By accumulating and normalizing values from the last time step, in a manner similar to the feedforward Sequencer calculations, we ensure that the likelihood expressed by γ^t is dependent upon all preceding values of γ , from time $t = 0$ to the current time step. Figure 2 gives a sample run, showing the

Input Sequence {1, 3, 2, 6, 5}		$pCoinc$					
Sequence Table		γ	γ	γ	γ	γ	
S_1	1 3 2 5 4	S_1	0	3	2	0	0
S_2	4 3 2 6 5	S_2	0	0	2	6	5
S_3	4 5 6 5	S_3	0	0	0	0	5
S_4	3 4 2 3 4	S_4	0	0	3	0	0

Likelihood over Coincidences		γ	γ	γ	γ	γ
$C_1[1]$	(.5)	.25	.13	.08	.04	
$C_2[2]$	(.2)	.1	.55	.33	.15	
$C_3[3]$	(0)	.5	.25	.15	.07	
$C_4[5]$	(.3)	.15	.07	.04	.57	
$C_5[6]$	(0)	0	0	.4	.18	

Figure 2: Feedback TP Likelihood Calculations.

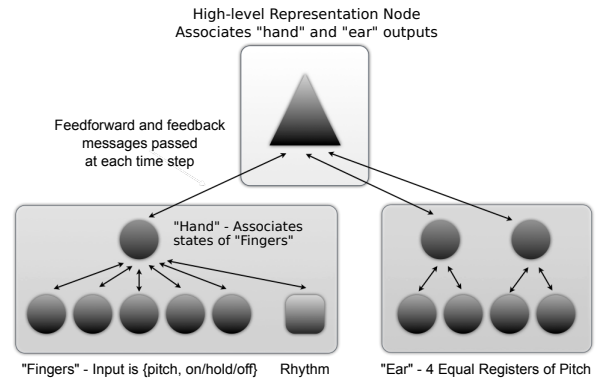


Figure 3: A “Sensory-Motor” HSMM Topology.

determination of the $pCoinc$ vector and the feedback likelihood over coincidences, for the same five inputs and four stored sequences, and assuming the *state* vectors from Figure 1 (the bracketed values for time $t - 4$ indicate that these likelihoods cannot be known).

Music as Motion: The HSMM Test Topology

In designing the music representation, we were interested in expressing an intuition that motor activity plays an important role in music perception and cognition. This intuition is not without support. Findings in (Zatorre, Chen, and Penhune 2007) and (Popescu, Otsuka, and Ioannides 2004) show that motor areas are recruited for the processing of musical information during listening. To express this connection, we took a sensory-motor approach, using two distinct representations for each musical event. The system works with symbolic music representations, at this stage, and for the sake of simplicity uses MIDI note values to represent pitch, and inter-onset times to represent rhythm. A more complex, invariant representation scheme is given in (Maxwell, Pasquier, and Eigenfeldt 2009). The topology used is shown in Figure 3.

Here we can see the division of the sensory and motor

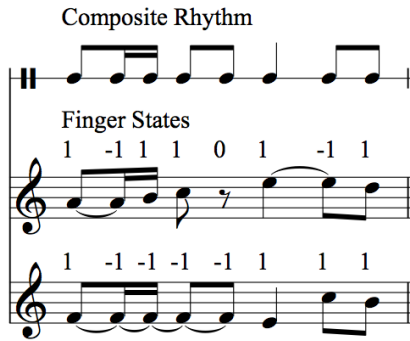


Figure 4: Articulating two parts with composite rhythm and “finger” states.

sides, with the sensory side (on the right) dividing the total MIDI pitch range into four equal registers. Each register is expressed as a 32-member pitch set vector—that is, the indices of non-zero values indicate active MIDI note numbers. On the motor side (left), polyphonic parts are represented in a manner analogous to monitoring the state of activity of “fingers” during a performance. Each finger node receives a two-member vector consisting of a MIDI note value and a finger state value; 1 for a note attack, -1 for a sustained note, and 0 for a note release. The single rhythm node receives inter-onset values, expressed in milliseconds, for each change of state in the fingers. This amounts to learning a sort of composite rhythm, which expresses the sum of onsets and offsets distributed across all instrumental parts. The rhythmic contour of an individual part is thus expressed as the combination of the composite rhythm and the state of the finger (1, -1, or 0). Figure 4 gives an example showing the articulation of two parts with a single composite rhythm.

As a side-note, this representation technique hints at another intuition of ours relating to the role of motor areas in music perception and cognition. While the “ear” will clearly perceive the composite rhythm, it is the “fingers” that articulate the individual parts, and thus, perhaps, serve as a guide for higher level recognition of individual parts in polyphonic textures. This is not to suggest that such a role *must* be played by fingers; it could just as well be handled by the voice, or even a tapping foot. The point is simply that the motor activity becomes associated with a single auditory stream, and thus serves as an aide to training higher cognitive functions of auditory scene analysis. Thus the process of an infant associating the muscular activity of crying with the auditory perception of its own crying voice could serve, for example, as the initial training required for isolating a single speaking voice in a crowded room.

In the implementation used for testing, the HSMM was coded in Objective-C and C, for Mac OS X 10.6. To optimise performance, the *Grand Central Dispatch* API was used, allowing the nodes on each level to process concurrently, depending on available hardware resources. MIDI input/output was handled by a MaxMSP patch, which communicated with the HSMM over Open Sound Control (OSC). The “VVOSC” framework was used for OSC support in the

Objective-C code. For the sake of simplicity, the SP of L1 nodes did not perform any actual clustering, as in the HTM, but simply stored *all* unique input representations. Pitch representations were simple MIDI note values, and inter-onset values were quantized in the MaxMSP patch, thus limiting the number of possible coincidences. During generation, on-line learning in the HSMM was temporarily disabled, in order to constrain the output to representations learned directly from the training material.

Music In, Music Out: The Generation Algorithms

The generation of output from an HSMM is a complex task, for which a number of different approaches could be taken. Since the processing of inputs necessarily points the state of the system toward a particular area in the space of representations, at each level in the hierarchy, the function of “continuation” is closely connected to that of generation. We discuss one possible approach, in the context of continuation, with the understanding that generation operates on the same principles.

As our sensory-motor topology and music representation imply that only one side of the HSMM—the side of the “hand” and “fingers”—will be responsible for generation, we will only be concerned with generating events at the “finger” nodes and the rhythm node (for the rest of this discussion we will refer to this set of nodes simply as “L1”). Our current approach has been to use prediction initiated at the “hand” node, which is passed down to L1. The decision to initiate at the hand (L2), rather than L1, was based on the understanding that the hand node is the first node to build representations that capture correspondences between the states of the individual fingers, and therefore, will be the lowest level capable of generating syntactically valid polyphonic output.

We begin by finding the currently active coincidence and sequence at the hand node, given the input. For the coincidence, this is simply a matter of finding the maximum “belief” of the node, expressed as $\arg \max_i BelC[i]$. However, when determining the current sequence, we are already faced with a choice: whether to use the node’s feedforward or feedback likelihood over sequences. The feedforward likelihood would perhaps be the more reliable of the two, as it is more strongly connected to the most recent inputs. On the other hand, the feedback likelihood (which is the feedback output of the parent, and hence a *coincidence* of the parent) has high-level temporal dependencies not present in the feedforward message.

As one possible solution, we chose to take the sum of the feedforward and feedback likelihoods:

$$seq = \arg \max_i (z^t[i] + \varepsilon^t[i])[i] \quad (7)$$

where seq is the most likely sequence, z is the feedforward likelihood over sequences, and ε is the feedback likelihood. This provides the possibility of negotiating between the feedforward and feedback messages, giving a sort of makeshift form of saliency detection. More sophisticated approaches to saliency modelling, for the purposes of negotiating such decisions, are left for future research.

Training the HSMM

For the current study we are testing the system's capacity for recognition, generation, and continuation; completion will be tested in a future study. Because we are most interested in output generation, we trained the system on a single work, making it easier to identify training materials in the generated music. The training work chosen was the Fugue from BWV 846 of Bach's Well-Tempered Klavier, which contains complex, polyphonic writing, and thus served as a demanding test case. Although the system is an online learner, we used an offline approach when running the initial training.

First, we ran live MIDI file playback through the representation encoding routine, and recorded the output. This gave us a sort of 'training script' that could be run iteratively through the system for rapid, multi-pass training. During the encoding, we clustered the MIDI events under 20 ms windows, in order to reduce the rate of data flowing into the HSMM, and also to help generalise the input representations into something similar to Cope's "Groups"—vertical 'slices' of music giving the total harmonic content at a given moment (Cope 2005). We ran the script in stages, each consisting of 5 iterations over the entire composition, and evaluated the system's performance at each stage.

During output, the HSMM decoded its coincidences, and passed MIDI representations back to MaxMSP for playback. Because the windowing scheme used for encoding the inputs led to situations in which a single voice could transition between consecutive note onset events (i.e., MIDI noteons), we wrote a simple MaxMSP patch to ensure that previous notes were released when a note onset arrived for a currently active voice.

Results

Recognition was tested by passing the scripted music representations to the HSMM, with onset delays generated by the script itself. Essentially, this amounted to playing the MIDI file to the HSMM directly—the script was used simply to ensure that rhythmic representations remained consistent under heavy system load. The HSMM was responsible for providing the live MIDI output, and thus performed a task similar to 'score following.' The system's ability to produce MIDI output, given an input representation, was generally strong, though we immediately noticed problems correlating pitch sequences with rhythmic patterns. We suspect this is due to the significantly smaller space of rhythmic representations learned, when compared to pitch and finger state representations. A method for encoding finger states and rhythmic values in a common input vector representation would likely improve performance in this regard.

Continuation was tested by toggling the HSMM's input between the scripted data and the HSMM's own generated output—that is, alternating input between the scripted events used in the Recognition tests, and the HSMM's own output. Toggling was performed arbitrarily during playback, sometimes following intuitively clear segment transitions in the music, and sometimes deliberately avoiding such transitions. With 5 to 10 training passes the system was generally capable of continuing in a harmonically related man-

ner, though it had a tendency to "loop" short musical fragments, and seemed to lack any obvious exploitation of motivic material. However, with continued training (from 15 to 20 training passes), the system did begin to produce continuations which exploited motivic fragments from the source work. Unfortunately, along with the increased capability to generate more defined motivic elements, the system also became more prone to looping.

Generation was tested by sending the HSMM a randomly chosen representation from the script as a starting point, then looping the HSMM's output back as input. As would be expected, results followed a similar pattern, with somewhat arbitrary output after the first 5 to 10 training passes, followed by increased ability to form identifiable motives with further training. Similarly to the continuation testing, as the system developed more integrated motivic output, it also showed a greater tendency to fall into looped patterns.

Although the harmonic structure of the output material tended to be fairly good (i.e., stylistically appropriate), the system did produce harmonic anomalies not present in the training material. This was expected, as the process of generating events at the "hand" level did not completely eliminate the need for probabilistic inference of coincidences at L1. For this reason, it was still possible for a given L1 node to generate output in a manner which *did not* acknowledge the output of its siblings.

Discussion and Conclusion

This initial testing of the HSMM, in a strictly musical context, showed definite promise, but also revealed some interesting problems and challenges. Although many aspects of the output were expected, the tendency to fall into loops was not. Further testing, while monitoring the internal state of the system, is necessary to determine whether this behaviour is pointing toward an implementation problem, or a systemic problem in the framework.

The looping behaviour appears to be caused by an inability for higher level nodes to progress beyond sequence boundaries. One possible solution we are considering is to approach the generation as a 'planning' process, in which longer musical passages would be generated *internally* by the system, and held in a temporary memory buffer. For each coincidence in the 'planned' output sequence, the system would first *confirm* its proposed output by passing it up the network. If the proposed output generated strong recognition, as indicated by belief levels in the network (i.e., *BelC* vectors), then the coincidence would be output. If a weak recognition was detected, the system would skip forward to the next proposed output in the buffer.

Of course, we also need to test the system for memory capacity, and for its ability to partition its learned memory space into stylistically distinct regions—i.e., by training it on contrasting musical styles and testing its capacity to continue in a stylistically appropriate manner.

Future Work

The most important task, at this stage, will be to isolate the cause of the looping behaviour reported in the Discus-

sion. Once we have found a solution to this problem, we will move on to designing a more sophisticated approach to “saliency” modelling. Our goal, in this regard, is to find a method for controlling the system’s behaviour through introspection of its internal state. We believe that such an approach could lead to more refined use of motivic material, by prioritising ‘cognitively salient’ sequences during generation. We are also interested in the possibility of generalising sequence recognition across the L1 “finger” nodes, so that all nodes could exploit the same motivic/sequence material. In the NuPIC HTM, this is facilitated by an optional mode which “clones” coincidences between nodes on the same level. We are considering implementing a similar function, though we are also investigating methods for achieving the desired result through alternate topologies and representations.

References

- Assayag, G. Dubnov, S. 2006. Universal prediction applied to stylistic music generation.
- Conklin, D., and Witten, I. 1995. Multiple viewpoint systems for music prediction. *Journal of New Music Research* 24(1):51–73.
- Cope, D. 2001. *Virtual Music*. Cambridge, MA, USA: MIT Press.
- Cope, D. 2005. *Computer Models of Musical Creativity*. Cambridge, MA, USA: MIT Press.
- Eck, D., and Schmidhuber, J. 2002. Finding temporal structure in music: Blues improvisation with lstm recurrent networks. In *Neural Networks for Signal Processing XII, Proc. 2002 IEEE Workshop*, 747–756. Citeseer.
- George, D.; Jaros, B.; and Inc, N. 2007. The htm learning algorithms. *March* 1:2007.
- George, D., J. H. 2009. Towards a mathematical theory of cortical micro-circuits. *PLoS Computational Biology* 5(10).
- Hawkins, J., and Blakeslee, S. 2004. On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines. *Henry Holt Company, New York, NY*.
- Lerdahl, F., and Jackendoff, R. 1984. *A Generative Theory of Tonal Music*. Boston, MA, USA: MIT Press.
- Maxwell, J.; Pasquier, P.; and Eigenfeldt, A. 2009. Hierarchical sequential memory for music: A cognitive model. In *International Society of Music Information Retrieval*.
- Menzel, W. 1998. Learning musical structure and style with neural networks. *Computer Music Journal* 22(4):44–62.
- Mozer, M. 1991. Connectionist music composition based on melodic, stylistic, and psychophysical constraints. *Music and connectionism* 195–211.
- Popescu, M.; Otsuka, A.; and Ioannides, A. 2004. Dynamics of brain activity in motor and frontal cortical areas during music listening: a magnetoencephalographic study. *Neuroimage* 21(4):1622–1638.
- Snyder, B. 2001. *Music and memory: an introduction*. The MIT Press.
- Wiggins, G.; Pearce, M.; and Müllensiefen, D. 2009. Computational modelling of music cognition and musical creativity. In Dean., ed., *Oxford Handbook of Computer Music and Digital Sound Culture*. Oxford, UK: Oxford University Press.
- Zatorre, R.; Chen, J.; and Penhune, V. 2007. When the brain plays music: auditory-motor interactions in music perception and production. *Nature Reviews Neuroscience* 8(7):547–558.