Exploring Reinforcement Learning for Mobile Percussive Collaboration

Nate Derbinsky
Computer Science & Engineering Division
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
nlderbin@umich.edu

Georg Essl
Electrical Engineering & Computer Science and Music
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
gessl@eecs.umich.edu

ABSTRACT

This paper presents a system for mobile percussive collaboration. We show that reinforcement learning can incrementally learn percussive beat patterns played by humans and supports real-time collaborative performance in the absence of one or more performers. This work leverages an existing integration between urMus and Soar and addresses multiple challenges involved in the deployment of machine-learning algorithms for mobile music expression, including tradeoffs between learning speed & quality; interface design for human collaborators; and real-time performance and improvisation.

Keywords

Mobile music, machine learning, cognitive architecture

1. INTRODUCTION

Our research goal is to develop systems that support real-time, collaborative musical expression on mobile devices. In this paper, we focus on percussion, and how machine-learning techniques can facilitate a collaborative performance, learning beat patterns from multiple sources in a local network.

Drum circles and other collaborative drum performances need a certain number to keep the rhythm going. Yet new participants may join or have to leave. When mobile devices are used as drumming interfaces we gain the potential to learn from a performer who is currently participating in the performance, or recall prior performance patterns of the performer. Hence we can construct a drum-circle performance where humans can play, but as needed have learning agents substitute for them should they not be present. Thus we arrive at a seamless transition between networked all-human to all-machine performance, where the machines learn to reproduce the performance of their human mentors.

As a first step, we are exploring reinforcement learning (RL) [14] as a technique by which to incrementally learn beat patterns of individual performers in an online fashion for real-time response. Over time, RL learns what *actions* a system should take in an *environment* such as to maximize the expected receipt of future *reward*. In this problem domain, an action is either to issue a beat, or not, at each point in time and the environment consists of a beat sequence being produced by a single performer. If RL succeeds in learning individual percussive policies, we can then apply this learned knowledge to facilitate collaborative performance in the absence of one or more human performers.

As with many machine-learning algorithms, however, there are numerous considerations when applying RL. The first is *feature selection*: what are the aspects of the performance upon which the system should condition as it learns action utility. The full space of relevant features is likely vast, such as time-series information, performance style, human error, etc. and it may take significant experimentation to identify features that lead to fast learning and interesting performance. Another important consideration is the *reward signal*: how, when, and to what degree is the system provided feedback about its learned policy relative to the performance. The *exploration policy* is another important factor: under what conditions and to what degree should the system deviate from its learned policy, such as to potentially improve its future performance. In context of a musical domain, exploration is closely related to the system's ability and tendency to improvise. A closely related concept is the *learning rate*: to what extent should the system expect a stochastic input signal, which relates to its ability to remain robust to performance errors, but also to encode acceptable forms of performer improvisation and quickly adapt to rhythm changes.

While traversing this design space of reinforcement-learning factors, there are additional implementation and evaluation challenges that stem from collaborative musical expression on mobile devices. First, there are issues of interface design: the system should make it easy for human performers to collaborate on musical pieces, but also to understand their interactions with an adaptive learning system. Second, there are tradeoffs in the space of automated performance quality and improvisation. It is desirable that the learning system quickly adapts to a performer's beat patterns, and can reproduce it with high fidelity, but there may also be a desire for controlled deviation, to avoid a "robotic" quality. Finally, for the system to be useful for interactive performance, all processing in the system, including learning, communication between devices, and sound processing, must execute in real time.

To explore systems that contend with these challenges, we leveraged prior work that integrated urMus, a mobile-music meta-environment, with Soar, a functional cognitive architecture [2]. Soar [10] provides a highly optimized, integrated framework of machine-learning mechanisms, including reinforcement learning. It is also sufficiently general, such that we can easily develop agents for our percussive collaboration task, and highly configurable, such that we can explore many parts of the RL design space. urMus provides the ability to quickly develop and experiment with music generation, performer communication, and interface design, all on a cross-platform environment for mobile devices [4, 6]. The integration supports an arbitrary number of agents to enhance any urMus interface element with real-time learning and decision-making.

In this paper, we present work on applying the urMus/Soar integration to the problem of mobile percussive collaboration. We describe our system design, which includes a Soar agent that interactively learns percussive rhythms using RL as well as novel urMus user-interface elements that assist performers to interact with and understand the adaptive process. We also present results of how a small space of feature representations, learning rates, and exploration policies affect learning and performance of a data set of drum-solo patterns. The system is implemented on networked iOS devices utilizing zeroconf addressing for automating connectivity [5].

2. RELATED WORK

Machine-driven rhythm and collaborative performance has seen extensive interest. Pachet presented a GUI based rhythm generator in which agents would drive individual rhythmic voices [13] taking a pre-defined rule-based approach. Brown [1] explored cellular automata (CA) for the same purpose. Live collaborative drumming was developed by Weinberg and his students [16, 9]. An autonomous drumming robot plays with human performers and the emphasis was the learning of joint improvisation. François *et al* [8] discuss visualization to support joint improvisation between human and machine. Levisohn and Pasquier proposed the use of subsumption architecture in the autonomous generation of rhythm, using simple layered rules to allow rapid rhythm generation. The Kinetic Engine [3] by Eigenfeldt explored networking and distributed agent-based systems to immitate collaborative drumming using fuzzy rules. Martins and Miranda use neural-network based agents to study emergence and evolution of rhythms [11]. Tidemann [15] developed a detailed learning scheme that takes arm motions of the drummer into account to allow detailed immiation of a non-human drummer using echo-state networks. The benefit of learning in live musical interactions has been demonstrated by Fiebrink [7].

Our work differs from previous works in multiple ways. For one we do not seek to generate rhythms or to make a machine collaborate with humans, but rather we seek to support on-the-fly-learning of rhythmic performance from a live performer and have the agent serve as a substitute for human players. The second is the use of mobile commodity devices for this purpose. Our reinforcement learning approach makes minimal assumption about musical structure while introducing parametric control of performance of the agent. In this work, we are motivated to understand how simple yet powerful machine learning techniques can facilitate live music performance.

3. SYSTEM DESIGN

We have implemented a Soar agent that learns and performs percussive patterns and integrated it within an urMus interface.

3.1 Soar Agent

The Soar agent assumes that time is discretized and categorized: unit-length time steps are spent either learning or performing. During a learning time step, the agent first brings to bear its current performance knowledge, as informed by the features that represent its current state. This knowledge will encode a utility policy of issuing a beat, or not. For instance, "given that in the last time step a beat occurred, the value of issuing a beat in the next time step is 0.54 and not issuing a beat is 0.2." The current exploration policy takes as input this utility information and decides whether the system should beat or not. Finally, the agent self-rewards based upon whether its decision matched the action of the human performer. We supply a -1 reward for incorrect action, and +1 for correct.

For this agent, Soar implements the SARSA online TD-learning algorithm [14]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

At a high level, this algorithm updates its expectation of the value (Q) of taking action a in state s with a proportion (α) of the difference between its prediction and experienced reward, where experienced reward is defined as the sum of immediate reward, r_{t+1}, and the discounted expectation (by γ) of the next selected action, $Q(s_{t+1}, a_{t+1})$. For this work, we use a static discount rate ($\gamma = 0.9$), whereas we experiment with the learning rate (α).

During a performance step, the agent makes use of the same action-utility information and exploration policy, but no reward is supplied and no updates are made to the agent's encoding of action utility. Note that the agent's perception of time is maintained independently for each category, and thus learning can either take place in batch before/after performance, or can be intermixed with performance (as would be useful during interactive collaboration).

Soar represents action-utility information as if-then rules: the conditions dictate the features of state as well as the intention to beat/not beat, while the rule action captures a numeric preference, indicating an expectation of future discounted reward [12]. This representation supports a flexible, non-tabular characterization of state-action utility estimation (i.e. different states need not use the same set of features). Furthermore, because multiple rules can fire in parallel, we experiment with coarse-coding representations, where we distribute utility information over multiple levels of generality.

As is common in RL systems, Soar draws from a Boltzmann distribution in order to select actions [14]. Boltzmann is an example of a *softmax* rule, where the greedy action is given highest selection probability, but all other possibilities are ranked and weighted according to their value estimates. The distribution chooses an action a, from amongst all n possible actions in state s at time t, with probability

$$\frac{e^{Q(s_t,a_t)/\tau}}{\sum_{b=1}^{n} e^{Q(s_t,b_t)/\tau}}$$

where the *temperature* parameter (τ) influences the relative weighting between alternatives: a high value weights towards random selection, whereas a low value weights towards greedy. We experiment with different values of the temperature parameter, which influences the likelihood of beating/not beating, given identical prior experience.

3.2 urMus Interface

The urMus application (see Figure 1) supports a number of UI elements to make it easier for human performers to produce percussive rhythms; train the Soar agent and interpret learning progress; and collaborate with other human performers and Soar agents.

3.2.1 Performance UI

The performance interface's central feature is a simple drum button. It is used to both perform live drumming as well

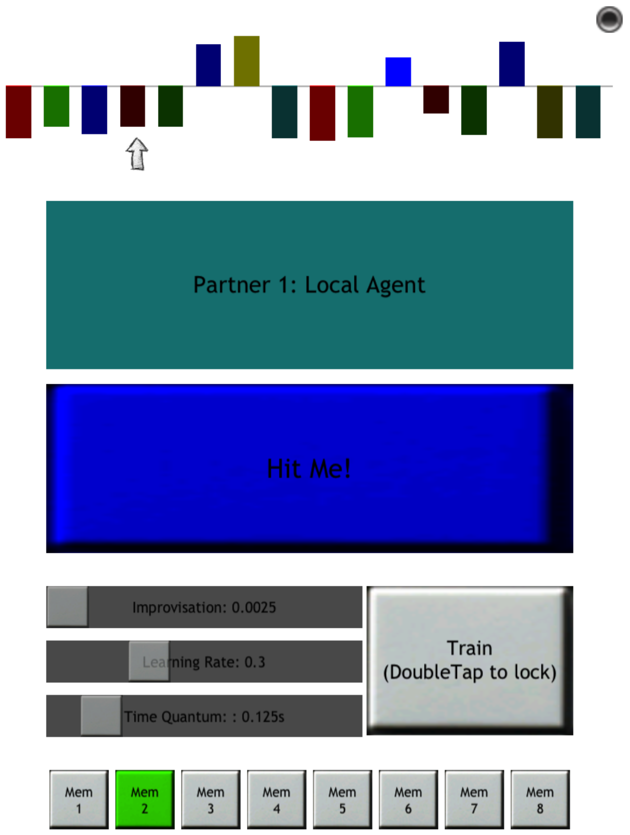

Figure 1: The urMus interface for the agent-learning collaborative drumming instrument.

as serve as the training input. The performance region is marked as an elevated button in the center of the screen labeled "Hit Me!".

In order to satisfy Soar's assumption regarding time discretization, this input is buffered and encoded within the nearest 125 msec, though this quantization granularity can be controlled by a slider in the bottom part of the interface.

Hence there is a minimum temporal quantization happening with respect to training. We experimented both with synchronizing the button's responsiveness with these quanta, and with allowing free performance of the drum button while quantizing its effect into training bins. Through empirical use, we found that delaying auditory feedback introduced user confusion and was interpreted as unresponsiveness due to unintentional system lag. Hence we do not tie the training quantization to the button responsiveness. However, this quantization is important. Hence we offer two main forms of feedback. One is a learned pattern display at the top of the device with an arrow indicating the currently active learning bin. Furthermore, the arrow will change color in sync with an auditory metronome beat, which ticks 4 times per repeating cycles.

3.2.2 *Learning UI*

Our first implementation of the system had the Soar agent learn at each discretized time step. A difficulty in this formulation is that if the performer stops playing, for even a short period of time, the agent overwrites prior pattern knowledge with null beats. To address this problem, we introduced a learning-control switch that has two forms of interaction. For short sequences, the control supports a tap-and-hold input, where the agent learns only as long as contact is maintained with the switch. For longer sequences, the control supports a double-tap input to toggle learning. This control allows the performer to apply top-down control, focusing learning during important beat sequences. We buffer beats to the nearest 16 time steps in order to synchronize knowledge between learning sessions.

We also found that it was useful for the performer to have some indication of learning progress, and so we devised a visualization of the action-utility policy (see Figure 2). Each bar represents a discrete time step and the vertical bar indicates the greedy policy decision at that time step: direction indicates action decision (up=beat, down=no beat) and the height indicates the degree of bias towards that decision (bigger = greater expected utility difference between decisions). During empirical usage, we have found this visualization to be useful feedback for assessing learning progress as well as detecting human errors in beat timing.

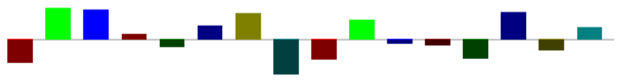

Figure 2: Visualization of the action-utility policy learned by the system for a 16-beat pattern.

3.2.3 *Collaboration UI*

We utilize zeroconf networking to connect to other performers using the same application in a local wireless network. Details of the implementation of zeroconf networking in urMus are described in earlier work [5]. Performers can simply join and leave a present networked performance by starting and exiting the applicaton within the network. The first application will advertise the performance if it fails to discover one that is already present (see Figure 3). The instrument can be used solo. In this case the soar agent can serve as a duo partner. The performer can train the agent for a certain type of rhythm and them, while it is performing play another rhythm juxtaposing it. In general a performer can train its local agent. In order for the performer to have access to a range of trained patterns a number of memory banks can be used and recalled. These can be loaded by local performance, or alternatively loaded from a networked performer.

The interface displays collaborator bars. The number of these bars scales, but at least one is always present for the local agent. The bar displays if a user or an agent is cur-

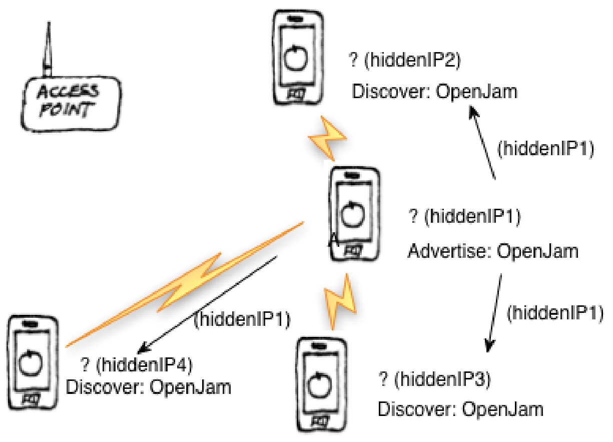

Figure 3: Network topology of the zeroconf discovery. The first device advertises, the rest discover.

rently playing. Human and agent performers have different color schemes, with humans using plain red and blue while agents using pastel versions of these colors. A network participant can send a pattern to another network participant by clicking on their network representation bars on the interface after a action-utility pattern memory has been selected. The recipient will be alerted to this and can either capture the pattern in their local memory by selecting a memory button, or dismiss it by clicking the network participant button from whom the pattern was sent. Hence performers can share learned patterns they find interesting or worthwhile.

In principle, sounds used can be picked at random from sample files. We however tend to use clap sound for self-performance and base-drum sounds for agent-based performance to distinguish between the two cases. The metronome is a closed hi-hat sound.

Each performer can record up to eight action-utility patterns in memory cells by selecting them while recording. Alternatively such patterns can be received from other network performers and stored.

4. EVALUATION

Our evaluation consisted of three research directions: (1) what effect would generalization in the feature representation have on learning performance? (2) what combination of learning rate and exploration rate would best tradeoff learning speed/quality and improvisation? and (3) was our system capable of learning and performing in real time? Before presenting results, we discuss the data sets, as well as experimental methodology and conditions.

4.1 Data Sets

To evaluate learning, we used the first 16 beats of five drum patterns[1]: "The Wanton Song" by Led Zeppelin, "Chuckin Sixteenths," "I Got the Feeling" by James Brown, "The Solid," and "Ticked." These are the base-drum patterns and a verification training run of 16 repetitions at 0.6 learning rate for each is shown in Figure 4.

4.2 Experimental Methodology

We performed all experimentation on an iPad 2 that was attached to a power source. We provided the learning system 10 sequential training trials, during which we evaluated 16-beat performance output in 10 independent episodes. We defined the accuracy of a performance as the proportion of beats produced that matched the input beat pattern. Therefore, each of the experimental conditions has 50 data points per trial. In addition to performance accuracy, we measured the maximum time it took the Soar agent to make a decision, which is a measure of the reactivity of the system for real-time performance.

4.3 Experimental Conditions

One experimental condition was the degree of noise in the input signal for the learning system. We utilized 5 conditions in which we introduced noise, during training, into the data sets. The *baseline* condition had no noise. For the remaining conditions, we assumed an independent, uniform probability of error for each of the 16 beats. The *point* condition represented a specific time step of difficulty, whereby error before the time step was 10%, error on a time step was 35%, and error following the time step was 25%. We also had three *systemic* conditions, whereby all time steps had an equal 10%, 20%, or 30% chance of error, which was

[1]http://www.onlinedrummer.com

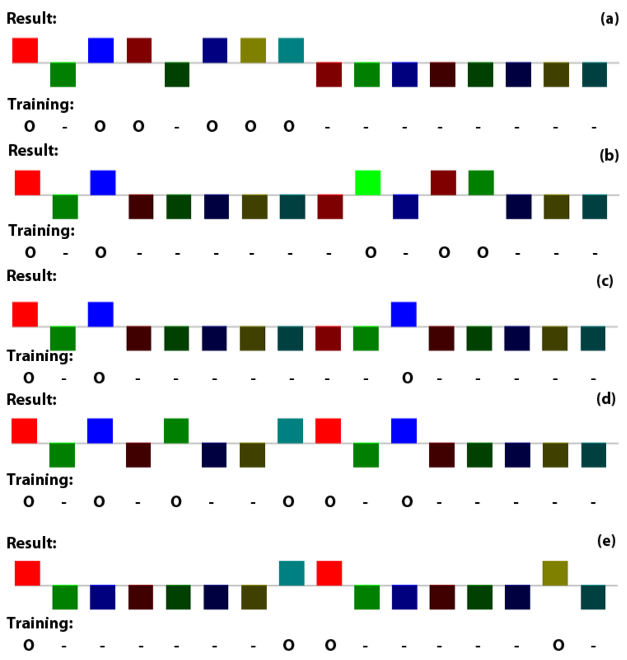

Figure 4: Training set and resulting pattern of (a)"The Wanton Song," (b) "Chuckin Sixteenths," (c) "I Got the Feeling," (d) "The Solid," and (e) "Ticked" base-drum patterns for 1 training trial with a 0.6 learning rate.

intended to represent varying experience of human performers.

Another experimental condition was feature representation. The simplest representation was simply to associate the absolute time step with an action (beat/no beat). In an effort to speed learning, we also experimented with a coarse-coding representation that represented, in addition to each absolute time step, the pair-wise neighborhood of the time step. For example, in a 2-beat example, the valid states would be (1-Beat, 1-NoBeat, 2-Beat, 2-NoBeat, 1-2-Beat, 1-2-NoBeat). The intent was for these neighborhoods to gain experience twice as fast, and speed learning in early trials, whereas absolute beats would become more accurate with further experience.

For all experiments, we also performed parameter sweeps over values of the learning rate (0.1, 0.3, 0.5, 0.7, 0.9) and temperature (10, 1, 0.1, 0.01, 0.001). In sum, our results represent 125,000 data points (10 trials x 10 episodes x 5 beat patterns x 5 error models x 2 feature representations x 5 learning rates x 5 temperatures).

4.4 Results

We now present the results of our experiments.

4.4.1 Feature Representation

When we compared accuracy of learning between the baseline and neighborhood feature representations, we found several surprising outcomes. First, of the 1250 distinct trials (controlling for learning rate, temperature, and noise model), only 57 (4.56%) had a difference in accuracy between baseline and neighborhood that was greater than one standard deviation. For 37 (2.96%), there was a benefit to the neighborhood representation (i.e. accuracy was greater), whereas the remainder actually hurt learning accuracy. The beneficial cases all had very low temperatures (i.e. more greedy selection, ≤ 0.1), relatively high learning rates ($31/37 : \geq 0.5$), and most had systemic error ($> 75\%$).

Furthermore, these benefits came at the end of the training: 32 out of 37 trials (> 86%) were on trial 5 or later. These results suggest that generalization in the feature representation is of benefit when the agent has had a good deal of experience with a noisy input signal (i.e. relatively prevalent human error), while the agent is expecting a relatively clean signal and is not improvising. While not a surprising outcome in hindsight, this contradicted our intent for the representation. Since benefit was sparse, and some trials suffered, we did not make use of the neighborhood representation further.

4.4.2 *Tradeoffs in Learning and Improvisation*

We define an ideal learning and performing system as one that quickly acclimates to the "spirit" of a human performer's beat pattern, such that it is accurate both in reproduction and variation. Variations from this ideal take several forms. A *slow* learner places undue burden on human performers and a *brittle* learner cannot make progress in the presence of human error. A *wild* performer will vary produced beats in a way that is not consistent with the human performer while a *robotic* performer will only repeat the beat sequences it has perceived. Thus, in the small design space we performed, we were seeking points that were interesting along these dimensions.

As a starting point, we considered the point error model, whereby the human performer is adept for most beats, but makes a reliable, but infrequent, mistake (for reference, this error model leads to approximately 95% expected accuracy). Figures 5, 6, and 7 summarize data for this model and illustrate how interactions between learning rate and exploration policy manifest. These charts present average learning accuracy versus trials, aggregated by learning rate, for temperatures of 10 (Figure 5), 1 (Figure 6), and 0.1 (Figure 7). For intuition, large temperatures tend towards random decisions, independent of experience, while small temperatures tend towards greedy decisions, exploiting experience. Figure 5 shows how unchecked exploration will ignore learning and lead to wild performance (note that 0.5 accuracy is tantamount to random percussion). Figure 7 shows that greedy decisions (temperature $\leq$ 0.1), given this error model, will lead to identical, often robotic performance, independent of learning rate. Finally, Figure 6 illustrates how accuracy over time, given moderate exploration, depends upon the learning rate (i.e. the degree to which the learner incorporates feedback from learning, which relates to the learner's expectation of signal noise).

Given the results in Figure 6, we explored the degree to which very high learning rates (i.e. expectation of clean input signals) could adapt to systemic errors given moderate exploration rate. We found that for learning rates of 0.7 and 0.9 and all systemic-error models (10 − 30%), learning was accurate (i.e. achieved accuracy equivalent to [1-error rate]) and fast (within one trial), as compared to other rates. This provides evidence that, at least for this simple model of error, RL can quickly and accurately adapt to percussive beats while performing with moderate, non-robotic variation.

4.4.3 *Reactivity*

Across all experimental conditions, we found that Soar required no longer than 9.84 milliseconds per decision and required 9 decisions to learn a beat or produce a beat. While this data suggests a maximum of more than 170 milliseconds per time step, which is too slow for real-time use, there are mitigating factors. First, the time for most decision was 10-100x faster than this maximum. Second, the most expensive computational operation, as revealed by sampled

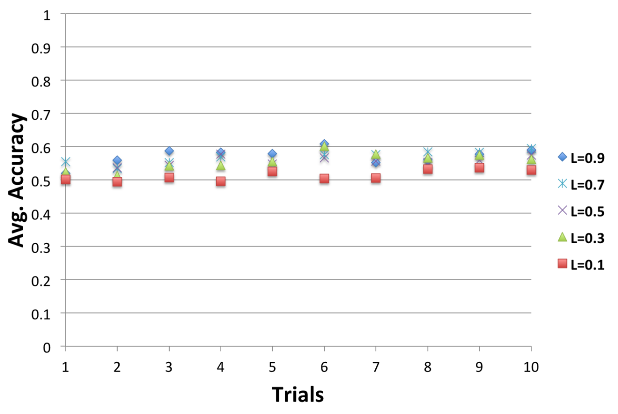

Figure 5: Point error-model learning ("wild"): temperature=10, L refers to learning rate (α).

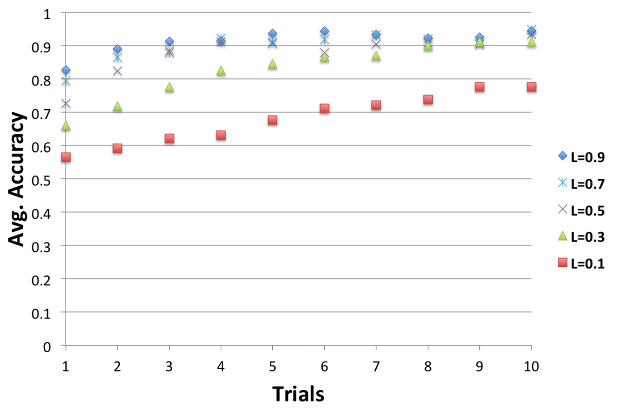

Figure 6: Point error-model learning: temperature=1, L refers to learning rate (α).

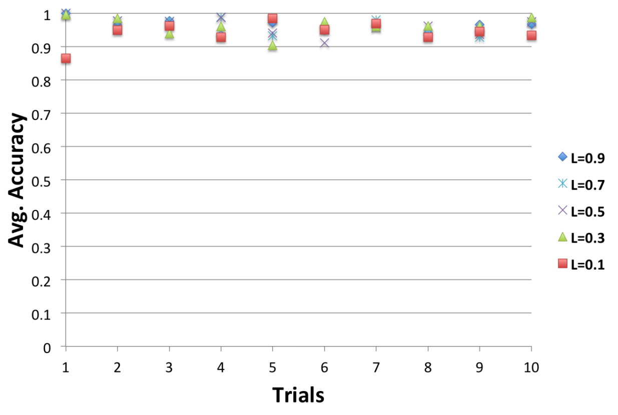

Figure 7: Point error-model learning ("robotic"): temperature=0.1, L refers to learning rate (α).

time profiling, was high-performance timers (thus, to measure reactivity, we actually reduced reactivity). Finally, in empirical usage, the system was highly responsive, allowing human performers to play, hear, and see beats in real time.

5. CONCLUSIONS

In this paper we discussed the use of reinforcement learning in online collaborative mobile drumming. The paradigm we explored is that of a mobile agent on the fly learning from a human performer to then be able to serve as a substitute, stand-in, or performance partner in a networked mobile drum circle performance. Hence we use human performers as live trainers and avoid explicit rule-based construction.

The system is implemented in urMus with Soar integration. Hence it is easy to experiment with a range of learning parameters and rule sets. We initially experimented with hierarchical rules but found them to not offer any benefits over individual bin learning. However should a different rule set form a different aesthetic it can be readily implemented without changing the core functionality of the interface. Hence we have an online learning system with good properties for experimentation and adaptation without requiring the implementation of learning algorithms in low-level languages.

We have shown that parameters natural to reinforcement learning can have useful performance interpretations. The Boltzmann temperature has a natural interpretation of improvisational freedom on top of a trained pattern. If the temperature is low a trained pattern will likely be reproduced as trained, while if the temperature increases, so does the likelihood of variation from that pattern, while respecting the given weights as starting point for the variation. Hence strongly reinforced positive or negative beat events prevail even under modest temperature, while weakly learned events will be subject to more randomness leading to an intuitive notion of how improvisation scales in our system. Additionally the user has control over the strength of learning, indicating how much the agent should trust the teaching examples provided to it.

There is much open work left. For example we have only begun to explore how to handle imprecise and changing performance by the user. Nor have we dealt with rhythmic structures that don't fall into an equally spaced bin setup. Tempo variations are possibly via the tempo control, but are not directly driven by the performers speed. While this is desirable for some kinds of drumming and sets a common temporal structure for the joint performance, it limits the stylistic expression.

6. ACKNOWLEDGMENTS

We gratefully acknowledge the help of Xin Fan with an early version of the urMus user interface code.

7. REFERENCES

[1] A. Brown. Exploring rhythmic automata. In F. Rothlauf, J. Branke, S. Cagnoni, D. Corne, R. Drechsler, Y. Jin, P. Machado, E. Marchiori, J. Romero, G. Smith, and G. Squillero, editors, *Applications of Evolutionary Computing*, volume 3449 of *Lecture Notes in Computer Science*, pages 551–556. Springer Berlin / Heidelberg, 2005.

[2] N. Derbinsky and G. Essl. Cognitive architecture in mobile music interactions. In A. R. Jensenius, A. Tveit, R. I. Godøy, and D. Overholt, editors, *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 104–107, Oslo, Norway, 2011.

[3] A. Eigenfeldt. The creation of evolutionary rhythms within a multi-agent networked drum ensemble. In *Proceedings of International Computer Music Conference*, pages 3–6, 2007.

[4] G. Essl. UrMus — An Environment for Mobile Instrument Design and Performance. *Proceedings of the International Computer Music Conference*, pages 270–273, 2010.

[5] G. Essl. Automated Ad Hoc Networking for Mobile and Hybrid Music Performance. In *Proceedings of International Computer Music Conference*, pages 399–402, 2011.

[6] G. Essl and A. Müller. Designing Mobile Musical Instruments and Environments with urMus. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 182–185, 2010.

[7] R. Fiebrink. *Real-time Human Interaction with Supervised Learning Algorithms for Music Composition and Performance.* PhD thesis, Princeton University, Princeton, NJ, USA, January 2011.

[8] A. R. J. François, E. Chew, and D. Thurmond. Visual feedback in performer-machine interaction for musical improvisation. In *Proceedings of the 7th international conference on New interfaces for musical expression*, NIME '07, pages 277–280, New York, NY, USA, 2007. ACM.

[9] G. Hoffman and G. Weinberg. Interactive improvisation with a robotic marimba player. *Auton. Robots*, 31:133–153, October 2011.

[10] Laird, J. E. *The Soar Cognitive Architecture.* MIT Press, Cambridge, 2012.

[11] J. M. Martins and E. R. Miranda. A connectionist architecture for the evolution of rhythms. In F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R. Drechsler, E. Lutton, P. Machado, J. H. Moore, J. Romero, G. D. Smith, G. Squillero, and H. Takagi, editors, *EvoWorkshops*, volume 3907 of *Lecture Notes in Computer Science*, pages 696–706. Springer, 2006.

[12] Nason, S., and J. E. Laird. Soar-RL: Integrating reinforcement learning with Soar, *Cognitive Systems Research*, vol. 6, no. 1, pp. 51–59, 2004.

[13] F. Pachet. Rhythms as emerging structures. In *Proceedings of 2000 International Computer Music Conference, Berlin, ICMA*, 2000.

[14] R. S. Sutton and A. J. Barton. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, 1998.

[15] A. Tidemann, P. Öztürk, and Y. Demiris. A groovy virtual drumming agent. In *Proceedings of the 9th International Conference on Intelligent Virtual Agents*, IVA '09, pages 104–117, Berlin, Heidelberg, 2009. Springer-Verlag.

[16] G. Weinberg and S. Driscoll. The design of a robotic marimba player: introducing pitch into robotic musicianship. In *Proceedings of the 7th international conference on New interfaces for musical expression*, NIME '07, pages 228–233, New York, NY, USA, 2007. ACM.