Interactive Rhythm Learning for Percussion Robots

Firstname Lastname Omitted for Review

Sound Computing Group

University of Anywhere

email@email.com

Introduction

Interest in musical robots and automata is as old as written history (Archimedes and Apollonius ca. 250 BC), and this interest has been steadily growing over the past few decades (Solis and Takanishi 2007) (Kapur 2005), particularly amongst practitioners of computer music. Although the manipulation of timbre has played a central role in digitally synthesized music (Wessel 1979), little work has treated the dynamic manipulation of timbre in musical robots. For this reason, the author has developed a robot called Kiki which plays djembe and can dynamically produce a range of timbres. Previous work has allowed the robot to learn to recognize timbral categories in human performance (Krzyzaniak and Paine 2015), and to match timbres played by a human. But how will the robot know in what contexts different timbres should be used? Can it learn this by listening to a human play?

The present paper proposes a solution to this question by operationalizing it as the problem of learning rhythms, where a rhythm comprises a sequence of timbral categories distributed in time. In particular, since a particular rhythm provides context for each of the timbres that comprise it, the goal is to make the robot capable of learning specific repeated rhythmic patterns, recall them according to what a human plays, and predict changes in the rhythmic patterns based on the sequence of timbres played by a human. In the absence of repeated rhythms, e.g. when the human is freely improvising, the goal is for the robot to learn the general character of the human's playing and match those features without simply mimicking the human. The method presented shall focus on challenges specific to robots with dynamic timbre-production capabilities, such as latencies introduced by striking mechanisms which may need to move a considerable distance between strokes.

Previous Work

This challenge intersects with several disciplines, and there exists relevant work in the fields of algorithmic composition, interactive computer music, musical robots, and statistical machine-learning. Popular algorithms used for composing music involve concatenating elements chosen from a list of possibilities (Kircher 1650) (Tomus II Lib VIII pars V) (Cope 2010); attempting to maintain a particular statistical distributions amongst elements (Hauer 1922) (Xenakis 1992); and attempting to meet predefined constraints (Zarlino 1968) (Hiller and Isaacson 1979) (Farbood and Schoner 2001). Although these methods are not typically interactive, computer music more generally often is (Rowe 1992) (Lewis 2000). However, in these systems the behaviour is often either hard-coded or is manually configured by the user. By contrast, similar work in the field of musical robotics attempts to learn certain aspects of the behaviour from a human interactor (Weinberg et al. 2005) (Hoffman and Weinberg 2011) (Kapur 2008). A somewhat separate line of work in machine-learning has used neural networks to learn temporal dependencies in both musical (Todd 1989) (Mozer 1999) (Eck and Schmidhuber 2002) (Boulanger-Lewandowski 2014) and nonmusical applications (Sutskever et al. 2011) (Graves 2013). However, again these methods are not interactive. The present paper extends the cited work in musical robotics by applying machine-learning techniques, and extends the cited machine-learning techniques by making them interactive.

Rhythmic Models

Components of Rhythm

A performed rhythm comprises at least three distinct temporal components: the structural component, tempo, and timing (Honing 2002). The structural rhythm may be extracted from a performed rhythm by using tempo tracking to isolate tempo (Scheirer 1998) (Goto 2001) (Percival and Tzanetakis 2014) and quantization to isolate timing (Desain and Honing 1989) (Cemgil 2004). There are also methods of doing the inverse: generating performances by applying tempo and timing to structural rhythm (Friberg et al. 2006). This study limits the model to include only the structural component of rhythm so it may be invariant to discrepancies in performance, consequently the word 'rhythm' shall be used to refer to structural rhythm henceforth.

Representations of Rhythm

In the 1940s and '50s, the composer Milton Babbitt developed two mathematical representations suitable for the proceduralization of rhythm. The first is known as a 'duration series', and the second as a 'timepoint series' (Babbitt 1962). Both deal only with note onsets which limits them to monophonic rhythms, but does not preclude the use of rests which can be conceptualized as having silent onsets. A duration series is an ordered set of integers representing relative durations, constructed by dividing an ordered set of inter-onset intervals by its greatest common divisor $\Delta\tau$. A timepoint series is an ordered set of integers representing the metric positions of note onsets. Suppose a grid is imposed on musical time, where the grid-spacing is $\Delta\tau$, and the grid positions are $i\Delta\tau \ \forall\, i \in \mathbb{N}$. Timepoints are the indices $1 + (i \mod C)$, where C is the number of timepoints in a larger structural unit such as a beat or measure. A timepoint *series* is the ordered set of those timepoints that are populated by onsets.

If a rhythmic model is to operate in realtime, it will take small chunks of input and generate small chunks of output, and the input and output will have to remain in phase with each other. It is not possible to guarantee this with a durational representation of rhythm. If, for instance, the model is given small durations as inputs and it continually outputs large durations, over time the temporal separation between input and output will be unbounded. Timepoints do not suffer this flaw as long durations can be built up in small chunks by repeated concatenations of $\emptyset$. Therefore timepoints shall be used for the current application.

Implementation

The following subsections shall first describe a basic, oversimplified model for learning rhythms, and in the subsequent subsections this basic model shall be expanded to make it suitable for real-world applications.

Basic Approach

Prediction

Let the assumption be made that the beat period and phase of the music are known, and the human's onset times are captured and quantized. The initial method presented here deals with one beat of music at a time, so C is set to the number of timepoints in a beat. Every beat of the input rhythm $\mathbb{I}$ and output rhythm $\mathbb{O}$ is encoded as a set of C real numbers. The values in $\mathbb{I}$ indicate the certainty that the human played an onset at the respective timepoint; because perfect transcription is assumed, the input values used in this study will either be 0 or 1. The values in $\mathbb{O}$ represent the probability that the robot will play an onset at the corresponding timepoint. At a time mathematically indistinguishable from the beginning of each beat (i.e. slightly before the beat, to account for computation time) $\mathbb{I}$ in the previous beat is used to calculate $\mathbb{O}$ in the subsequent

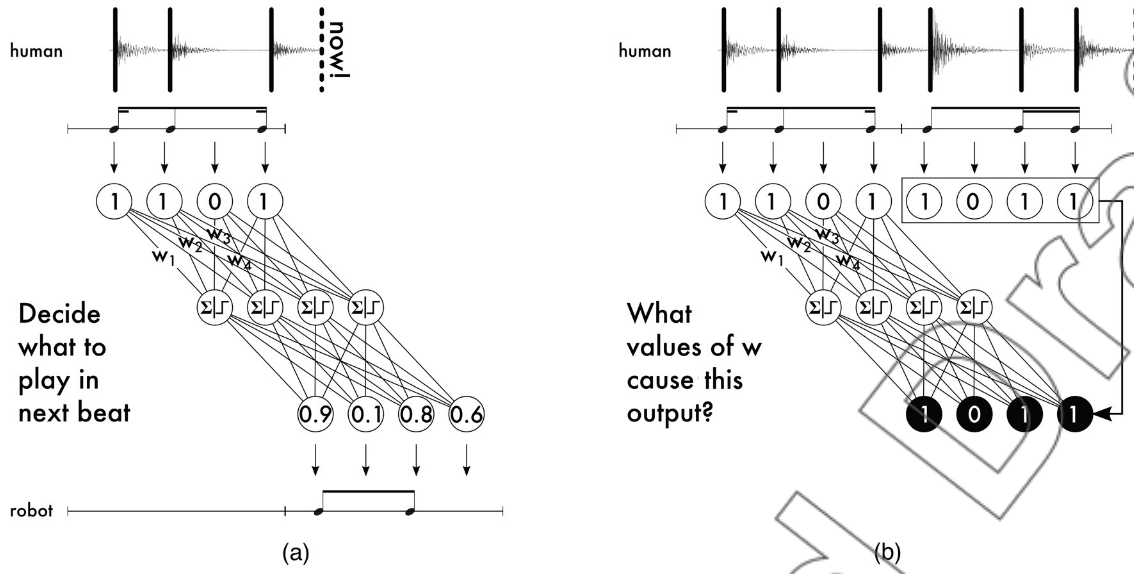

Figure 1. (a) The basic rhythm generation algorithm. At the beginning of one beat, input rhythms from the previous beat are binarized and fed into a feedforward neural network. The network outputs the next beat of the output rhythm. (b) The basic training algorithm. The network's outputs are interpreted as a prediction about the next beat of input rhythm, and at the end of that beat, the network is updated with whatever the input rhythm actually contained.

beat. The model used is a feedforward neural network with C real-valued inputs and outputs (other architectures shall be discussed below). So, at the beginning of each beat, the network's inputs are loaded with $\mathbb{I}$ and propagated through the network. The outputs are interpreted as $\mathbb{O}$. This is depicted in Figure 1 (a).

Learning

The network weights are initialized to random values. Consequently the outputs are random, so the network must be trained in order to produce meaningful output. It is desirable for the robot to be *adaptable* to musical characters that may be different between users or from moment to moment during interaction with a single user. For this reason an online learning strategy is adopted. It is possible to train a large network *offline* on a very large corpus of existing or constructed rhythms of different characters, thereby mimicking adaptability by brute force (this is further considered below). There may be certain advantages to this approach. However, an online approach is taken here for the following reasons.

1. It is a design principal of interactive robots that they should treat humans as individuals, not just generic humans (Fong et al. 2003). One way of achieving this is allowing the robot to learn different models for different humans.
2. The author considers it to be a more poetically beautiful concept for the robot to learn directly from its interactors, and recall this information during subsequent interactions. This arrangement gives the robot a kind of personal history and also allows individuals to interact with each other through the robot, and such would not be the case with a strictly offline approach.

Online learning is accomplished by interpreting $\mathbb{O}$ as a prediction about what $\mathbb{I}$ will contain in the next beat (i.e. what the human will play). After making such a prediction at the beginning of one beat, the robot waits until the end of that beat to find out what $\mathbb{I}$ actually contained. That rhythm is loaded into the network outputs, and the network weights are updated using the standard backpropagation / gradient descent algorithm, which attempts to nudge the network's weights towards values that would have caused the desired $\mathbb{O}$ for the given $\mathbb{I}$. This is depicted in Figure 1 (b).

Improvements

The basic model as presented is capable of interactively generating timepoint series. However, it makes a number of assumptions that limit its applicability to the intended purpose. These shall be rectified presently.

Timbre

A rhythm, as executed on a given percussion instrument, is rarely understood to be simply a sequence of onsets distributed in time. At the very least, a rhythm is additionally understood to be a sequence of timbral categories. Previous work has shown that it is possible to classify these timbral categories in real time, as they are played by a human percussionist (Krzyzaniak and Paine 2015)(Herrera et al. 2002). Consequently, timbre may be included in the model of the human's playing (i.e. the network inputs). Similarly, if the robot is capable of producing different timbres, then they may be modeled as well at the network output. This is accomplished by replacing each input and output neuron in the simple model with a *cluster* of neurons, each member of the cluster representing a discrete timbre. This arrangement is depicted in Figure 3.

For the inputs, only those neurons that represent the timbre that the human played at the given timepoint are set to 1. During training the input clusters are mapped to the output clusters; in the present study the human and robot will be assumed to play similar instruments such that the map is bijective.

To interpret the network's outputs, a slightly more sophisticated statistical method must be adopted to determine what the robot should play. Suppose that the network output $\mathbb{O}$ contains C clusters of neurons (one per timepoint), $\mathbb{O} = \{\mathbb{O}_1, \mathbb{O}_2 ... \mathbb{O}_C\}$. Each cluster $\mathbb{O}_c \in \mathbb{O}$ contains N neurons (each representing a timbral category), with activation levels of $\mathbb{O}_c = \{\mathbb{O}_c^1, \mathbb{O}_c^2 ... \mathbb{O}_c^N\}$. Let $\mathbb{R}$, also containing N neurons in each of C clusters, be the rhythm to be actually played by the robot, i.e. each $\mathbb{R}_c$ contains 1 as a member at most once, and all other members are 0. Each activation level $\mathbb{O}_c^n$ is interpreted as the probability that the corresponding timbral category should be included in $\mathbb{R}$, referred to henceforth as $P[\mathbb{O}_c^n]$ for simplicity, i.e. $P[\mathbb{O}_c^n] = P(\mathbb{R}_c^n = 1 | \mathbb{O}_c, n) = \mathbb{O}_c^n$.

The overall probability that some unspecified category should be played for a given cluster, referred to henceforth as $P[\mathbb{O}_c]$, is $P[\mathbb{O}_c] = P(\mathbb{R}_c \ni 1 | \mathbb{O}, c) = \max(1, \sum_{n=1}^{N} \mathbb{O}_c^n)$. In order to construct a rhythm for the robot to play that satisfies these probabilities, a decision is made about which timbral category, if any, is to be included in $\mathbb{R}_c$ from cluster $\mathbb{O}_c$ by iterating over all $\mathbb{O}_c^n \in \mathbb{O}_c$, and selecting the first unit, if any, that satisfies the criterion $\mathbb{R}_c^n = 1 \text{ if } \sum_{i=1}^{n} \mathbb{O}_c^i \geq r_c$, where $0 < r_c \leq 1$ is a evenly distributed random number chosen independently for each $\mathbb{O}_c \in \mathbb{O}$.

This model assumes that the network has been trained. Prior to training, the expected value of each neuron cluster is likely to be greater than 1, depending on which activation functions are used and how the weights were initialized. If the standard practice is used, whereby weights and rest states are initialized to random numbers with zero mean, and the logistic sigmoid $\frac{1}{1+e^{-x}}$ is the output-layer activation function, the expected value of each output cluster will be about $0.5N$ (depending on the hidden-layer activation function), and again the network will generate too many notes. If an online learning strategy is adopted, the network should be pre-trained to generate zero at all output neurons for all input vectors. This may be accomplished by feeding the network random input vectors and training with a target output of all zeros, until it has 'learned' to output all zeros, according to the definition given in Equation 1.

Activation Function

Since valid values for output neurons lie between 0 and 1, it has thusfar been assumed, according to standard practice, that a logistic sigmoid function would be used for the output-layer activation. Notice, however, that this makes it impossible for the network to indicate that with 100% likelihood the robot should play *nothing* on a given timepoint. This is exacerbated by the presence of multiple neurons per cluster, because the neuron activations are summed, thereby increasing the probability of a spurious onset. Worse, if there are many timepoints per beat, this situation will occur frequently; even if the summed probability is small for a given timepoint, spurious onsets will be likely over time. So an activation function should be chosen which can explicitly represent 0. One option is a 'truncated sigmoid', which is $\frac{1}{1+e^{-x}}$ when x is greater than some threshold, and 0 otherwise. This was found to significantly outperform the logistic sigmoid on all experiments presented in the Evaluation section of this paper.

Timing

Predicting each beat at the very beginning of that beat assumes that the robot can strike the drum instantaneously if an onset is generated in the first timepoint of the predicted beat. A human cannot strike a drum at the precise moment they decide to do so, and it might not be realistic to impose that constraint on a robot. In other words, the robot has latency, which must be dealt with. For a system with a single human and robotic drummer, let there be defined three representations of the present – t_h, t_r, and t_c. These are, respectively, the present as defined by the human, the robot's striking mechanism, and a computer which is receiving the human's input and sending commands to the robot's striker. From the human's perspective, these are all translated with respect to one-another. t_r lags t_c by some amount l_{cr} which represents the entire latency between the computer sending a command and the the sound of robot striking the drum reaching the human. The robot should typically be designed in such a way that l_{cr} is constant. Furthermore, because a computer cannot act upon information until it has received and processed it, t_c lags t_h by at least l_{hc_min}. For the case at hand, in which the computer is receiving rhythms via a microphone, l_{hc_min} must represent a worst-case scenario estimate that includes maximum buffering uncertainty, analysis latency (including the duration of audio required to perform stroke-classification) and, if rhythmic quantization is performed, the maximum unit of time by which a note could be quantized backwards to end up on the first timepoint of the beat (i.e. half of the grid spacing for grid-based quantizers). Additionally, the computer may deliberately introduce some arbitrary padding latency, so that ultimately t_c lags t_h by a total amount l_{hc}. If t_h must be in phase with t_r (i.e. translated by an integer number of beats), padding must be introduced such that l_{hc} is the total amount of time needed between the human striking the drum and the computer sending a command to the robot, such that the robot's stroke falls on the desired timepoint of a future beat, as heard by the human. It would be given by $l_{hc} = \Delta t(\text{ceil}(\frac{l_{cr}+l_{hc\_min}}{\Delta t}) + B) - l_{cr}$, where Δt is the beat duration, B is a nonnegative integer constant, and ceil(x) is the ceiling function which returns the smallest integer not less than x. Using $B = 0$ results in the minimum requisite value for l_{hc}, and increasing B specifies additional whole beats of delay.

Given a particular network topology and training strategy, the total number of beats separating the human from the robot $\beta = \frac{l_{hc}+l_{cr}}{\Delta t}$ should not spontaneously change values according to the beat duration Δt, but must be constant for the life of the network. This can be accomplished by choosing a desired $\beta >= \text{ceil}(\frac{l_{cr}+l_{hc\_min}}{\Delta t})$, and defining a minimum allowable beat duration, Δt_{min}, such that $\beta \Delta t_{min} >= l_{cr}$ Then, the appropriate lag is simply $l_{hc} = \beta \Delta t - l_{cr}$.

A training scheme that accounts for these multiple representations of the present is depicted in Figure 2 and Figure 3. From the computer's perspective, the network still predicts what the robot should play in the immediate future, and learns from the immediate past. However, from the human's perspective, rather than predicting the robot's rhythm in the very next beat, the network predicts the rhythm β beats in the future. Then, $\beta + 1$ beats later, the network is updated with what the human actually played in that beat. This necessitates three distinct steps at each training cycle. One forward pass through the network is made to predict the robot's output. Then, in order to put the network in the correct state for training, another forward pass is made using the beat that ended β beats previously as input. Subsequently, a training pass is made over the network, using the beat that just ended as the target output.

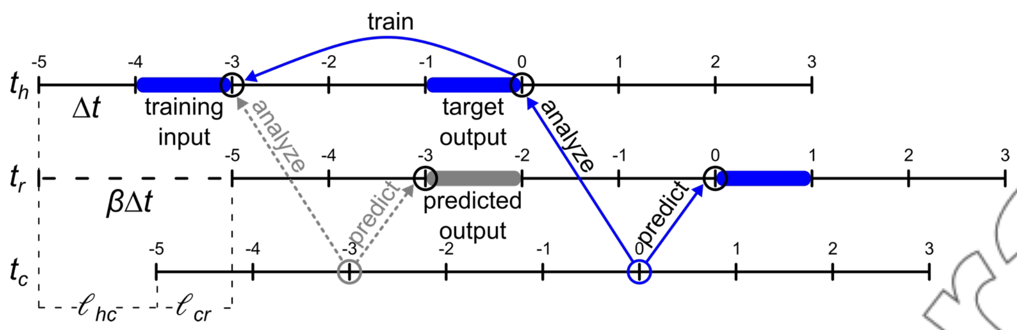

Figure 2. Training algorithm that incorporates system latencies, as run at time $t = 0$. From the human's perspective, at the beginning of each beat, the computer trains the network by comparing the what the human just played to what the network predicted she would play. Then the computer predicts what the human will play β beats later.

Long-term Dependencies

One very obvious and grave problem with the model presented thus far is that it only models temporal dependencies one beat in duration; it can only learn longer rhythms where each beat in the rhythm is unique. Formally, the problem is that feed-forward networks model *functions*, which map input to output, whereas rhythms might more properly be discrete *dynamical systems*, in which the current state (i.e. beat), and consequently the subsequent state, is determined by *all past states*. As stated in the Previous Work section, Recurrent Neural Networks (RNNs) are a family of very deep network topologies that model discrete dynamical systems with long-term temporal dependencies, which have been used to generate music offline and non-interactively. Although recent advances have made them very powerful, they are not well suited to the task at hand. In particular, the timing solution present in the *Timing* section above means that rhythm generation in this context is not *causal*. In other words, it is not possible, at each step, to train the current state before putting the network into the next state. Instead, because of the time-delay, training always occurs on a network state that is a few cycles in the past. Of course it is in principle possible to save and subsequently restore past states for training, but by that time predictions have already been made based on future states, so reality effectively bifurcates at each training cycle, and it is not

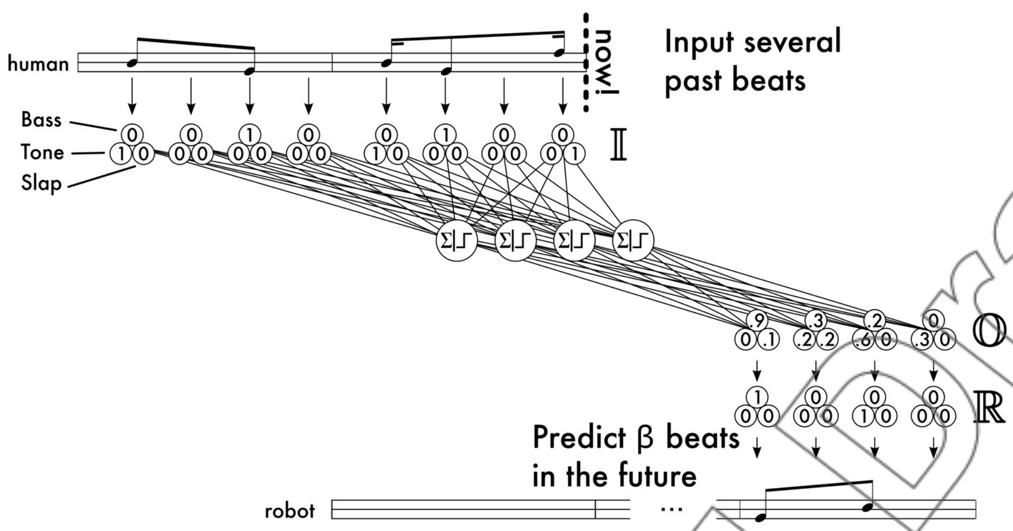

Figure 3. The improved algorithm. Each input and output neuron has been replaced by a cluster of neurons to represent discrete timbral categories. (NB – layers are fully connected although some connections have been omitted here). Time delay is handled by predicting several beats in the future. Longer-range dependencies are modeled by including more of the past rhythm as network input.

clear which branch to follow. It may or may not be possible to design a non-causal dynamical neural network, but further investigation shall be left for future work. So in order to expand the temporal range of the network, the feedforward model is upheld, but more input neurons are included to encompass several past beats. At each training cycle (the beginning of each beat), the input vector is shifted one beat to the left, and the rightmost input neurons are populated with the most recently concluded beat. This entire vector is propagated through the network to produce the next beat of output. This is shewn in Figure 3. Training occurs as before.

Evaluation

A set of experiments were performed to assess the network's ability to learn both specific rhythms and the musical character of improvisation. These experiments were carried out using two types of methodology. On type consists of strictly numerical analysis, whereby encoded rhythms were fed into the network offline, and it's outputs were analyzed statistically. The other type involves the realtime input and output of sound. In this latter type, contact-mic recordings were made of the core djembe strokes, and audio-editing software was used to arrange these into audio files containing metronomically precise rhythms. These recordings were played into the realtime onset-detector and stroke-classifier described in (Krzyzaniak and Paine 2015), which was trained to classify these particular strokes with 100% accuracy. Onset times were ascertained by examining the audio-buffer timestamps and counting samples into the buffer to where the onset occurred. The first two strokes in the constructed file were used to define the tempo, and subsequent strokes were quantized using a simple grid-based quantizer. The quantized strokes were then fed into the network in real-time, one beat at at time. The network outputs were either sent to the robot to be played in realtime, one timepoint at at time, or to a software simulation of the robot which delays incoming messages by l_{cr} before playing a recording of the stroke. These two types of methodology are functionally identical. This software can receive live input from a human just as easily as a constructed audio file. However, in the absence of robust tempo-following and quantization, the software is rather inaccurate at transcribing human performance, which makes it difficult to isolate the network's behaviour. Because accurate transcription was not a goal of this study, assessment with live human input was not performed.

Network Topology

In all of the following studies, the network had three stroke categories in both the input and output rhythms (understood to be bass, tone, and slap), twelve timepoints per beat, 6 input beats (i.e. 216 individual input neurons), 1 output beat (72 neurons), and a single hidden layer with 9 individual neurons. The output layer used a truncated sigmoid activation function as described in the *Activation Function* subsection, with threshold -4, and the hidden layer used softplus ($\ln(1 + e^x)$). Output is predicted two beats in advance, i.e. $\beta = 2$. In each experiment, the only model parameter, the learning-rate ℓ, was chosen by trial and error to be approximately optimal for the given task.

The network shall be said to have 'learned' a rhythm when the robot is 95% likely to play that rhythm in response to a particular input. This is defined as follows. Let $\mathbb{O}_c^n$ be the activation level of the n^{th} neuron in the c^{th} cluster in the output rhythm, let $\mathbb{R}_c^n$ be a binary value representing whether the robot actually plays the n^{th} timbre at timepoint c, and $\mathbb{T}_c^n$ is the same in the target rhythm. The probability $P(\mathbb{R} = \mathbb{T}|\mathbb{O})$ that the robot will actually play the target rhythm is defined to be

$$P(\mathbb{R} = \mathbb{T}|\mathbb{O}) = \prod_{c=1}^{C} \begin{cases} P[\mathbb{O}_c^{\mathrm{ix}(1 \in \mathbb{T}_c)}] & \text{if } \mathbb{T}_c \ni 1 \\ 1 - P[\mathbb{O}_c] & \text{otherwise} \end{cases} \qquad (1)$$

where $\mathrm{ix}(1 \in \mathbb{T}_c)$ is the index of 1 in $\mathbb{T}_c$. Notice that if $\mathbb{T}$ is several beats in duration, the product will have to be accumulated over several training cycles.

Learning Specific Rhythms

The first set of tasks involve learning specific rhythms that are played repetitively by the human. The goal of this is to ascertain the degree to which the network can learn when it is very precisely appropriate to play specific timbres. The dataset used in these tasks comprised all of the repeated djembe rhythms in (Billmeier and Keïta 1999) (i.e. the parts labelled 'djembe 1', 'djembe 2', etc ..., but not 'signal' or 'introduction' or parts for other instruments). At times, the djembe serves an accompaniment role, so certain common accompaniment patterns appear in several pieces; such rhythms were only included once. This resulted in 66 unique rhythms ranging in length from 1 to 16 beats (mean 3.79), and containing a mixture of simple and compound meter (as written). Beat groupings were used as given by the beaming in the text, although it is worth noting

Table 1. Learning Specific Rhythms

Experiment	Unit of Result	Result
Individual Rhythms	Number of repetitions required to learn a rhythm	14.32
Corpus of Rhythms	Number of repetitions required to learn and retain 65 rhythms	106.5
Signal Rhythm	Probability of playing correct rhythm following a signal	96%

that the Western conception of 'beat' is not entirely applicable to West African music, and this represents only one of several possible encodings of the rhythms; another encoding might involve keeping the duration eighth-notes (quavers), instead of the beat, constant across rhythms. Some rhythms were written with superfluous repetitions, e.g. a 2-beat rhythm written twice to fill four beats; the superfluous beats were discarded. The few instances of unusual strokes in the data were replaced with one of the core three. The network was tested with this dataset on three tasks, to be presently discussed; results are summarized in Table 1.

Individual Rhythms

In the author's experience playing in community drumming ensembles in the United States, a common musical piece consists of most members playing a single accompaniment rhythm in unison, repeated ad infinitum, while a select few members play counter-rhythms or improvised solos. Several such pieces may be played during a single rehearsal or performance. Can the robot learn to play the accompaniment rhythms in this scenario? Can it do so in a reasonable length of time? Assessment of this task was performed by initializing the network weights, and repeatedly feeding a rhythm from the dataset into the network one beat at a time. Here, the rhythms are taken out of context and assumed to have been repeated infinitely, so even during the first training cycle, inputs representing past beats were populated with previous repetitions of the rhythm – i.e. each rhythm was *rotated* through the network. Then, learning duration was measured by counting the number of times the rhythm was rotated through the network in its entirety before it had been learned. This entire process, starting with initializing the network, was repeated for every rhythm in the dataset ($N = 66$) and the results were averaged across rhythms. That entire process was repeated 10 times and the averages were averaged. With $\ell = 0.65$, the mean learning duration was 14.32 repetitions of the rhythm ($N = 10, \sigma = 1.52$). In the cited scenario, the network weights would have to be manually initialized between pieces to achieve these results. If the weights are not initialized in between pieces, it takes a little longer to learn each rhythm. In a test that initialized the network before the first rhythm but not subsequently between rhythms, using $\ell = 0.5$, the average duration was measured to be 21.42 repetitions of the rhythm ($N = 10, \sigma = 2.760955$). On the one hand, this is very feasible, as it takes only 42 seconds to repeat a 4-beat rhythm 21 times at 120 beats per minute. On the other hand it may be tedious to repeat a rhythm 21 times to a robot. Note however that the definition of 'having learned' a rhythm is relatively stringent; the network is capable of representing 7.9×10^{28} unique 4-beat rhythms, and it is thus extremely unlikely that the network will output the correct rhythm by chance alone, yet the given definition specifies that such will happen 19 times out of 20. A more relaxed definition would be that the network has 'learned' a rhythm after each neuron is 95% likely to output its target value. Repeating the the last experiment with this relaxed definition and $\ell = 0.3$ yields a mean of 9.87 repetitions of the rhythm ($N = 10, \sigma = 0.68$). With such a definition, it is not likely that the network will output the correct rhythm precisely, but it is significantly more likely than random to get each timepoint correct, which should produce something that resembles the rhythm. In other words, after hearing a new rhythm only a few times, the network should sound like it is starting to get the gist.

In many musical situations, it may be appropriate for the robot not to play the same rhythm as the human, but to always accompany rhythm a with rhythm b. In principal, two human percussionists could train the robot to exhibit this behaviour, if one human's rhythm was used as the network input and the other human's as the target output.

Corpus of Rhythms

This shows that a previously-untrained network can learn individual rhythms, but can it also retain rhythms it has learned? In a more standard machine-learning scenario the network would be pre-trained with a corpus of rhythms prior to human interaction, either by offline training or by previous online interaction with humans. Can the network learn a corpus of rhythms? The same 66 rhythms as above were used to test this. Here, after initializing the network and training it to output zeros, each rhythm was rotated through the network in its entirety once, one after the next, round robin, until all rhythms had been learned. Indeed, on average, the network learned all of the rhythms after hearing each rhythm 305.50 times ($\ell = 0.15$, $N = 10, \sigma = 158.55$). Longer rhythms take longer to learn, according to Equation 1; in the current dataset one outlier rhythm was twice as long as any other, and this one always took much longer to learn than the others. Excluding it from the dataset resulted in a mean of 106.50 repetitions of each rhythm ($N = 10, \sigma = 11.38$). The convergence rate, in general, could probably be improved using standard techniques (annealing, scrambling the order on each cycle, Newton's method, batch learning, etc...), although here the goal was to demonstrate that the network is capable of learning when to produce certain timbres across a large variety of contexts – offline batch learning with maximum efficiency was not the primary goal.

If the only goal were to learn specific rhythms, it would of course be faster and more precise to simply transcribe what the human plays, hash it, and store it in a database. Some advantages of the method presented here are as follows: this method is agnostic to the length of rhythmic patterns so it is not necessary to explicitly segment the pattern boundaries; neural networks tend to be impervious to small variations in input, as may arise as the result of transcription error or creative variation; fuzziness at the output may be desirable to alleviate monotony – because a trained network contains a statistical model of the human's playing, this fuzziness may be accepted as musically interesting variation.

Signal Rhythms

Another task not suitable for database lookup is predicting changes in the music according to context. Often in the rhythmic music of Africa and Latin America, special rhythms are used to signal musical changes. A good example comes from a pair of rhythms in the dataset, called Yankadi and Makru. The basic Yankadi rhythm is repeated until the drum leader plays a signal rhythm (either on a whistle or a drum) which signals the switch to Makru. Then the drummers repeat the Makru rhythm until the leader plays the signal again, which prompts the transition back to Yankadi, etcetra. So, the signal rhythm could be followed by either Yankadi or Makru, depending on the context.

Can the network learn and retain the ability to predict what the human will play following the signal? This was tested as follows. A numerical sequence was constructed comprising 3 repetitions of Yankadi, followed by the signal, followed by 3 repetitions of Makru, followed by the signal again. Three repetitions are the minimum such that the network cannot learn this sequence as a single rhythm, i.e. every six beat window is not unique; in other words, the network can not know in advance whether the human plans on playing the signal or another repetition of the current rhythm. This rhythm was used to train the network with $\ell = 0.1$ until the least accurate output neuron over the whole sequence was at least 40% accurate, (roughly meaning that it was about 40% accurate in guessing whether the human would branch to the signal or continue repeating the rhythm). Then, an audio recording was constructed which contained an arbitrary number of repetitions of each rhythm (always 8), each followed by the signal. This audio file was fed into the network and conceptually paused after the signal, at which time the network had already predicted the next two beats. The probability that the predicted two beats would belong to the correct rhythm was calculated again according to Equation 1. This process was repeated 20 times (10 transitions to each of Yankadi and

Table 2. *Learning Improvisation*

Experiment	Unit of Result	Result
Note Density	Correlation and slope of regression line for in vs. out note density	$r = 0.977;\ m = 1.0$
Meter	Probability of playing on an incorrect timepoint	0.39%
Syncopicity	Correlation and slope of regression line of input vs. output syncopicity	$r = 0.96;\ m = 0.80$

Makru). On average, the probability that the network would output the first two beats of the correct rhythm was 0.96 ($N = 20, \sigma = 0.029$). During this test, the network continued to learn online from the input sequence with a rate of 0.1. As learning goes hand-in-hand with forgetting previous information, one might predict that the extra repetitions of Yankadi and Makru in the test sequence would have caused the network to forget the meaning of the signal in the meanwhile. On the contrary, no significant change in accuracy was observed over time (it actually increased marginally), even though this task updated the network weights 720 times during testing (there were that many beats in the input sequence).

Learning Improvisation

The preceding section treats the network's ability to learn specific, repeated rhythms. In a different scenario, a human may improvise rhythms with no explicit structural repetition. In that case, the robot would not be expected to play in unison with the human, but should instead either match or deliberately oppose salient characteristics of the human's music. Three tasks were used to assess the network's ability to do this, as will be presently discussed, and the results are summarized in Table 2.

Note Density

One such character is the note density, d, of the music, which may be defined here as the number of onsets divided by the number of timepoints in a given span of music, or, if rhythms are statistical, $d = \sum_{c=1}^{C} P[\mathbb{O}_c]/C$. Very sparse rhythms may have a different aesthetic character than very dense rhythms, and in many cases it would be appropriate for the robot to match the human's d. Consider a scenario wherein the human improvises a rhythm with a particular character and corresponding d for a while, and after some time begins improvising with a new character and corresponding d. Can the robot adapt to this change? This was tested as follows. First, a very long rhythmic training sequence was constructed comprising 6000 beats. For each 120-beat segment (representing one minute of playing at 120 BPM), a random d was chosen, and a random rhythm was generated in that segment by, with a probability of d, populating each timepoint with an onset in a random timbral category. The resulting sequence then contained 50 such concatenated segments. To train the network, the entire sequence was shifted through the network, to simulate previous interaction with a human. Then a new 6000-beat validation sequence was constructed by the same means, and shifted through the network. At each beat of the validation sequence, the observed rhythmic density of the network input (previous 6 beats) and predicted output (one beat) were recorded and subsequently correlated. The input density and output density were found to be linearly correlated with $r = 0.977$, and the regression line had a slope of 1, meaning that the network is very good at matching the note density of improvised rhythms. In certain scenarios, it might be desirable for a robot to oppose the human's d rather than mimic it, such as when a foreground musical role in one part should be paired with a background role in the other part; the described method does not present an obvious way of achieving this.

Meter

Another salient characteristic of rhythm is its *meter*. In particular, in most cases, the human's decision to play in either simple or compound meter should be matched by the robot. Simple meter is characterized by a binary or quaternary subdivision of the beat with onsets on the 1^{st}, 4^{th}, 7^{th}, 10^{th} timepoints (i.e. 16th notes or semiquavers) of any beat, whereas compound meter uses a tripartite subdivision with onsets occurring on the 1^{st}, 5^{th}, and 9^{th} timepoints (triplets). These shall be referred to as the 'valid' timepoints for the corresponding meters. The network's ability to match the input meter was tested as follows. A training set of 1000 6-beat rhythms were constructed for each meter, by randomly populating the respective valid onsets with an average note density of 1 onset per beat. Each rhythm was rotated through the network once, alternating meters. Then a validation set of 100 6-beat rhythms in each category were similarly constructed and rotated through the network. Because meter is defined by those timepoints where onsets should *not* occur, the probability of playing on at least one invalid timepoint was measured for each beat of output, and averaged across all beats. On average, the network produced output on an invalid timepoint with a probability of 0.0039 ($N = 1200, \sigma = 0.029$). Although this is reasonably low, notice that it took a very large number of training cycles to achieve. With much less training, the network will begin producing invalid onsets with a probability of exactly 0 on the majority of beats, but will spuriously produce a relatively high probability, distributed in small quantities across all invalid neurons, on a handful of beats. More training reduces the frequency and value of these spuriously high probabilities, but does not seem to eliminate them. So although the network can match meter given enough training, it does have trouble producing output of exactly 0 on specific timepoints.

Syncopicity

Yet another salient characteristic of rhythm is syncopicity, s, which is a measure of how syncopated a rhythm is. This experiment considers only the 1^{st}, 4^{th}, 7^{th}, 10^{th} timepoints of any beat, referred to collectively as the 'valid' timepoints, whereas the 4^{th} and 10^{th} specifically are considered the syncopated timepoints. Here s shall refer to the rhythmic density defined over the syncopated timepoints divided by the rhythmic density defined over the valid timepoints for a given span of rhythm. Random sequences with given s were constructed by, with probability s, populating syncopated timepoints with an onset in a random category, and doing so with a probability of $1 - s$ in other valid timepoints, resulting in an overall note-density of 0.5. The network's ability to match syncopicity was tested as for note density; i.e. a 6000 beat training sequence followed by a 6000 beat validation example, with 120-beat segments with random r. Output syncopicity was 96% linearly correlated with input syncopicity. The slope of the regression line was 0.80 indicating that although the network was very good at matching s, it has some trouble producing extreme values of s. This is consistent with the results of meter test above, which suggests the network has some trouble outputting exactly 0 on specific timepoints.

Future Work

In the future, the work presented in this paper should be tested with human interactors. How does this method compare to playing with another human, or alone, or to a robot that generates random rhythms, non-interactively? Does this method feel responsive? If users had a percussion robot employing this algorithm at home, would it encourage them to play longer or more frequently than they would otherwise? Will fuzziness at the robot's output be interpreted as creativity or mistake?

References

Archimedes, and Apollonius. ca. 250 BC. *Kitab Arshimidas fi al-binkamat and San'at al-zamir*. British Library: Oriental Manuscripts, Add MS 23391. in the Qatar Digital Library. URL http://www.qdl.qa/en/archive/81055/vdc_100022555293.0x000001.

Babbitt, M. 1962. "Twelve-tone rhythmic structure and the electronic medium." *Perspectives of New Music* :49–79.

Billmeier, U., and M. Keïta. 1999. *Mamady Keïta : a life for the djembe : traditional rhythms of the Malinke*. Arun.

Boulanger-Lewandowski, N. 2014. “Modeling High-Dimensional Audio Sequences with Recurrent Neural Networks.” Ph.D. thesis, "Universit’e de Montr’al".

Cemgil, T. 2004. “Bayesian Music Transcription.” Ph.D. thesis, Ph. D. Netherlands: Radboud University of Nijmegen.

Cope, D. H. 2010. “Recombinant music composition algorithm and method of using the same.” US Patent 7,696,426.

Desain, P., and H. Honing. 1989. “The quantization of musical time: A connectionist approach.” *Computer Music Journal* :56–66.

Eck, D., and J. Schmidhuber. 2002. “A first look at music composition using LSTM recurrent neural networks.” *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale* .

Farbood, M., and B. Schoner. 2001. “Analysis and synthesis of Palestrina-style counterpoint using Markov chains.” In *Proceedings of the International Computer Music Conference*. pp. 471–474.

Fong, T., I. Nourbakhsh, and K. Dautenhahn. 2003. “A survey of socially interactive robots.” *Robotics and autonomous systems* 42(3):143–166.

Friberg, A., R. Bresin, and J. Sundberg. 2006. “Overview of the KTH rule system for musical performance.” *Advances in Cognitive Psychology* 2(2-3):145–161.

Goto, M. 2001. “An audio-based real-time beat tracking system for music with or without drum-sounds.” *Journal of New Music Research* 30(2):159–171.

Graves, A. 2013. “Generating sequences with recurrent neural networks.” *arXiv preprint arXiv:1308.0850* .

Hauer, J. 1922. “Sphaerenmusik.” *Melos* 3:132–133.

Herrera, P., A. Yeterian, and F. Gouyon. 2002. "Automatic classification of drum sounds: a comparison of feature selection methods and classification techniques." In *Music and Artificial Intelligence*. Springer, pp. 69–80.

Hiller, L. A., and L. M. Isaacson. 1979. *Experimental Music; Composition with an electronic computer*. Greenwood Publishing Group Inc.

Hoffman, G., and G. Weinberg. 2011. "Interactive improvisation with a robotic marimba player." *Autonomous Robots* 31(2-3):133–153.

Honing, H. 2002. "Structure and interpretation of rhythm and timing." *Tijdschrift voor Muziektheorie* 7(3):227–232.

Kapur, A. 2005. "A history of robotic musical instruments." In *Proceedings of the International Computer Music Conference*. Citeseer, pp. 21–28.

Kapur, A. 2008. "Digitizing North Indian Music: Preservation and Extension using Multimodal SensorSystems, Machine Learning and Robotics." Ph.D. thesis, University of Victoria.

Kircher, A. 1650. "Musurgia universalis sive Ars magna consoni et dissoni." *Rome: Corbeletto* URL http://imslp.org/wiki/Musurgia_Universalis_(Kircher,_Athanasius).

Krzyzaniak, M., and G. Paine. 2015. "Realtime Classification of Hand-Drum Strokes." In *International Conference on New Interfaces for Musical Expression*. pp. 400–403.

Lewis, G. E. 2000. "Too many notes: Computers, complexity and culture in Voyager." *Leonardo Music Journal* 10:33–39.

Mozer, M. C. 1999. "Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multiscale processing." *Musical Networks: Parallel Distributed Perception and Performance* 227.

Percival, G., and G. Tzanetakis. 2014. "Streamlined Tempo Estimation based on Autocorrelation and Cross-correlation with Pulses." *IEEE/ACM Transactions on Audio, Speech, and Language Processing* .

Rowe, R. 1992. "Machine listening and composing with cypher." *Computer Music Journal* :43–63.

Scheirer, E. D. 1998. "Tempo and beat analysis of acoustic musical signals." *J Acoust Soc Am* 103(1):588–601.

Solis, J., and A. Takanishi. 2007. "An overview of the research approaches on musical performance robots." In *International Conference on Computer Music*. pp. 356–359.

Sutskever, I., J. Martens, and G. E. Hinton. 2011. "Generating text with recurrent neural networks." In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. pp. 1017–1024.

Todd, P. M. 1989. "A connectionist approach to algorithmic composition." *Computer Music Journal* :27–43.

Weinberg, G., S. Driscoll, and M. Parry. 2005. "HAILE–AN INTERACTIVE ROBOTIC PERCUSSIONIST." *Ann Arbor: Scholarly Publishing Office, University of Michigan Library* .

Wessel, D. L. 1979. "Timbre space as a musical control structure." *Computer music journal* :45–52.

Xenakis, I. 1992. *Formalized music: thought and mathematics in composition*. Number 6 in Harmonologia Series. Pendragon Press.

Zarlino, G. 1968. *The Art of Counterpoint: Part Three of Le Istitutioni Harmoniche, 1558*. Yale University Press.