PLAY-ALONG MAPPING OF MUSICAL CONTROLLERS

Rebecca Fiebrink
Princeton University
Computer Science
rfiebrink@acm.org

Perry R. Cook
Princeton University
Computer Science, also Music
prc@cs.princeton.edu

Dan Trueman
Princeton University
Music
dtrueman@princeton.edu

ABSTRACT

We describe our tool for interactively creating musical controller mappings using a "play-along" paradigm, in which a user pretends to play along with a musical score in real-time using an arbitrary input control modality. As the user "performs," a supervised machine learning system builds a training dataset from the user's gestures and the synthesis engine's current parameters. After one or more training stages, the algorithm learns a mapping from user inputs to synthesis parameters. Control is transferred to the user, who can then control the synthesis in real-time.

1. INTRODUCTION

Many of us recall going to video arcades as children and, our money gone, forlornly moving the joystick along with our favorite game's attract mode as if we were really playing. What if a computer recognized such attempts to control it, and actually gave over control to the user? And what if, instead of a quarter, it required that the user bring his own joystick (or webcam, or sensor glove, or any other input method) and gesture along until it became clear how the user intended to control the game?

It is in this spirit that we propose a "play-along" approach to creating controller mappings for sound synthesis and other interactive purposes. We have previously built a system for using supervised machine learning algorithms to learn the relationship between generic controller inputs and synthesis engine parameters, which allows users to interactively adjust the parameters of the learning process and add new mapping examples in real-time, even during a performance. Here, we discuss our new play-along method for creating mapping datasets in real-time, with the goal of improving efficiency and fun. In this approach, the training dataset is constructed implicitly while the user pretends to play along to a musical score using a controller.

2. BACKGROUND AND MOTIVATION

Machine learning (ML) methods are an established technique in the creation of new sound and music interfaces. In particular, supervised ML methods have been often used to generate a model of the mapping from controller inputs to synthesis outputs, using training datasets consisting of "true" input/output pairs. Neural networks (NNs) have been particularly popular for musical control since the 1990's, when Lee et al. [7] and Fels and Hinton [3] employed them for mappings in music and speech synthesis, respectively. While NNs learn a continuous function mapping, an array of discrete classifier algorithms are suitable for mapping the inputs into a discrete set of output classes (e.g., selecting a sound event based on the category of an input gesture).

With few exceptions, ML-based systems for constructing musical controller mappings have treated the training set creation, model training, and use of the trained models (i.e., performance using the mapped controller) as three separate activities. A user performs each activity with a different user interface, for example, and only the third activity is executed in real-time while making sound. One exception is the FlexiGesture input controller of Merrill and Paradiso [8], which allowed users to iteratively present example controller gestures and control the sound using the resulting mapping. They used dynamic time warping to match controller gestures to sonic gestures, despite the richer mapping functions that ML methods might allow, citing the need for "significant amounts of training data" for successful learning.

A small amount of training data, however, does not preclude successful application of ML methods if the learning problem is narrow in scope, if the benefits of the associated decrease in training time outweigh the possible loss of accuracy, and if success is defined by factors other than objective accuracy. Fails and Olsen [2] propose an interactive approach to supervised learning for image

labeling tasks that is contingent on the first two criteria. In their system, users are able to improve the learning outcome by iteratively training, evaluating the model's performance, creating more training data points to correct algorithm errors, and re-training. Here, fast training time is essential to tight human-computer interaction, and focus on a limited learning task (labeling of a small set of objects) enables both acceptable speed and accuracy. Our approach to "on-the-fly" musical classification proposed in [5] is also concerned with relatively limited tasks (vowel/consonant recognition, or discrimination between two artists) such that training can happen in real-time during a performance.

In our recent work [4], we have begun to explore the potential applications for highly interactive, creative ML for music, where the user engages with all stages of the learning process in order to achieve an outcome where success may be quite subjective and personal. One such application is controller mapping creation, where the primary goal of the user may not be to teach the computer an ideal, preconceived mapping, but rather to create a mapping that allows for new musical expression and exploration. To this end, we have previously constructed a system, called the Wekinator, which presents a single GUI for real-time interaction with all stages of the learning process. Users demonstrate pairs of controller inputs (e.g., a joystick position) and "true" synthesis parameter outputs in real-time to create and augment a training dataset, train a NN or classifier to learn the mapping in a few seconds, and run the trained algorithm to allow the user to play using the new mapping. Our focus in building the Wekinator was on the support of easy user interaction with all aspects of the learning process, in order to better facilitate rapid exploration of synthesis algorithms and controller mappings, in compositional or even real-time performance contexts.

In our experience, the relatively small number of training examples created during real-time interaction with the Wekinator, even during a short performance, is adequate for the somewhat subjective task of constructing controller mappings. Additionally, the many points of interaction with the real-time learning process allow the user to shape the learning outcome in appropriate and musically compelling ways. However, even though the training takes place in real-time, the creation of training examples is a data entry task as much as a musical one: the user/performer repeatedly enters the desired output values of each synthesis parameter into the Wekinator GUI, then demonstrates with the input controller a gesture or position that should induce a trained model to generate those outputs. Training set creation for a controller bears little resemblance to performing with the controller, adding very many training examples with widely differing parameters can become slow and tedious, and the user's attention is focused on adequately sampling a parameter space that may not be well understood rather than on the *sound* as he will be controlling it. By addressing these problems, we believe the real-time training set creation process can be even faster, more efficient, and more compelling for both the user and, in the case of live performance, the audience.

3. PLAY-ALONG MAPPING

3.1. Definition

In play-along mapping as we define it, the training set is generated automatically as the computer autonomously performs a musical "score" and the human gestures along with her chosen controller(s) as if she were controlling the sounds she hears. The training set creation therefore resembles the musical task of expressive control over sound that is the end goal of the mapping-learning process. It is up to the composer or sound designer what the musical score should be—it may be through-composed or randomly generated, a systematic exploration of the parameter space of a synthesis method or a brief exposition of the musical material that will be played later. And it is up to the user/performer, who might also be the composer, to choose the control modality and gestural vocabulary and to interact with the learning algorithm to ensure that it can successfully learn the intended mapping.

3.2. System Foundation

The Wekinator, our previously constructed system for real-time, interactive machine learning for music, forms the core of our play-along system. The Wekinator presents a single GUI with which the user may specify the input controllers and the controller-specific features (e.g., FFT bin magnitudes or joystick axis positions), choose a learning algorithm and its parameters, create a training dataset, train and evaluate a model, and run a trained model to perform using the learned mapping. All these steps are performed in real-time.

The Wekinator comes with a suite of built-in feature extractors for several input modalities, including time- and spectral-domain audio features, laptop inputs including trackpad position and internal accelerometer values, human input device (HID) controllers such commodity game controllers and custom sensor arrays, and video inputs including edge detection and color-tracking of objects. These feature extraction components are implemented in ChucK [10] and Processing [9]. The Wekinator implements neural networks, AdaBoost, support vector machines, decision trees, and k-nearest neighbor learning algorithms using the Weka machine learning library [11]. The learning component and the GUI are both implemented in Java. All communication among ChucK, Java, and Processing is performed via OSC [12]. Users may implement their own feature extractors (e.g., for other custom controllers) in ChucK or any other OSC-enabled environment.

The real- or discrete-valued outputs of a model can be used in real-time by any synthesis engine that speaks OSC. For a simple example, a neural network might be trained to set the value of the frequency or gain of an oscillator. Or, the outputs could drive higher-level musical parameters, such as thematic content or form. Multiple models are learned in parallel to accommodate any number of parameters, for example controlling frequency, gain, and form simultaneously. The number and continuous or discrete nature of parameters is published by the synthesis engine via OSC.

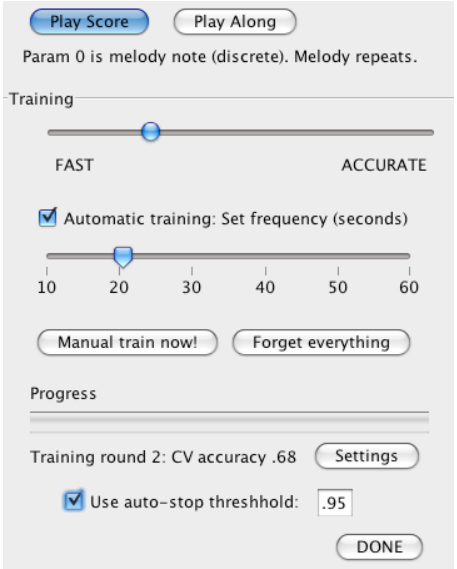

Figure 1. Play-along GUI

3.3. Play-Along Interface

We have added functionality to the Wekinator to allow a user to implicitly construct the training dataset by "playing along," and to control the parameters and behavior of the play-along mapping process. The GUI for this process appears in Figure 1. It allows the user to play and stop the computer score, opt to just listen to the playing score (during which no training examples are created) or to begin playing along, at which time the computer constructs training examples by extracting features from the input controllers and querying the synthesis engine for its current parameters at a given, continuous rate. The user can stop and restart the play-along process at any time. She can also view annotations or instructions associated with particular points in the score. The user may choose to manually trigger training of the learner (in a separate thread from the GUI and sound production), or she can specify that training should occur automatically at a set timing interval. In either case, the user can choose to monitor the cross-validation accuracy of the learning algorithm and may choose to set an accuracy threshold which, if crossed, will result in termination of the learning process. At this point, control over the synthesis parameters transfers to the user: features continue to be extracted in real-time from the inputs, but now they are passed to the trained model, which produces outputs that drive the synthesis engine.

At any point, the user may decide to change the learning algorithm, its parameters, or the input features, and then resume play-along training or use the original Wekinator training interface in hope for a better or different outcome. During the play-along mapping process itself, the user may also manipulate a slider for coarse control over training time. This slider provides a continuum from "very fast" to "very accurate," and its position is translated into an algorithm-specific set of parameters (e.g., a combination of learning rate and convergence threshold for NNs, and the number of boosting rounds for AdaBoost). This slider allows the user to make the tradeoff between time and accuracy based on his current priorities, regardless of his familiarity with the actual algorithm parameters.

3.4. Examples

We have constructed several examples that demonstrate the potential uses of play-along mapping. Each example uses a different synthesis engine and the new Wekinator play-along system as described above.

3.4.1. Notes and Form

In the Notes example, a ChucK synthesis class plays a melody for its score in play-along mode. The melody is specified by the composer as an array of MIDI note values and durations. To play along, a user decides how to trigger each note of the melody. For example, a user may wish that each key on the computer keyboard should correspond to a note. He may play along with the melody as if on a piano keyboard, with pitch increasing as keys move to the right; this will allow him to improvise freely with the notes of the melody by playing these keys, using his piano keyboard skills. Or, a user with no piano skills who simply wants the ability to play back the melody itself, varying timing of the notes, may choose to play along using the number button '1' for the first note, '2' for the second, etc. For a more radical mode of control, the user may choose to play along using the laptop's internal accelerometer, using different tilt positions to trigger the different notes.

When Notes is run as a classification problem, the outputs it produces will be discrete note values. Alternatively, Notes may be run as a continuous learning problem, where the model instead learns the frequencies of the notes, and a continuous input controller such as the accelerometer can be used to slide between frequencies.

The Form example uses discrete classification in a manner similar to Notes, except that the output class value controls the higher-level form of the piece. The synthesis

code play-along score plays through four distinct sections of a piece, and the user plays along to show how he will trigger thematic material from each section. In this example, the score annotations inform the user which section of the score is currently playing.

3.4.2. Synthesis exploration

The Synthesis Exploration example shows how a randomly-generated score can be used to explore the sonic space of a synthesis algorithm. The synthesis method is a physical model of a bowed-string instrument, whose parameters include bow position, vibrato gain, and bow pressure. While physical models sometimes offer an intuitive means of control, it is often not well-understood how their many parameters will interact to make a particular sound (or no sound at all). It is therefore a challenge to develop controllers to drive physical models in interesting and musical ways, a problem explored in [1].

The Synthesis Exploration score performs a slow random walk along each parameter over time, thus traversing an unknown section of the sonic space of the synthesis algorithm. As these parameters change, a listener may perceive the resulting sound to be changing rapidly at times and slowly at others, and to be sometimes desirable and sometimes horrid. The player can therefore play along according to such judgments to produce a controller for the algorithm that reserves greater controller bandwidth for areas of the parameter space that are sonically desirable and expressive.

3.4.3. Interactive video

In this example, we demonstrate the use of play-along mapping for a non-musical end: that of controlling an image on screen. Here, the "synthesis engine" is a graphical engine in Processing whose parameters are the size and position of a circle. The score itself includes a meta-parameter to specify whether the size and position may change simultaneously during play-along learning, or whether only one parameter changes at a time.

While not very visually interesting, this makes a compelling example for experimenting with input controllers that may or may not make it easy to control size without affecting position and vice versa (related to the idea of matching integrality or separability of the task structure and the controller [6]). Unlike the physical model example, here the perceptual effects of each parameter are clearly decoupled.

4. CONCLUSIONS AND FUTURE WORK

Building on our existing system for interactive, real-time learning, we have constructed a system for creating mappings from arbitrary controller inputs to arbitrary synthesis engine parameters in real-time, wherein the user "plays" his controller along with a set musical score. In general, we are excited by the new possibilities presented by taking existing—and even relatively old—machine learning techniques and applying them in real-time in highly interactive, creative systems. Future work will explore further applications of this approach to supporting novel compositional and performance paradigms.

5. ACKNOWLEDGEMENTS

We thank Sumit Basu for his ideas and discussion. This material is based upon work supported under a National Science Foundation Graduate Research Fellowship. This work is also supported by a John D. and Catherine T. MacArthur Foundation Digital Media and Learning grant.

6. REFERENCES

[1] Demoucron, M. "On the control of virtual violins." Ph.D. dissertation, KTH, 2008.

[2] Fails, J. and D. R. Olsen, Jr., "Interactive machine learning," *Intl. Conf. on Intelligent User Interfaces*, 2003, pp. 39-45.

[3] Fels, S. S. and G. E. Hinton, "Glove-Talk: A neural network interface between a data-glove and a speech synthesizer," *IEEE Trans. on Neural Networks*, vol. 4, 1993.

[4] Fiebrink, R., D. Trueman, and P. R. Cook. "A meta-instrument for interactive, on-the-fly machine learning." *Proc. NIME*, 2009.

[5] Fiebrink, R., G. Wang, and P. R. Cook. "Foundations for on-the-fly learning in the ChucK programming language." *Proc. ICMC*, 2008.

[6] Jacob, R., L. Sibert, D. McFarlane, and M. Mullen, Jr. "Integrality and separability of input devices," *ACM TOCHI*, 1(1): 3-26, 1994.

[7] Lee M., A. Freed, and D. Wessel, "Real-time neural network processing of gestural and acoustic signals," *Proc. ICMC*, 1991, pp. 277-280.

[8] Merrill, D. J., and J. A. Paradiso, "Personalization, expressivity, and learnability of an implicit mapping strategy for physical interfaces," *Extended Abstracts: Human Factors in Computing Systems (CHI'05)*, 2005, pp. 2152-2161.

[9] Reas, C., and B. Fry, "Processing: A learning environment for creating interactive web graphics," *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2003.

[10] Wang, G., and P. Cook, "ChucK: A concurrent, on-the-fly audio programming language," *Proc. ICMC*, 2003.

[11] Witten, I., and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Francisco: Morgan Kaufmann, 2005.

[12] Wright, M., and A. Freed, "Open sound control: A new protocol for communicating with sound synthesizers," *Proc. ICMC*, 1997.