DANCE DANCE CONVOLUTION

Chris Donahue
Department of Music
University of California, San Diego
9500 Gilman Drive La Jolla, CA 92093, USA
cdonahue@ucsd.edu

Zachary C. Lipton, Julian McAuley
Department of Computer Science
University of California, San Diego
9500 Gilman Drive La Jolla, CA 92093, USA
{zlipton, jmcauley}@cs.ucsd.edu

ABSTRACT

Dance Dance Revolution (DDR) is a popular *rhythm-based* video game. Players perform steps on a *dance platform* in synchronization with music as directed by on-screen *step charts*. While many step charts are available in standardized packs, users may grow tired of existing charts, or wish to dance to a song for which no chart exists. We introduce the task of *learning to choreograph*. Given a raw audio track, the goal is to produce a new step chart. This task decomposes naturally into two subtasks: deciding when to place steps and deciding which steps to select. We demonstrate deep learning solutions for both tasks and establish strong benchmarks for future work.

1 INTRODUCTION

Dance Dance Revolution (DDR) is a popular rhythm-based video game with millions of players worldwide (Hoysniemi, 2006). Players perform *steps* atop a *dance platform*, containing four buttons, each labeled with an *arrow*. An on-screen *step chart* prompts players to step on the buttons at specific, musically salient points in time. Scores depend upon both *hitting the right buttons* and *hitting them at the right time*. Step charts vary in difficulty with harder charts containing more steps and more complex sequences.

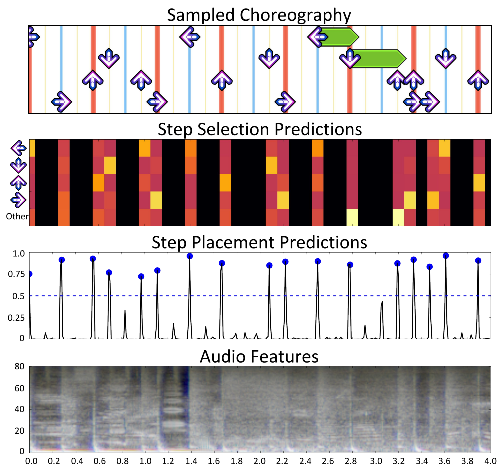

Figure 1: Proposed *learning to choreograph* pipeline for four seconds of the song *Knife Party feat. Mistajam - Sleaze*.

Despite the game's popularity, players have some reasonable complaints: For one, packs are limited to songs with favorable licenses, meaning players may be unable to dance to their favorite songs. Even when charts are available, players may tire of repeatedly performing the same charts. Although players can produce their own charts, the process is painstaking and requires significant expertise.

This paper introduces *learning to choreograph*, the task of producing a step chart from raw audio. We break the problem into two subtasks: First, *step placement* consists of identifying a set of timestamps in the song at which to place steps. This process can be conditioned on a user-specified difficulty level. Second, *step selection* consists of choosing which steps to place at each timestamp. Running these two steps in sequence yields a playable step chart (Figure 1). [1]

For both prediction stages of learning to choreograph, we demonstrate the superior performance of neural networks over strong alternatives. Our best model for step placement jointly learns convolutional neural network (CNN) representations and a recurrent neural network (RNN), which

[1] Demonstration video showing human choreography and the output of *Dance Dance Convolution* side-by-side: https://youtu.be/yUc3O237p9M

integrates information across consecutive time slices. Our best model for step selection consists of a conditional LSTM generative model which receives high-level rhythm features as auxiliary information.

2 Methods

Before applying our step placement algorithms, we transform raw audio samples into perceptually-informed representations. Music files arrive as lossy encodings at $44.1kHz$. We decode the audio files into stereo PCM and average the two channels to produce a monophonic representation. We then compute a multiple-timescale short-time Fourier transform (STFT) using window lengths of $23ms$, $46ms$, and $93ms$ and a stride of $10ms$. We reduce the dimensionality of the STFT magnitude spectrum by applying a Mel-scale filterbank yielding 80 frequency bands. Then we scale the filter outputs logarithmically in accordance with human perception of loudness. Finally, we prepend and append seven frames of past and future context to each frame.

2.1 Step Placement

We consider several models to address the step placement task. Each model's output consists of a single sigmoid unit which estimates the probability that a step is placed. For all models, we augment the audio features with a one-hot representation of difficulty.

Following state-of-the-art work on musical onset detection (Schlüter & Böck, 2014), we adopt a convolutional neural network (CNN) architecture. This model consists of two convolutional layers followed by two fully connected layers. Our first convolutional layer has 10 filter kernels that are 7-wide in time and 3-wide in frequency. The second layer has 20 filter kernels that are 3-wide in time and 3-wide in frequency. We apply 1D max-pooling after each convolutional layer, only in the frequency dimension, with a width and stride of 3. Both convolutional layers use rectified linear units (ReLU) (Glorot et al.). Following the convolutional layers, we add two fully connected layers with ReLU activation functions and 256 and 128 nodes respectively.

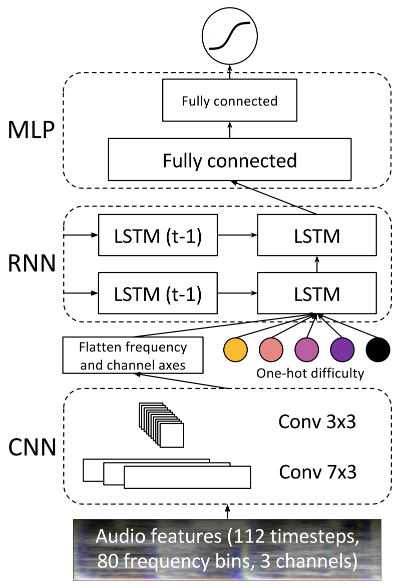

Figure 2: CNN/RNN/MLP model used for difficulty-conditioned step placement

To improve upon the CNN, we propose a C-LSTM model (Figure 2), combining a convolutional encoding with an LSTM-RNN (Hochreiter & Schmidhuber, 1997) that integrates information across longer windows of time. Our C-LSTM contains two convolutional layers (of the same shape as the CNN) applied across the full unrolling length. The output of the second convolutional layer is a 3D tensor, which we flatten along the channel and frequency axes (preserving the temporal dimension). The flattened features at each time step then become the inputs to a two-layer LSTM with 200 nodes per layer. We train this model using 100 unrollings for backpropagation through time.

2.2 Step Selection

We treat the step selection task as a sequence generation problem. Our approach follows related work in language modeling where RNNs are well-known to produce coherent text that captures long-range relationships (Mikolov et al., 2010; Sutskever et al., 2011; Sundermeyer et al., 2012).

Our LSTM model passes over the ground truth step placements and predicts the next token given the previous sequence of tokens. The output is a softmax distribution over the game's 256 possible steps. As inputs, we use a more compact *bag-of-arrows* representation containing 16 features (4 per arrow) to depict the previous step. For each arrow, the 4 corresponding features represent the states *on, off, hold,* and *release*. We add an additional feature that functions as a *start* token to denote the first step of a chart. For this task, our LSTM consists of 2 layers of 128 cells each. We use 64 steps of unrolling, an average of 100 seconds for the easiest charts and 9 seconds for the hardest.

To inform our LSTM of the non-uniform rhythmic spacing of the step placements, we provide the following two pieces of auxiliary information: (1) Δ-*beat* adds two features representing the number of beats since the previous and until the next step; (2) *beat phase* adds four features representing which sixteenth note subdivision of the beat the current step most closely aligns to.

3 Experiments

We collected a dataset consisting of 203 songs, labeled by 9 annotators. One particularly prolific annotator, *Fraxtil*, annotated 90 of these songs for all five difficulty levels. The remaining songs are from a large multi-author collection called *In The Groove* (ITG). In total, across all five difficulty settings, we obtain around 35 hours of annotated audio and $350,000$ steps. [2] We augment our dataset for step selection by synthesizing mirror images of each chart (i.e., interchanging left and right) which we found to improve performance for all models

For step placement, we compare the performance of our proposed CNN and C-LSTM models against a logistic regressor (LogReg) and a 2-layer MLP. For step selection, we compare our proposed LSTM model against a fixed-window MLP and an n-gram model using modified Kneser-Ney smoothing (Chen & Goodman, 1998) with backoff. Both the MLP (MLP5) and n-gram model (KN5) predict the next step from four steps of history and the MLP received the same auxiliary information as the LSTM. We also show the performance of an LSTM model trained with only 5 steps of unrolling (LSTM5) to demonstrate the advantage of longer context.

Model	Dataset	PPL	AUC	F-score
LogReg	Fraxtil	1.205	0.601	0.609
MLP	Fraxtil	1.097	0.659	0.665
CNN	Fraxtil	1.082	0.671	0.678
C-LSTM	Fraxtil	**1.070**	**0.682**	**0.681**
LogReg	ITG	1.123	0.599	0.634
MLP	ITG	1.090	0.637	0.671
CNN	ITG	1.083	0.677	0.689
C-LSTM	ITG	**1.072**	**0.680**	**0.697**

Table 1: Perplexity, area under curve and F-score assessed for the step placement task.

Model	Dataset	PPL	Acc.
KN5	Fraxtil	3.681	0.528
MLP5	Fraxtil	3.428	0.557
LSTM5	Fraxtil	3.185	0.581
LSTM64	Fraxtil	**3.011**	**0.613**
KN5	ITG	5.847	0.356
MLP5	ITG	4.786	0.401
LSTM5	ITG	4.447	0.441
LSTM64	ITG	**4.342**	**0.444**

Table 2: Perplexity and per-token accuracy assessed for the step selection task.

Our experiments demonstrate that on both the step placement (Table 1) and the step selection (Table 2) tasks, deep neural network models outperform traditional baselines. For the step placement task, the best performing method by all metrics is the C-LSTM. For the step selection task, LSTMs outperform other models.

Data augmentation and the inclusion of Δ-beat and beat phase side information give a significant increase in performance to both the MLP and LSTMs for step selection. For example, the LSTM64 model trained on the Fraxtil dataset without side information or data augmentation only achieves a PPL of 3.526 and accuracy of 0.562. Step selection models perform better on the single-author Fraxtil dataset in comparison to the multi-author ITG. Author style tends to be distinctive and thus a collection of single-author sequences is more predictable.

4 Related Work

A few prior systems attempt automatic synthesis of step charts (O'Keeffe, 2003; Nogaj, 2005), however neither establishes a reproducible evaluation methodology or learns the semantics of steps from data. The most closely related work to our step placement task is concerned with onset detection

[2]All data shall be made available at publication time

(Bello et al., 2005; Dixon, 2006), which has previously been attempted with deep neural networks (Eyben et al., 2010; Schlüter & Böck, 2014). Our step selection task most closely resembles conditional language modeling. A recent wave of work in RNNs for language modeling began with (Mikolov et al., 2010; Sutskever et al., 2011). Inspired by this work, several recent papers extend the methods to polyphonic music generation and transcription (Boulanger-Lewandowski et al., 2012; Chu et al., 2016; Sigtia et al., 2016). To our knowledge, ours is the first paper to attempt end-to-end DDR choreography from raw audio with deep learning.

References

Juan Pablo Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B Sandler. A tutorial on onset detection in music signals. *IEEE Transactions on speech and audio processing*, 2005.

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv:1206.6392*, 2012.

Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard University, 1998.

Hang Chu, Raquel Urtasun, and Sanja Fidler. Song from pi: A musically plausible network for pop music generation. *arXiv:1611.03477*, 2016.

Simon Dixon. Onset detection revisited. In *Proceedings of the 9th International Conference on Digital Audio Effects*. Citeseer, 2006.

Florian Eyben, Sebastian Böck, Björn W Schuller, and Alex Graves. Universal onset detection with bidirectional long short-term memory neural networks. In *ISMIR*, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *AISTATS, year=2011*.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

Johanna Hoysniemi. International survey on the dance dance revolution game. *Computers in Entertainment (CIE)*, 2006.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, 2010.

Adam Nogaj. A genetic algorithm for determining optimal step patterns in dance dance revolution. Technical report, State University of New York at Fredonia, 2005.

Karl O'Keeffe. Dancing monkeys (automated creation of step files for dance dance revolution). Technical report, Imperial College London, 2003.

Jan Schlüter and Sebastian Böck. Improved musical onset detection with convolutional neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014.

Siddharth Sigtia, Emmanouil Benetos, and Simon Dixon. An end-to-end neural network for polyphonic piano music transcription. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2016.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Interspeech*, 2012.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.