Dance Dance Convolution

Chris Donahue[1], Zachary C. Lipton[2], and Julian McAuley[2]

[1]Department of Music, University of California, San Diego
[2]Department of Computer Science, University of California, San Diego
cdonahue@ucsd.edu, {*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. [1] This task decomposes naturally into two subtasks: deciding when to place steps and deciding which steps to select. For the *step placement* task, we combine recurrent and convolutional neural networks to ingest spectrograms of low-level audio features to predict steps, conditioned on chart difficulty. For *step selection*, we present a conditional LSTM generative model that substantially outperforms n-gram and fixed-window approaches.

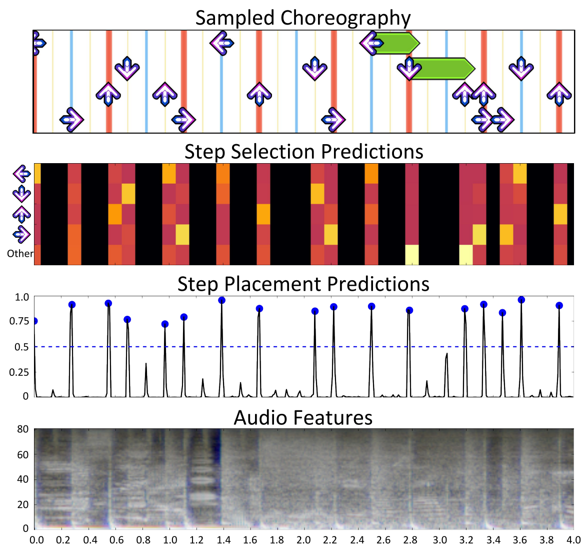

Figure 1: Proposed *learning to choreograph* pipeline for four seconds of the song *Knife Party feat. Mistajam - Sleaze*.

1 Introduction

Dance Dance Revolution (DDR) is a rhythm-based video game with millions of players worldwide [27]. Players perform steps atop a dance platform, following prompts from an on-screen *step chart* to step on the platform's buttons at specific, musically salient points in time. Scores depend upon both hitting the correct buttons and hitting them at the correct time. Step charts vary in difficulty with harder charts containing more steps and more complex sequences. The dance pad contains *up*, *down*, *left*, and *right* arrows, each of which can be in one of four states: *on*, *off*, *hold*, or *release*. Because the four arrows can be activated or released independently, there are 256 possible step combinations at any instant.

Step charts exhibit rich structure and complex semantics to ensure that step sequences are both challenging and enjoyable. Charts tend to mirror musical structure: particular sequences of steps correspond to different motifs (Figure 2), and entire passages may reappear as sections of the song are repeated. Moreover, chart authors strive to avoid patterns that would compel a player to face away from the screen.

The DDR community uses simulators, such as the open-source *StepMania*, that allow fans to create and play their own charts. A number of prolific authors produce and disseminate *packs* of charts, bundling metadata with relevant recordings. Typically, for each song, packs contain one chart for each of five difficulty levels.

[1] End-to-end demo of *Dance Dance Convolution*: **http://deepx.ucsd.edu/ddc**
Demonstration video showing human choreography and the output of *Dance Dance Convolution* side-by-side: **https://youtu.be/yUc3O237p9M**

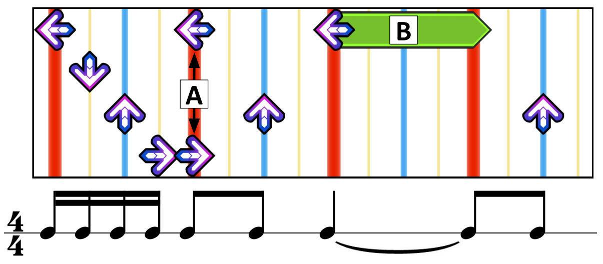

Figure 2: A four-beat measure of a typical chart and its rhythm depicted in musical notation. **Red:** quarter notes, **Blue:** eighth notes, **Yellow:** sixteenth notes, **(A)**: *jump* step, **(B)**: *freeze* step

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.

In this paper, we seek to automate the process of step chart generation so that players can dance to a wider variety of charts on any song of their choosing. We introduce the task of *learning to choreograph*, in which we learn to generate step charts from raw audio. Although this task has previously been approached via ad-hoc methods, we are the first to cast it as a learning task in which we seek to mimic the semantics of human-generated charts. 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*. This process is depicted in Figure 1.

Progress on learning to choreograph may also lead to advances in music information retrieval (MIR). Our step placement task, for example, closely resembles *onset detection*, a well-studied MIR problem. The goal of onset detection is to identify the times of all musically salient events, such as melody notes or drum strikes. While not every onset in our data corresponds to a DDR step, every DDR step corresponds to an onset. In addition to marking steps, DDR packs specify a metronome click track for each song. For songs with changing tempos, the exact location of each change and the new tempo are annotated. This click data could help to spur algorithmic innovation for *beat tracking* and *tempo detection*.

Unfortunately, MIR research is stymied by the difficulty of accessing large, well-annotated datasets. Songs are often subject to copyright issues, and thus must be gathered by each researcher independently. Collating audio with separately-distributed metadata is nontrivial and error-prone owing to the multiple available versions of many songs. Researchers must often manually align their version of a song to the metadata. In contrast, our dataset is publicly available, standardized and contains meticulously-annotated labels as well as the relevant recordings.

We believe that DDR charts represent an abundant and under-recognized source of annotated data for MIR research. *Stepmania Online*, a popular repository of DDR data, distributes over $350Gb$ of packs with annotations for more than $100k$ songs. In addition to introducing a novel task and methodology, we contribute two large public datasets, which we consider to be of notably high quality and consistency. [2] Each dataset is a collection of recordings and step charts. One contains charts by a single author and the other by multiple authors.

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 integrates information across consecutive time slices. This method outperforms CNNs alone, multilayer perceptrons (MLPs), and linear models.

Our best-performing system for step selection consists of a conditional LSTM generative model. As auxiliary information, the model takes *beat phase*, a number representing the fraction of a beat at which a step occurs. Additionally, the best models receive the time difference (measured in beats) since the last and until the next step. This model selects steps that are more consistent with expert authors than the best n-gram and fixed-window models, as measured by perplexity and per-token accuracy.

[2] All code and data shall be released upon publication.

1.1 Contributions

In short, our paper offers the following contributions:

- We define *learning to choreograph*, a new task with real-world usefulness and strong connections to fundamental problems in MIR.
- We introduce two large, curated datasets for benchmarking DDR choreography algorithms. They represent an under-recognized source of music annotations.
- We introduce an effective pipeline for learning to choreograph with deep neural networks.

2 Related Work

Several academic papers address DDR. These include anthropological studies [27, 2] and two papers that describe approaches to automated choreography. The first, called *Dancing Monkeys*, uses rule-based methods for both step placement and step selection [36]. The second employs genetic algorithms for step selection, optimizing an ad-hoc fitness function [35]. Neither establishes a reproducible evaluation methodology or learns the semantics of steps from data.

Our step placement task closely resembles the classic problem of musical onset detection [3, 14]. Several onset detection papers investigate modern deep learning methodology. Eyben et al. [16] employ bidirectional LSTMs (BLSTMs) for onset detection; Marchi et al. [33] improve upon this work, developing a rich multi-resolution feature representation; Schlüter and Böck [39] demonstrate a CNN-based approach (against which we compare) that performs competitively with the prior BLSTM work. Neural networks are widely used on a range of other MIR tasks, including musical chord detection [28, 9] and boundary detection [46], another transient audio phenomenon.

Our step selection problem resembles the classic natural language processing task of statistical language modeling. Classical methods, which we consider, include n-gram distributions [11, 37]. Bengio et al. [4] demonstrate an approach to language modeling using neural networks with fixed-length context. More recently, RNNs have demonstrated superior performance to fixed-window approaches [34, 43, 44]. LSTMs are also capable of modeling language at the character level [29, 30]. While a thorough explanation of modern RNNs exceeds the scope of this paper, we point to two comprehensive reviews of the literature [31, 23]. Several papers investigate neural networks for single-note melody generation [5, 15, 12, 24] and polyphonic melody generation [8].

Learning to choreograph requires predicting both the *timing* and the *type* of events in relation to a piece of music. In that respect, our task is similar to audio sequence transduction tasks, such as musical transcription and speech recognition. RNNs currently yield state-of-the-art performance for musical transcription [6, 10, 40]. RNNs are widely used for speech recognition [20, 21, 22, 38], and the state-of-the-art method [1] combines convolutional and recurrent networks. While our work is methodologically similar, it differs from the above in that we consider a different application and introduce a new learning task.

3 Data

Basic statistics of our two datasets are shown in Table 1. The first dataset contains 90 songs choreographed by a single prolific author who works under the name *Fraxtil*. This dataset contains five charts per song corresponding to increasing difficulty levels. We find that while these charts overlap significantly, the lower difficulty charts are not strict subsets of the higher difficulty charts (Figure 3). The second dataset is a large multi-author collection called *In The Groove* (ITG); this dataset contains 133 songs with one chart per difficulty, except for 13 songs which lack charts for the highest difficulty.

Note that while the total number of songs is relatively small, when considering all charts across all songs the datasets contain around 35 hours of annotations and 350,000 steps. The two datasets have similar vocabulary sizes (81 and 88 distinct step combinations, respectively). Around 84% of the steps in both datasets consist of a single, instantaneous arrow.

Note that step charts contain several invariances, for example interchanging all instances of left and right results in an equally plausible sequence of steps. To augment the amount of data available for training, we generate four instances of each chart, by mirroring left/right, up/down (or both). Doing so considerably

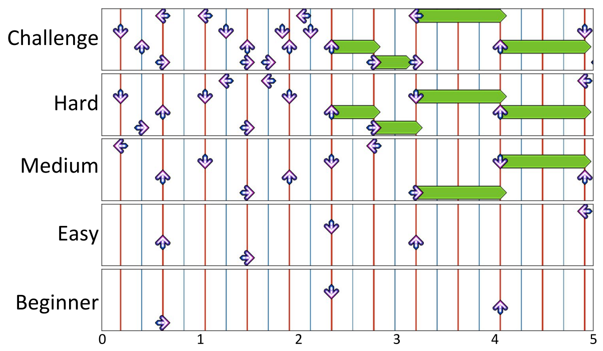

Figure 3: Five seconds of choreography by difficulty level for the song *KOAN Sound - The Edge* from the Fraxtil training set.

improves performance in practice.

Table 1: Dataset statistics

Dataset	Fraxtil	ITG
Num authors	1	8
Num packs	3	2
Num songs	90 (3.1 hrs)	133 (3.9 hrs)
Num charts	450 (15.3 hrs)	652 (19.0 hrs)
Steps/s	3.135	2.584
Vocab size	81	88

In addition to encoded audio, packs consist of metadata including a song's title, artist, a list of time-stamped tempo changes, and a time offset to align the recording to the tempos. They also contain information such as the chart difficulties and the name of the choreographer. Finally, the metadata contains a full list of steps, marking the measure and beat of each. To make this data easier to work with, we convert it to a canonical form consisting of (*beat*, *time*, *step*) tuples.

The charts in both datasets echo high-level rhythmic structure in the music. An increase in difficulty corresponds to increasing propensity for steps to appear at finer rhythmic subdivisions. Beginner charts tend to contain only quarter notes and eighth notes. Higher-difficulty charts reflect more complex rhythmic details in the music, featuring significant amounts of eighth and sixteenth note steps (8th, 16th) as well as triplet patterns (12th, 24th) (Figure 4).

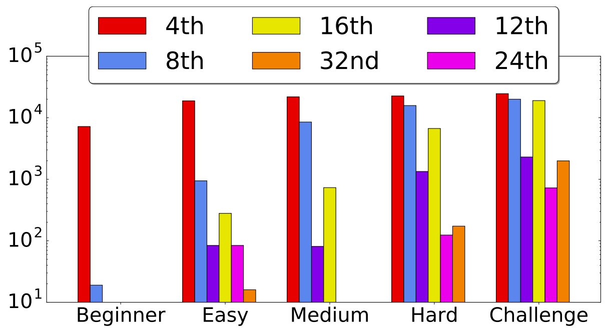

Figure 4: Number of steps per rhythmic subdivision by difficulty in the Fraxtil dataset.

4 Problem Definition

A step can occur in up to 192 different locations (subdivisions) within each measure. However, measures contain roughly 6 steps on average. This level of sparsity makes it difficult to uncover patterns across long sequences of (mostly empty) frames via a single end-to-end sequential model. So, to make automatic DDR choreography tractable, we decompose it into two subtasks: step placement and step selection.

In *step placement*, our goal is to decide at what precise times to place steps. A step placement algorithm ingests raw audio features and outputs timestamps corresponding to steps. In addition to the audio signal, we provide step placement algorithms with a one-hot representation of the intended difficulty rating for the chart.

Step selection involves taking a discretized list of step times computed during step placement and mapping each of these to a DDR step. Our approach to this problem involves modeling the probability distribution $P(m_n|m_0, \ldots, m_{n-1})$ where m_n is the n^{th} step in the sequence. Some steps require that the player hit two or more arrows at once, a *jump*; or hold on one arrow for some duration, a *freeze* (Figure 2).

5 Methods

We now describe our specific solutions to the step placement and selection problems. Our basic pipeline works as follows: (1) extract an audio feature representation; (2) feed this representation into a step placement algorithm, which estimates probabilities that a ground truth step lies within that frame; (3) use a peak-picking process on this sequence of probabilities to identify the precise timestamps at which to place steps; and finally (4) given a sequence of timestamps, use a step selection algorithm to choose which steps to place at each time.

5.1 Audio Representation

Music files arrive as lossy encodings at $44.1kHz$. We decode the audio files into stereo PCM audio 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 $10ms$. Shorter window sizes preserve low-level features such as pitch and timbre while larger window sizes provide more context for high-level features such as melody and rhythm [25]. We compute the magnitudes of the STFT (and discard phase), yielding three channels of two-dimensional spectrograms.

We reduce the dimensionality of the STFT magnitude spectrum by applying a Mel-scale [41] filterbank yielding 80 frequency bands. We scale the filter outputs logarithmically to better represent human perception of loudness. Finally, we prepend and append seven frames of past and future context respectively to each frame.

For fixed-width methods, the final audio representation is a $15 \times 80 \times 3$ tensor. These correspond to the temporal width of 15 representing $150ms$ of audio context, 80 frequency bands, and 3 different window lengths. To better condition the data for learning, we normalize each frequency band to zero mean and unit variance. Our approach to acoustic feature representation closely follows the work of Schlüter and Böck [39], who develop similar representations to perform onset detection with CNNs. We extract features using the ESSENTIA library [7].

5.2 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 onset detection [39], 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) [19]. Following the convolutional layers, we add two fully connected layers with rectifier activation functions and 256 and 128 nodes, respectively.

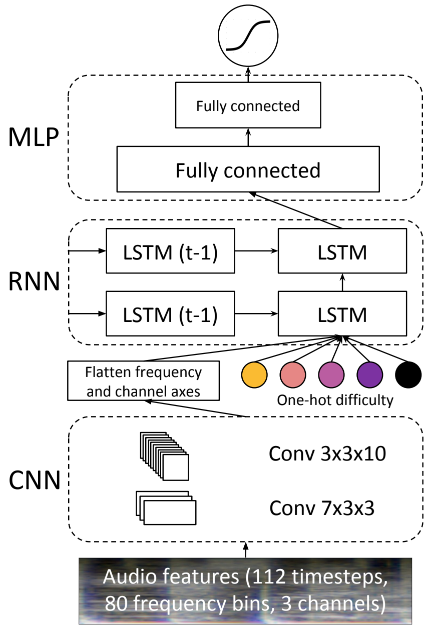

Figure 5: C-LSTM model used for step placement

To improve upon the CNN, we propose a C-LSTM model, combining a convolutional encoding with an RNN that integrates information across longer windows of time. To encode the raw audio at each time step, we first apply two convolutional layers (of the same shape as the CNN) 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 RNN.

The C-LSTM contains long short-term memory (LSTM) units [26] with forget gates [17]. The LSTM consists of 2 layers with 200 nodes each. Following the LSTM layers, we apply two fully connected ReLU layers of dimension 256 and 128. This architecture is depicted in Figure 5. We train this model using 100 unrollings for backpropagation through time.

A chart's intended difficulty influences decisions both about how many steps to place and where to place them. For low-difficulty charts, the average number of steps per second is less than one. In contrast, the highest-difficulty charts exceed seven steps per second. We tried training all models both with and without conditioning on difficulty, and found this feature to be informative. We concatenate difficulty features to the flattened output of the CNN before feeding the vector to the fully connected (or LSTM) layers (Figure 5).[3] We initialize weight matrices following the scheme of Glorot and Bengio [18].

Training Methodology We minimize binary cross-entropy with mini-batch stochastic gradient descent. For all models we train with batches of size 256, scaling down gradients when their l_2 norm exceeds 5. We apply 50% dropout following each LSTM and fully connected layer. For LSTM layers, we apply dropout in the input to output but not temporal directions, following best practices from [47, 32, 13]. Although the problem exhibits pronounced class imbalance (97% negatives), we achieved better results training on imbalanced data than with re-balancing schemes. We exclude all examples before the first step in the chart or after the last step as charts typically do not span the entire duration of the song.

For recurrent neural networks, the target at each frame is the ground truth value corresponding to that frame. We calculate updates using backpropagation through time with 100 steps of unrolling, equal to one second of audio or two beats on a typical track (120 BPM). We train all networks with early-stopping determined by area under the precision-recall curve on validation data.

5.3 Peak Picking

Following standard practice for onset detection, we convert sequences of step probabilities into a discrete set of chosen placements via a peak-picking process. First we run our step placement algorithm over an entire song to assign the probabilities of a step occurring within each $10ms$ frame. [4] We then convolve this sequence of predicted probabilities with a Hamming window, smoothing the predictions and suppressing double-peaks from occurring within a short distance. Finally, we apply a thresholding function to choose which peaks are high enough (Figure 6). Because the number of peaks varies according to chart difficulty, we choose one threshold per difficulty level. We consider predicted placements to be true positives if they lie within a $\pm 20ms$ window of a ground truth.

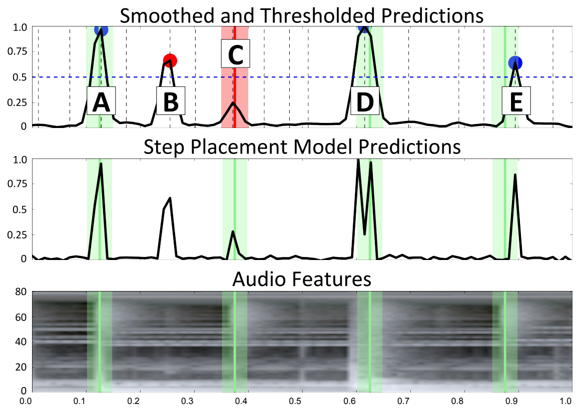

Figure 6: One second of peak picking. **Green:** Ground truth region **(A)**: true positive, **(B)**: false positive, **(C)**: false negative, **(D)**: two peaks smoothed to one by Hamming window, **(E)**: misaligned peak accepted as true positive by $\pm 20ms$ tolerance

5.4 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 [34, 44, 43].

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 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 fea-

[3] For LogReg and MLP, we add difficulty to input layer.

[4] In DDR, scores depend on the accuracy of a player's step timing. The highest scores require that a step is performed within $22.5ms$ of its appointed time; this suggests that a reasonable algorithm should place steps with an even finer level of granularity.

tures represent the states *on, off, hold,* and *release*. We found the bag-of-arrows to give equivalent performance to the one-hot representation while requiring fewer parameters. We add an additional feature that functions as a *start* token to denote the first step of a chart. For this task, we use an LSTM with 2 layers of 128 cells each.

Finally, we provide additional musical context to the step selection models by conditioning on rhythmic features (Figure 7). To inform models of the non-uniform spacing of the step placements, we consider the following three features: (1) Δ-*time* adds two features representing the time since the previous step and the time until the next step; (2) Δ-*beat* adds two features representing the number of beats since the previous and until the next step; (3) *beat phase* adds four features representing which sixteenth note subdivision of the beat the current step most closely aligns to.

Training Methodology For all neural network models, we learn parameters by minimizing cross-entropy. We train with mini-batches of size 64, and scale gradients using the same scheme as for step placement. We use 50% dropout during training for both the MLP and RNN models in the same fashion as for step placement. We use 64 steps of unrolling, representing an average of 100 seconds for the easiest charts and 9 seconds for the hardest. We apply early-stopping determined by average per-step cross entropy on validation data.

6 Experiments

For both the Fraxtil and ITG datasets we apply 80%, 10%, 10% splits for training, validation, and test data, respectively. Because of correlation between charts for the same song of varying difficulty, we ensure that all charts for a particular song are grouped together in the same split.

6.1 Step Placement

We evaluate the performance of our step placement methods against baselines via the methodology outlined below.

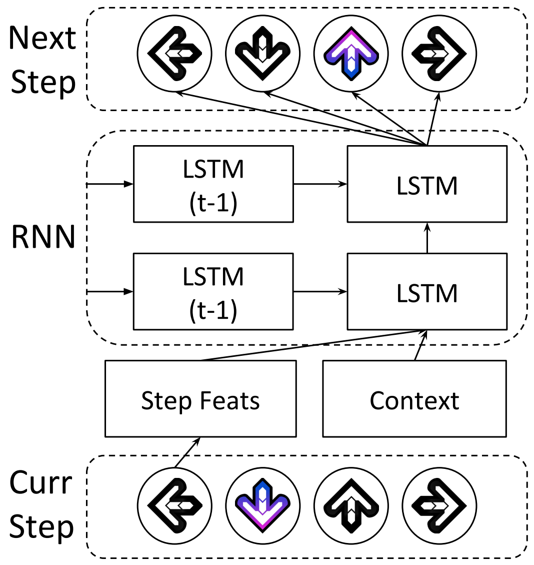

Figure 7: RNN model used for step selection

Baselines To establish reasonable baselines for step placement, we first report the results of a logistic regressor (LogReg) trained on flattened audio features. We also report the performance of an MLP. Our MLP architecture contains two fully-connected layers of size 256 and 128, with rectifier nonlinearity applied to each layer. We apply dropout with probability 50% after each fully-connected layer during training. We model our CNN baseline on the method of Schlüter and Böck [39], a state-of-the-art algorithm for onset detection.

Metrics We report each model's perplexity (PPL) averaged across each frame in each chart in the test data. Using the sparse step placements, we calculate the average per-chart area under the precision-recall curve (AUC). We average the best per-chart F-scores and report this value as F-score^c. We calculate the micro F-score across all charts and report this value as F-score^m.

In Table 2, we list the results of our experiments for step placement. For ITG, models were conditioned on not just difficulty but also a one-hot representation of chart author. For both datasets, the C-LSTM model performs the best by all evaluation metrics. Our models achieve significantly higher F-scores for harder difficulty step charts. On the Fraxtil dataset, the C-LSTM achieves an F-score^c of 0.844 for the hardest

Table 2: Results for step placement experiments

Model	Dataset	PPL	AUC	F-score^c	F-score^m
LogReg	Fraxtil	1.205	0.601	0.609	0.667
MLP	Fraxtil	1.097	0.659	0.665	0.726
CNN	Fraxtil	1.082	0.671	0.678	0.750
C-LSTM	Fraxtil	**1.070**	**0.682**	**0.681**	**0.756**
LogReg	ITG	1.123	0.599	0.634	0.652
MLP	ITG	1.090	0.637	0.671	0.704
CNN	ITG	1.083	0.677	0.689	0.719
C-LSTM	ITG	**1.072**	**0.680**	**0.697**	**0.721**

difficulty charts but only 0.389 for the lowest difficulty. The difficult charts contribute more to F-score^m calculations because they have more ground truth positives. We discuss these results further in Section 7.

6.2 Step Selection

Baselines For step selection, we compare the performance of the conditional LSTM to an n-gram model. Note that perplexity can be unbounded when a test set token is assigned probability 0 by the generative model. To protect the n-gram models against unbounded loss on previously unseen n-grams, we use modified Kneser-Ney smoothing [11], following best practices in language modeling [34, 44]. Specifically, we train a smoothed 5-gram model with backoff (*KN5*) as implemented in [42].

Following the work of Bengio et al. [4] we also compare against a fixed-window 5-gram MLP which takes 4 bag-of-arrows-encoded steps as input and predicts the next step. The MLP contains two fully-connected layers with 256 and 128 nodes and 50% dropout after each layer during training. As with the LSTM, we train the MLP both with and without access to side features. In addition to the LSTM with 64 steps of unrolling, we train an LSTM with 5 steps of unrolling. These baselines show that the LSTM learns complex, long-range dependencies. They also demonstrate the discriminative information conferred by the Δ-time, Δ-beat, and beat phase features.

Metrics We report the average per-step perplexity, averaging scores calculated separately on each chart. We also report a per-token accuracy. We calculate accuracy by comparing the ground-truth step to the argmax over a model's predictive distribution given the previous sequence of ground-truth tokens. For a given chart, the per token accuracy is averaged across time steps. We produce final numbers by averaging scores across charts.

Table 3: Results for step selection experiments

Model	Dataset	PPL	Accuracy
KN5	Fraxtil	3.681	0.528
MLP5	Fraxtil	3.744	0.543
MLP5 + Δ-time	Fraxtil	3.495	0.553
MLP5 + Δ-beat + beat phase	Fraxtil	3.428	0.557
LSTM5	Fraxtil	3.583	0.558
LSTM5 + Δ-time	Fraxtil	3.188	0.584
LSTM5 + Δ-beat + beat phase	Fraxtil	3.185	0.581
LSTM64	Fraxtil	3.352	0.578
LSTM64 + Δ-time	Fraxtil	3.107	0.606
LSTM64 + Δ-beat + beat phase	Fraxtil	**3.011**	**0.613**
KN5	ITG	5.847	0.356
MLP5	ITG	5.312	0.376
MLP5 + Δ-time	ITG	4.792	0.402
MLP5 + Δ-beat + beat phase	ITG	4.786	0.401
LSTM5	ITG	5.040	0.407
LSTM5 + Δ-time	ITG	4.412	0.439
LSTM5 + Δ-beat + beat phase	ITG	4.447	0.441
LSTM64	ITG	4.780	0.426
LSTM64 + Δ-time	ITG	**4.284**	**0.454**
LSTM64 + Δ-beat + beat phase	ITG	4.342	0.444

In Table 2 we present results for the step selection task. For the Fraxtil dataset, the best performing model was the LSTM conditioned on both Δ-beat and beat phase, while for ITG it was the LSTM conditioned on Δ-time. While conditioning on rhythm features was generally beneficial, the benefits of various features were not strictly additive. Representing Δ-beat and Δ-time as real numbers outperformed bucketed representations.

Additionally, we explored the possibility of incorporating more comprehensive representations of the audio into the step selection model. We considered a variety of representations, such as conditioning on CNN features learned from the step placement task. We also experimented with jointly learning a CNN audio encoder. In all cases, these approaches led to rapid overfitting and never approached the performance of the conditional LSTM generative model; perhaps a much larger dataset could support these approaches. Finally, we tried conditioning the step selection models on both difficulty and chart author but found these models overfit quickly as well.

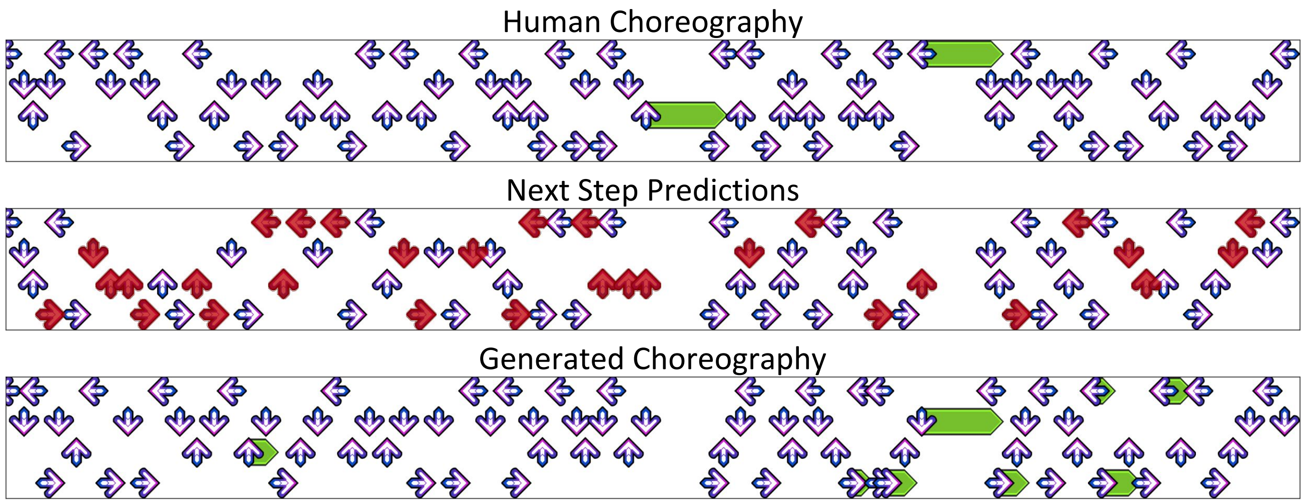

Figure 8: **Top:** A real step chart from *Fraxtil* dataset on the song *Anamanaguchi - Mess*. **Middle:** One-step lookahead *predictions* for the LSTM model, given Fraxtil's choreography as input. The model predicts the next step with high accuracy (errors in red). **Bottom:** Choreography generated by conditional LSTM model.

7 Discussion

Our experiments establish the feasibility of using machine learning to automatically generate high-quality DDR charts from raw audio. Our performance evaluations on both subtasks demonstrate the advantage of deep neural networks over classical approaches. For step placement, the best performing model is an LSTM with CNN encoder, an approach which has been used for speech recognition [1], but, to our knowledge, never for music-related tasks. We noticed that by all metrics, our models perform better on higher-difficulty charts. Likely, this owes to the comparative class imbalance of the lower difficulty charts.

The superior performance of LSTMs over fixed-window approaches on step selection suggests both that DDR charts exhibit long range dependencies and that recurrent neural networks can exploit this complex structure. In addition to reporting quantitative results, we visualize the step selection model's next-step predictions. Here, we give the entire ground truth sequence as input but show the predicted next step at each time. We also visualize a generated choreography, where each sampled output from the LSTM is fed in as the subsequent input (Figure 8). We note the high accuracy of the model's predictions and qualitative similarity of the generated sequence to Fraxtil's choreography.

For step selection, we notice that modeling the Fraxtil dataset choreography appears to be easy compared to the multi-author ITG dataset. We believe this owes to the distinctiveness of author styles. Because we have so many step charts for Fraxtil, the network is able to closely mimic his patterns. While the ITG dataset contains multiple charts per author, none are so prolific as Fraxtil.

A promising direction for future work is to make the selection algorithm audio-aware. We know qualitatively that elements in the ground truth choreography tend to coincide with specific musical events: jumps are used to emphasize accents in a rhythm; freezes are used to transition from regions of high rhythmic intensity to more ambient sections.

DDR choreography might also benefit from an end-to-end approach, in which a model simultaneously places steps and selects them. The primary obstacle here is data sparsity at any sufficiently high feature rate. At $100Hz$, about 97% of labels are null. So in 100 time-steps of unrolling, an RNN might only encounter 3 ground truth steps.

We demonstrate that step selection methods are improved by incorporating Δ-beat and beat phase features, however our current pipeline does not produce beat phase information. In lieu of manual tempo input, we are restricted to using Δ-time features when executing our pipeline on previously unseen recordings. If we trained a model to detect beat phase, we would be able to use these features for step selection.

8 Conclusions

By combining insights from musical onset detection and statistical language modeling, we have designed and evaluated a number of deep learning methods for *learning to choreograph*. We have introduced standardized datasets and reproducible evaluation methodology in the hope of encouraging wider investigation into this and related problems. We emphasize that the sheer volume of available step packs presents a rare opportunity for MIR: access to large amounts of high-quality annotated data. This data could help to spur innovation for several MIR tasks, including onset detection, beat tracking, and tempo detection.

9 Acknowledgements

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575 [45].

References

[1] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *ICML*, 2015.

[2] B. G. Behrenshausen. Toward a (kin) aesthetic of video gaming the case of dance dance revolution. *Games and Culture*, 2007.

[3] J. P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. B. Sandler. A tutorial on onset detection in music signals. *IEEE Transactions on speech and audio processing*, 2005.

[4] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *JMLR*, 2003.

[5] J. J. Bharucha and P. M. Todd. Modeling the perception of tonal structure with neural nets. *Computer Music Journal*, 1989.

[6] S. Böck and M. Schedl. Polyphonic piano note transcription with recurrent neural networks. In *ICASSP*, 2012.

[7] D. Bogdanov, N. Wack, E. Gómez, S. Gulati, P. Herrera, O. Mayor, G. Roma, J. Salamon, J. R. Zapata, and X. Serra. Essentia: An audio analysis library for music information retrieval. In *ISMIR*, 2013.

[8] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *ICML*, 2012.

[9] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Audio chord recognition with recurrent neural networks. In *ISMIR*, 2013.

[10] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. High-dimensional sequence transduction. In *ICASSP*, 2013.

[11] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard University, 1998.

[12] H. Chu, R. Urtasun, and S. Fidler. Song from pi: A musically plausible network for pop music generation. *arXiv:1611.03477*, 2016.

[13] A. M. Dai and Q. V. Le. Semi-supervised sequence learning. In *NIPS*, 2015.

[14] S. Dixon. Onset detection revisited. In *Proceedings of the 9th International Conference on Digital Audio Effects*, 2006.

[15] D. Eck. A first look at music composition using lstm recurrent neural networks. Technical Report IDSIA-07-02, 2002.

[16] F. Eyben, S. Böck, B. W. Schuller, and A. Graves. Universal onset detection with bidirectional long short-term memory neural networks. In *ISMIR*, 2010.

[17] F. Gers and J. Schmidhuber. Recurrent nets that time and count. In *International Joint Conference on Neural Networks (IJCNN)*, 2000.

[18] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[19] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *AISTATS*, 2011.

[20] A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *ICML*, 2014.

[21] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *ICML*, 2006.

[22] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, 2013.

[23] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 2016.

[24] G. Hadjeres and F. Pachet. Deepbach: a steerable model for bach chorales generation. *arXiv:1612.01010*, 2016.

[25] P. Hamel, Y. Bengio, and D. Eck. Building musically-relevant audio features through multiple timescale representations. In *ISMIR*, 2012.

[26] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[27] J. Hoysniemi. International survey on the dance dance revolution game. *Computers in Entertainment (CIE)*, 2006.

[28] E. J. Humphrey and J. P. Bello. Rethinking automatic chord recognition with convolutional neural networks. In *ICMLA*, 2012.

[29] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv:1506.02078*, 2015.

[30] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[31] Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv:1506.00019*, 2015.

[32] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzell. Learning to diagnose with LSTM recurrent neural networks. In *ICLR*, 2016.

[33] E. Marchi, G. Ferroni, F. Eyben, L. Gabrielli, S. Squartini, and B. Schuller. Multi-resolution linear prediction based features for audio onset detection with bidirectional lstm neural networks. IEEE, 2014.

[34] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, 2010.

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

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

[37] R. Rosenfeld. Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, 2000.

[38] T. N. Sainath, R. J. Weiss, A. Senior, K. W. Wilson, and O. Vinyals. Learning the speech front-end with raw waveform cldnns. In *Interspeech*, 2015.

[39] J. Schlüter and S. Böck. Improved musical onset detection with convolutional neural networks. In *ICASSP*, 2014.

[40] S. Sigtia, E. Benetos, and S. Dixon. An end-to-end neural network for polyphonic piano music transcription. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2016.

[41] S. S. Stevens, J. Volkmann, and E. B. Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 1937.

[42] A. Stolcke. Srilm-an extensible language modeling toolkit. In *Interspeech*, 2002.

[43] M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. In *Interspeech*, 2012.

[44] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

[45] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, et al. Xsede: accelerating scientific discovery. *Computing in Science & Engineering*, 2014.

[46] K. Ullrich, J. Schlüter, and T. Grill. Boundary detection in music structure analysis using convolutional neural networks. In *ISMIR*, 2014.

[47] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv:1409.2329*, 2014.