jaki: User-Controllable Generation of Drum Patterns using LSTM Encoder-Decoder and Deep-Q Reinforcement Learning

Fred Bruford[1], SKoT McDonald[2], and Mark Sandler[1]

[1] Queen Mary University of London
[2] inMusic Brands, Inc
fred.bruford@gmail.com

Abstract. In this paper we present ongoing work on *jaki*, an automatic drum pattern generation algorithm designed for use in computer-assisted composition. Using a LSTM Encoder-Decoder model and Deep-Q reinforcement learning, *jaki* generates 1-bar continuations of a 'seed' pattern that can be controlled by musical features, thus generating pattern variations that fit the style of the seed whilst exhibiting certain user-defined characteristics. This paper provides an overview of the algorithm, with several examples of patterns that have been generated with it, along with evaluation and discussion of potential further improvements to it. The algorithm is available for public use, and may be found at `https://github.com/fredbru/jaki`, with further examples.

Keywords: drum patterns, generation, reinforcement learning, LSTM

1 Introduction

An important application of musical computational creativity is computer-assisted composition. Automating certain compositional processes may make them faster for a composer to perform, and computationally generated musical ideas may act as a source of inspiration to them. As the composer may still want some control over the final result, designing generative systems that can be 'tuned' according to user-defined characteristics, such as musical features, could be desirable.

Due to the prevalence of drum loops or drum patterns in many genres of music, a number of systems for generating them computationally have been proposed, with user control a key element in many drum loop generation systems. For example, tools have been designed that enable users to control high-level parameters like complexity in drum pattern combination (Sioros & Guedes, 2011) or syncopation and density for modifying Markov probability distributions representing drum patterns (Jordà, Gómez-Marín, Faraldo, & Herrera, 2016). Others allow a user to sort through pre-made loops from multiple generation algorithms (Vogl, Leimeister, Nuanáin, & al, 2016), or choose a source pattern and the speed of variation generation in a genetic algorithm (Nuanáin, Herrera, & Jorda, 2015). A related method that allows users to supply their own target training data samples for a Variational Autoencoder (VAE)-based system was proposed in (Tokui, 2020). Finally, a conditioning method for LSTMs has been proposed to control drum pattern generation based on various features including metrical structure and analysis of other instrument parts in a track (Makris, Kaliakatsos-Papakostas, Karydis, & Kermanidis, 2019).

Deep reinforcement learning has been proposed to offer another means of tuning music generation algorithms, for example by setting rewards that ensure their output abides by certain music-theoretic rules (Jaques, Gu, Turner, & Eck, 2017). We propose that the same approach may also be applied to work with user-defined musical features, and could offer an alternate means of integrating high-level user control into a drum pattern generation system.

In this paper, we present *jaki*, a drum pattern generation system. *jaki* generates continuations of *symbolic* drum patterns using a Long-Short Term Memory (LSTM) Encoder-Decoder architecture coupled with a Deep-Q reinforcement learning network (DQN). Given an input 1 bar 'seed' drum pattern, the LSTM predicts a second bar of the pattern. The DQN then tunes this new pattern according to a set of musical features, four currently implemented: syncopation, cymbal density, drum density and repetition. The user sets the desired level (low, medium, high) of these features, individually or in any combination, according to what characteristics they want the pattern to exhibit. The reward of the DQN is calculated based on how close the feature levels extracted from the generated pattern are to the target feature levels, and how well it fits the style of the input pattern as predicted by the LSTM. *jaki* therefore aims to generate continuations of drum patterns that exhibit user-defined musical characteristics, whilst being stylistically consistent.

Our approach offers possible advantages to existing approaches to assistive music generation. While neural network-based models may excel at generating stylistically accurate musical content, their tendency to generate the most probable content to fit a dataset may lead to musically uninteresting results (Yang, Chou, & Yang, 2017). Connecting a DQN to a conventional neural network can alleviate this issue whilst keeping the stylistic accuracy, as the rewards of the DQN may be set to maximize features that represent possibly interesting qualities, such as syncopation. Our approach is also flexible in that it could include any possible features, in any combination. As the role of the DQN is to learn a mapping from actions to rewards (see 2.2) the DQN is blind to the reward function, meaning that it may be changed without changing the DQN itself. As in *jaki* the feature targets are incorporated within the reward function, they can be easily modified, allowing a user to set any combination of the four features to any target value, and making it easy for further features to be added.

2 Algorithm

As shown in Figure 1, *jaki* consists of two connected models, an LSTM Encoder-Decoder model and a DQN, the former predicting a pattern and the latter tuning the pattern according to the feature targets and LSTM predictions.

2.1 Generating Drum Patterns with Sequence-to-Sequence learning

In this system, the role of the LSTM is to generate a predicted pattern to be tuned by the DQN, and a model of musical style to constrain this tuning. As *jaki* is designed to generate full 1-bar patterns, this is a problem of sequence-to-sequence learning, for which an LSTM Encoder-Decoder architecture may be used (Sutskever, Vinyals, & Le, 2014). The LSTM Encoder-Decoder uses two separate LSTM networks, an encoder and a decoder. In brief, the role of the encoder is to learn a representation of the input sequence. The decoder then takes this representation as its input and uses it to generate an output sequence.

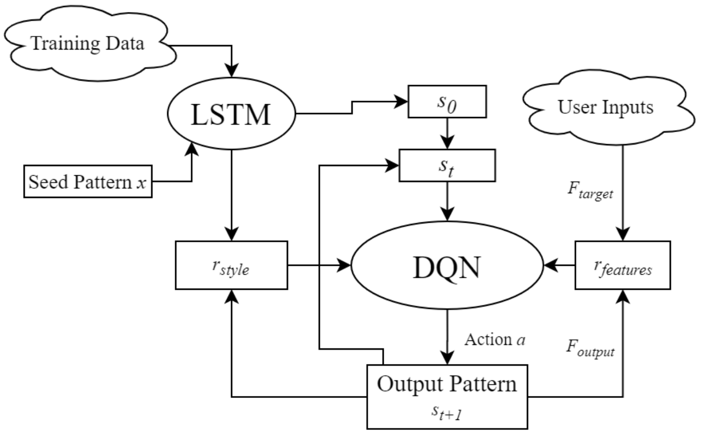

Fig. 1. *jaki* System Diagram

LSTM Training: The LSTM was trained on a dataset of approximately 5500 2-bar MIDI drum patterns to predict the second bar from the first. Each pattern had 5 instrument parts: kick, snare, closed hihat, crash cymbal and tom. As they were MIDI patterns, there was no timbral information. All patterns were in 4/4 with a minimum note value of a sixteenth note. The algorithm was tempo-agnostic, with each 1-bar pattern represented as a 32-by-16 one-hot vector, with 32 different hit/rest combinations of the 5 parts occurring in the dataset, and 16 metrical positions per pattern considering semiquaver precision in one 4/4 bar.

2.2 Tuning to Features using Reinforcement Learning

Formulating the task of generating and tuning a drum pattern as a generic reinforcement learning problem is the first step towards its practical application. In reinforcement learning, an agent and its environment may be defined as a tuple (s_t, a_t, r_t, s_{t+1}). In a given environment state s_t at time t, the agent takes an action a_t leading to a new state s_{t+1}. Based on s_{t+1}, the action is assigned a reward r_t which the system attempts to maximise.

Considering the generated pattern as the environment, with state s_t representing its current arrangement of notes, each action a_t is defined as any single change to this arrangement - either removing or adding a note. In order to generate patterns that both exhibit user-defined musical characteristics and fit the style of the seed pattern, the overall reward r_t for a_t in s_t is the sum of a style reward, r_{style} and a feature reward, $r_{features}$.

$$r(s_t, a_t) = r_{style} + r_{features}$$

r_{style} is calculated using the prediction of the LSTM from the seed pattern x. The fitness of the output pattern to the style of the seed pattern is calculated by the LSTM as the log probability of the output state s_{t+1} given the input x.

$$r_{style} = log\, p(s_{t+1}|x)$$

r_{style} controls the output pattern s_{t+1} to ensure it fits the style of x. The feature reward, $r_{features}$, enables tuning to musical characteristics. The number of features used, n, is chosen by the user. For each feature, the user inputs a target score F^n_{target} based on their desired characteristics (e.g. a high syncopation score for a syncopated pattern). $r_{features}$ is calculated as the sum of absolute differences between target scores F^n_{target} and the feature scores calculated on the output pattern, F^n_{output}, with a weighting factor w_n for each feature:

$$r_{features} = \sum_{i=1}^{n} |F^n_{target} - F^n_{output}| w_n$$

Finally, the most probable pattern predicted by the LSTM is used as the first state s_0. The LSTM thus connects to the DQN through both r_{style} and s_0.

Termination: The algorithm terminates when the overall reward r_t reaches a given threshold set as a hyperparameter. However, it is not guaranteed that for every possible combination of target feature scores and seed pattern there will be a pattern that meets the chosen threshold. To ensure the algorithm still attempts to find the best possible pattern in these cases and terminates at a partial solution, this threshold is decreased over time.

Controllable Musical Features: In theory, any musical features applying to drum patterns may be integrated within this model, though at the expense of ease-of-use. Currently, four simple features are implemented for user control, each which may be set to Low, Medium or High in any combination:

– **Cymbal Density**: Total number of cymbal hits (crash or hihat) in pattern.
– **Drum Density**: Total number of drum hits (kick, snare, tom) in pattern.

- **Syncopation**: Sum of syncopation in each part, calculated using measure from (Longuet-Higgins & Lee, 1984) as implemented in (Bruford, Lartillot, McDonald, & Sandler, 2020).
- **Repetition**: Proportion of hits at the same metrical position in first two and second two beats of the pattern, aka ‘symmetry’ in (Bruford et al., 2020).

2.3 Deep-Q Learning

Following from the above formulation, the role of Deep-Q Learning as performed by the DQN is to predict the action-value function, $Q(a_t, s_t)$, estimating the value of an action as the sum of expected future rewards from taking that action (not just the immediate reward) (Mnih et al., 2013). $Q(a_t, s_t)$ is predicted using a neural network via *experience replay.* As the agent interacts with its environment, it stores its experience at each time step in a buffer as an (s_t, a_t, r_t, s_{t+1}) tuple. It then samples from this memory to select an action, choosing between *exploration* or *exploitation* as it learns by taking either random actions or actions predicted to maximize $Q(a_t, s_t)$.

One issue with standard Deep-Q learning is that it can be prone to overestimating the value of actions. A variant applied in *jaki* that has been shown to alleviate this problem is Double Deep-Q Learning (van Hasselt, Guez, & Silver, 2016). Here, two separate networks are used, with one predicting what action to take, and the other calculating the target Q-value of that action. As these networks operate as a single system, they are represented as such in Figure 1.

2.4 Implementation

jaki was written in Python 3.8.5 using Keras and Tensorflow libraries. It is publicly available at `https://github.com/fredbru/jaki`, with functionality for generating patterns via a command line interface with MIDI file support and a pre-trained LSTM model. The encoder network consists of two LSTM layers of 250 and 100 nodes, with the latter shared as the input to the decoder network. Each of the two DQNs was the same size and shape, with two hidden layers of 24 and 48 nodes. For complete details, please see the GitHub link above.

A final implementation detail to note is that during operation the environment is reset to the initial state s_0 after every 50 time steps. This was found to significantly speed up the generation process by reducing the time spent in unfavourable states that are far away from the LSTM prediction and hence have low r_{style}.

3 Evaluation

The aim of our evaluation was to understand how well *jaki* could generate patterns to fit various feature targets, in particular targets for each feature individually and features in combination, and how well it kept to the target style under these conditions. For 100 seed patterns not included in the training dataset, we used *jaki* to generate patterns under seven different conditions: each feature individually, a random combination of two and three of the features, and all features together. For each of the 100 trials, the target feature score for each feature was set randomly.

For each trial, the distance of the generated pattern to the target style and target feature(s) was measured using the style and feature rewards r_{style} and $r_{features}$ calculated at the final step of generation. As discussed in Section 2.2, the reward threshold required for the DQN algorithm to terminate decreases over time; if the DQN fails to generate a pattern that meets the target reward within a fixed number of time steps, the required reward value is lowered. This decreasing threshold means that the algorithm will search for a partial solution when it cannot perfectly meet the targets. The final rewards achieved at termination, scaled between 0 and 1, therefore indicate the performance of the algorithm based on how well the final generated pattern meets the target feature scores or target style, where a reward of 1 indicates perfectly meeting the targets, and 0 not at all. We take the mean of the 100 trials of each condition.

3.1 Results and Discussion

The results of the evaluation are shown in Table 1. In general the more features that are combined, the harder it is for the algorithm to generate a pattern that meets the target feature scores. It is likely however that for some combinations of features and seed patterns it is not possible to meet the target feature scores perfectly. This may be because making the required musical changes to exhibit the target scores results in too much of a deviation from the LSTM prediction. As more feature targets are added, the amount the pattern has to change increases. In these cases, a weight to control the style reward could be useful to allow patterns to deviate further from the seed to meet the feature targets. In some cases it may also be that a change in one feature affects the others - for example removing a note to decrease density could decrease syncopation if the note was syncopated.

Features	**Mean final** r_{style}	**Mean final** $r_{features}$
Syncopation	0.911	0.917
Cymbal Density	0.900	0.856
Drum Density	0.917	0.940
Repetition	0.922	0.925
2 Features	0.911	0.846
3 Features	0.903	0.821
All Features	0.909	0.775

Table 1. Average of final style reward, r_{style} and feature reward $r_{features}$ for *jaki* run on 100 seed patterns with randomly set feature targets for seven conditions.

A further observation is that the average final r_{style} for each condition varies very little, an unexpected result, as we would expect it to decrease along with $r_{features}$ for harder conditions. An explanation of this is that the environment is repeatedly reset to the initial state s_0, the most probable pattern as predicted by the LSTM, after a fixed number of time steps if the algorithm is unable to meet the reward threshold. Reverting repeatedly to s_0 means that the pattern is less likely to deviate strongly from the LSTM prediction. To encourage a larger deviation from the stylistic prediction, we could reset the environment less frequently, or add an additional weight to r_{style} that decreases its contribution to the overall reward.

Finally, it is interesting to note that the most difficult feature for the algorithm to control is the Cymbal Density feature, with a mean final $r_{features}$ of 0.856, versus the other individual features which are all above 0.917. This is explained by the composition of the dataset. The use of cymbals is relatively consistent throughout each two bar pattern in our dataset, with variation from the first to second bar usually coming from adding ornamentation on snare, kick drum or toms, or sometimes a single crash cymbal. While the algorithm avoiding adding extra cymbals to the pattern is technically a correct reflection of the musical corpus, from a user perspective this is possibly an undesirable feature, as a user may still want to easily add or remove cymbals to vary a pattern. This undesigned behaviour is typical of the kind of unpredictable behaviour that neural network-based systems can exhibit, that can equally become a source of confusion or interest depending on the application.

4 Examples

From a quantitative evaluation such as that performed above, it can be difficult to assess the subjective quality of a generative system. We therefore provide a selection of example patterns generated by *jaki* to be discussed. Figure 2 shows patterns generated by *jaki* for three seed patterns not included in the training set. Each seed is shown with the most probable continuation predicted by the LSTM (initial state s_0), and three continuations generated combined with the DQN with various feature targets. Further examples with audio are provided via the GitHub link above.

As the examples suggest, the LSTM predictions tend to be simplified versions of the seed patterns which, though they may be useful, does support the need for feature-based control to make the patterns more interesting. Otherwise, the examples back up the conclusions from the evaluation above. In most cases the DQN is able to find solutions that are stylistically valid, and exhibit the desired feature scores, with the best results appearing to be when just one or two features are used, e.g. Pattern B iii) and A ii). The challenge of changing cymbal density can also be seen in Pattern C iv) where a high cymbal density only adds one cymbal hit to the pattern.

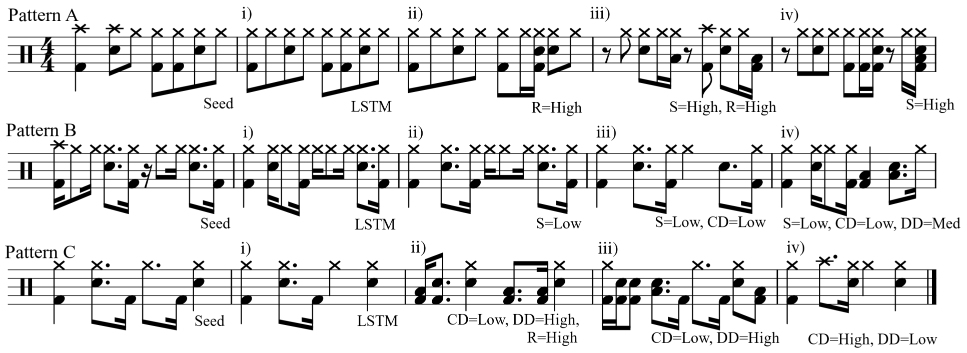

Fig. 2. Example patterns generated by *jaki* with various feature scores. CD = Cymbal density, DD = Drum Density, R = Repetition, S = Syncopation.

5 Conclusion

In this paper we have presented ongoing work into a drum pattern generator that can generate controlled continuations of 1 bar patterns, using an LSTM Encoder-Decoder model coupled with a DQN incorporating musical features into its reward function. The system has been shown to be capable of generating variations on patterns that maintain stylistic features whilst exhibiting user-defined musical characteristics. However, the system is not always able to generate patterns that exactly fit all the desired feature scores and still fit the stylistic model.

For further work, we intend to investigate these cases further, first by looking deeper into the style model of the LSTM and testing different ways of combing the style reward with the feature reward. In cases where solutions are not possible, we will investigate ways to allow more control over the way features are combined, so that a user can prioritise certain features. We will also continue to add new controllable features, as the flexibility of the DQN algorithm makes them easy to integrate in the reward function. In addition to rhythmic features as used in this paper, we aim to include controls for performance techniques such as velocity and expressive timing.

References

Bruford, F., Lartillot, O., McDonald, S., & Sandler, M. (2020). Multidimensional Similarity Modelling of Complex Drum Loops with the GrooveToolbox. In *Proceedings of The International Society for Music Information Retrieval Conference (ISMIR).*

Jaques, N., Gu, S., Turner, R. E., & Eck, D. (2017). Tuning recurrent neural networks with reinforcement learning. In *International Conference on Learning Representations (ICLR).*

Jordà, S., Gómez-Marín, D., Faraldo, , & Herrera, P. (2016). Drumming with style: From user needs to a working prototype. In *Proceedings of The International Conference on New Interfaces for Musical Expression (NIME).*

Longuet-Higgins, H. C., & Lee, C. S. (1984). The Rhythmic Interpretation of Monophonic Music. *Music Perception.*

Makris, D., Kaliakatsos-Papakostas, M., Karydis, I., & Kermanidis, K. L. (2019, June). Conditional neural sequence learners for generating drums' rhythms. *Neural Computing and Applications*, *31*(6), 1793–1804.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*.

Nuanáin, C. O., Herrera, P., & Jorda, S. (2015). Target-based rhythmic pattern generation and variation with genetic algorithms. In *Sound and Music Computing Conference (SMC).*

Sioros, G., & Guedes, C. (2011). Complexity Driven Recombination of MIDI Loops. In *Proceedings of The International Society for Music Information Retrieval Conference (ISMIR).*

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *Advances in neural information processing systems*.

Tokui, N. (2020). *Towards democratizing music production with ai-design of variational autoencoder-based rhythm generator as a daw plugin.*

van Hasselt, H., Guez, A., & Silver, D. (2016). Deep Reinforcement Learning with Double Q-learning. In *Thirtieth AAAI conference on artificial intelligence.*

Vogl, R., Leimeister, M., Nuanáin, C. , & al, e. (2016). An Intelligent Interface for Drum Pattern Variation and Comparative Evaluation of Algorithms. *Journal of the Audio Engineering Society*.

Yang, L.-C., Chou, S.-Y., & Yang, Y.-H. (2017). Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv 1703.10847*.