Reinforcement Learning DJ: Note-Beat Navigation using Deep Q Learning

Omer Hazon [1] **Rahul Palamuttam** [1] **Catalin Voss** [1]

Abstract

We model musical composition as a reinforcement learning (RL) problem in which an agent can choose as an action to add or subtract sets of repeated notes of various pitches from a discrete length melody. We reward an agent for creating bars of music that are close to a corpus of human-generated MIDI music. We utilize deep-Q-learning. Our resulting compositions contain diverse harmonies and inherent global structure by construction. We describe how our approach can be integrated into a real time composition workflow in which the RL agent and human composer take actions on a virtual "launchpad" instrument together to produce a longer, coherent song.

Figure 1. "Launchpad" tool for electronic music composition and performance. The grid can be laid out such that each row corresponds to a note and each column corresponds to a repeated pattern of that note, using additional buttons for offsets.

1. Introduction

We study the task of musical manipulation with reinforcement learning.

Algorithmic music generation has been a topic of much recent interest for its potential to help create copyright-free background music (e.g. for video games or promotional videos) or to assist composers and performers in the creative process. The problem of generating plausible-sounding music is well known to be difficult. The space of music we deem 'pleasurable' is large but very sparse in both MIDI and audio representations and evaluation is difficult at scale – even determining whether a generated track sounds like music at all is often highly subjective.

As neural networks have been used to generate believable content in the text and image domains, machine learning researchers and musicians alike have become interested in using these higher dimensional models for assisting musical composition. Recently, several approaches based on recurrent neural networks (RNNs) have successfully generated *locally believable* music through "NoteRNNs" (Jaques et al., 2016b), similar to Character RNNs (Graves, 2013). However, most of the resulting compositions crucially lack global structure beyond a few bars of a song. Addressing this problem with various memory structures (LSTMs, etc.) in the network has not been successful yet.

To address this challenge, we take a reinforcement learning approach that is constrained to maintain musical structure by design. We aim to mimic the process of musical composition where an artist, viewed as a reinforcement learning agent, learns to play an instrument. We take inspiration from an electronic music composition and performance tool called the "Launchpad", which is often used in a configuration where pressing one of many buttons triggers a repeatedly played pattern of a particular note across a bar of music. Our reinforcement learning agent is given such a virtual launchpad and can choose as an action to add or subtract sets of repeated notes of various pitches from a discrete length melody, and to offset the note patterns in time with respect to each other, i.e. determine the relative phases of each note pattern.

We reward the agent for using its instrument to generate music that is close to samples from an existing dataset of human MIDI compositions (Raffel, 2016). We model our agent using a deep-Q-network with a linear and a deep Q-function approximation.

We observe that our agent creates compositions that contain diverse harmonies and, of course, global structure by design.

[*]Equal contribution [1]Stanford University. Correspondence to: Rahul Palamuttam <rpalamut@stanford.edu>, Catalin Voss <catalin@cs.stanford.edu>, Omer Hazon <hazon@stanford.edu>.

CS 234 Final Project, Stanford, CA, 2018.

However, these compositions quickly converge to a local optimum reward at which point the agent will only take minimal actions to avoid diverging from the high-reward state. We explore two techniques to encourage our agent to create more varying musical improvisations on a longer nice-sounding path throughout the state space:

First, since exploration of the state space is an important aspect of being able to traverse it along a high reward path, and since the states with high reward are sparser and more relevant to the agent, we test an exploration scheme whereby higher reward states are prioritized for exploration. By first creating a simpler, smaller state space with only one note and a generated bar length of 4, we compare ordinary tabular Q-learning with an e-greedy policy to a learning scheme whereby the first stage is purely exploratory, exploring higher reward states with a higher priority, and the second stage follows a purely greedy policy using the resulting learned Q function. The exploration stage is based on Markov Chain Monte Carlo (MCMC) and visits states with a frequency proportional to $\exp(\beta R(s))$, where β is a parameter that may be tuned.

Second, on the large state space, we present a real-time implementation in which the resulting policy can be used by a human composer to explore the launchpad music representation together with the RL agent. For example, the composer can take actions to perturb the RL agent's state to a different key and see if the agent can find a way to recover into pleasurable-sounding music in the new key.

1.1. Related Work

Most previous machine learning approaches to music generation rely on recurrent neural networks (RNNs) to model probabilities of note occurrences as a function of history, encompassed in the recurrent hidden state of the network. This has been illustrated in the popular demonstration of Note-RNN, an adaptation of a Character-RNN, modeling one note at a time like characters for text generation (Graves, 2013). These methods rely on ordinary supervised learning based on MIDI data along with backpropagation through time. Long short-term memory networks (LSTMs) have been used to attempt creating longer, more structured compositions (Eck & Schmidhuber, 2002). Recently, reinforcement learning has been proposed to tune an RNN-approach based on non-differentiable rules from music theory, e.g. not overly repeating the same note, staying on key, and using a musical motif throughout a piece (Jaques et al., 2016a; Jaques et al., 2016), as well as a separate "judging RNN". Other projects have proposed generative algorithmic models that are evaluated directly using human input (Le Groux & Verschure, 2010), such as asking users to slide a bar indicating musical tension and using that as the reward signal to learn to generate musical pieces with high tension. Other work does not attempt to generate music independently but only fills in the duet role as a response to a human musician (Mann).

2. Approach

Due to the complexity of the music generation problem, we make several strong simplifications. First, we define the unit of generation as single bar of music as opposed to an arbitrary length. A longer track can be composed by concatenating several outputted bars which are only slightly modified (by one action at a time) from one to the next.

Each state generates one bar of music. However, rather than encoding music in the time domain, a state generates music in a frequency-like domain we term the beat domain, defined by a set of N keys repeated every b time-steps, where $b \in \{2^1, 2^2, 2^3, ...2^B\}$ (see Table 1). The overall bar length is 2^B, which we set to 64 in our experiments, so $B = 6$. We let $N = 24$ and define these to be the keys evenly around middle C (47 to 71 in the MIDI domain).

Given 24 notes and a bar length of 64, each note track can be composed of 7, (or $B + 1$) basis vectors (Figure 2), defined by the periodicity of beating ($\{2^1, 2^2, \ldots, 2^B\}$, plus an additional single-beat basis vector). The offset values are between 0 and 63, or ($2^B - 1$) and wrap around modulo 64 (or 2^B). For each note there are 7, (or $B + 1$) actions to toggle the presence of any of the basis vectors and 6, (or B) actions to increment the offset by powers of 2 ($\{2^0, 2^1, 2^2, \ldots, 2^{B-1}\}$). We also allow for an additional do-nothing action.

Table 1. Demonstration of the state space structure.

Launchpad representation example					
Note	**Beat every 2 steps**	**Beat every 4 steps**	**Beat every 8 steps**	...	**offset** $\in [0, 64)$
C	0	1	0	...	34
$C\sharp$	0	1	0	...	34
D	1	0	1	...	63
$D\sharp$	1	0	1	...	63
E	1	1	1	...	0
F	0	0	1	...	8
$F\sharp$	0	0	0	...	3
⋮	⋮	⋮	⋮	⋮	⋮
$B\flat$	0	1	1	...	58
B	1	1	0	...	47

The beat locations are staggered so as not to overlap with each other, as shown in Fig. 2. This transform allows even

Figure 2. Each note contributes one track, each track can be composed of a subset of the basis vectors added together. Each row in the figure is a basis vector. The basis vectors are designed to be non-overlapping so a sum of any subset will remain binary. Each track may be offset from the other note tracks by a circular shift.

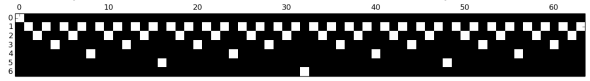
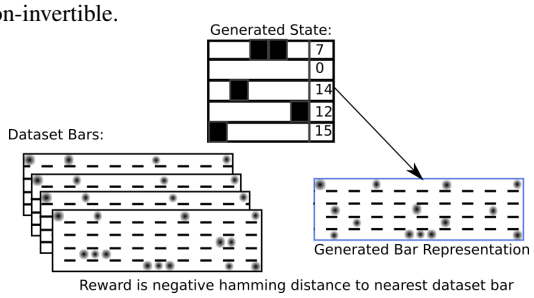


Figure 3. The mapping from state to the MIDI bar representation is non-invertible.



sparse representations to have beat-like and repeating properties. Every available note pitch has its own beat domain, and an offset number, or phase, from 0 to $2^B - 1$, where 2^B is the length of the bar. The total collection of notes and their beat types and offsets is referred to as the launchpad representation. If there are N available notes and $B + 1$ available beat types, then the state space will have a size of $2^{N(2B+1)}$, taking into account the differing offset values, and the action space will have a size of $N(2B + 1) + 1$, taking into account beat-toggling actions, offset actions, and one do-nothing action. When a generated beat domain output is to be evaluated by comparison with preexisting musical tracks, it is converted from the beat domain to the time domain. Note that this conversion is not invertible. See Fig. 3.

We restrict the output space to be binary, except for the offset variables, meaning that for each note, each beat type can either be on or off. The resulting output space resembles the configuration space of a $N \times (B + 1)$ board of buttons for beats, and $N \times B$ buttons for offsets, resembling a Launchpad.

2.1. Music Navigation

The reinforcement learning task is then framed as follows. Within the state space of all launchpad representations (state space $|S| = 2^{N(2B+1)}$), our agent must take actions that correspond to pressing any button on the launchpad (action space $|A| = N(2B + 1) + 1$), in a way that maximizes a reward function that assesses the MIDI bar generated from each state. All actions are deterministic and always succeed. The navigation task is meant to mimic the process of an electronic music artist who is given a Launchpad in a certain configuration, and by listening to the generated audio takes a sequence of actions consisting of single button presses to get the generated audio to sound more musically pleasing. This can be implemented as an non-episodic MDP or an episodic one with a fixed maximum composition length.

2.2. Reward Function

The stochastic reward function is constructed from a preprocessed music dataset. The dataset is converted into a set of bars of the same length as the bar representation of the state space.

For each MIDI track, we estimate the global tempo of the track and then quantize the track into piano-roll bar representations with 64 beats per bar, generally assuming a $\frac{4}{4}$ rhythm. We then only retain bars above a note count threshold (20 in our implementation) that exhibit at least a certain number of different keys (2 in our implementation). For an implementation of our approach that does not depend on key velocity, we binary-threshold all key velocities at 1.

Given a state in the launchpad representation, the evaluation of its reward proceeds as follows:

1. We convert the state to the bar representation:
2. Sample K bars from the dataset, also in the bar representation:
$$\{\vec{m}^{(i)}\}_{i=1}^{K}.$$
3. Return a reward:
$$R(s) = -\min_i \sum_j (m_j^{(i)} - \textit{MIDI-ROLL}(s)_j)^2,$$
where *MIDI-ROLL*(s) is the converted state into the intermediate MIDI roll representation of size $N \times 2^B$ $(= 24 \times 64$, here).

2.3. Deep Q Learning

Given the immensely large state space, we utilize a deep-Q-function approximation implemented in a neural network (DQN).

2.3.1. Neural Network Architecture

Like in (Mnih et al., 2015), we simplify our architecture from the conventional Q-function formulation to output a Q-value for each action a given a state s, so $Q(s, a) = \hat{Q}_{DQN}(s; \mathbf{w})[a]$, where $\mathbf{w}$ are the DQN parameters.

We consider two separate architecture choices:

1. A simple linear (baseline) model that approximates
$$\hat{Q}(s, a) = [M\ \mathbf{vec}(s) + b]_a$$

where $M \in \mathbb{R}^{|S| \times |A|}$ and $b \in \mathbb{R}^{|A|}$ with a single fully connected layer:

$$s \rightarrow [\textbf{fc} \times |A|] \rightarrow \texttt{actions}$$

2. A deep network containing three fully connected (fc) layers:

$$s \rightarrow [\textbf{fc} \times 1024] \rightarrow \textbf{relu} \rightarrow [\textbf{fc} \times 512] \rightarrow \textbf{relu}$$

$$\rightarrow [\textbf{fc} \times |A|] \rightarrow \texttt{actions}$$

This architecture is motivated by the nature DQN (Mnih et al., 2015), but does not utilize convolutional layers since the problem does not resemble vision.

2.3.2. Training

We utilize experience replay with a buffer size of 10^6. We implement the episodic setting and let each sampled episode have a song length of $1,000$ bars. We utilize a separate target network like in (Mnih et al., 2015). We employ a decaying ϵ-greedy exploration strategy with $\epsilon_t \in [0.1, 0.0001]$ in the linear and $\epsilon_t \in [0.2, 0.01]$ in the multilayer case. We use a discount factor of $\gamma = 0.99$.

We train our Q-network to minimize the loss function

$$L(s, a, r) = \left((r + \gamma \max_{a' \in A} Q(s', a';; \mathbf{w}^-)) - Q(s, a; \mathbf{w}) \right)^2,$$

for a given observation of reward r upon taking action a from state s in a minibatch. We use the Adam Optimizer (Kingma & Ba, 2014) to perform the training step update

$$\mathbf{w} = \mathbf{w} + \alpha \left(r + \gamma \max_{a' \in A} \hat{Q}\left(s', a'; \mathbf{w}^-\right) - \hat{Q}\left(s, a; \mathbf{w}\right) \right) \nabla_{\mathbf{w}} \hat{Q}(s, a$$

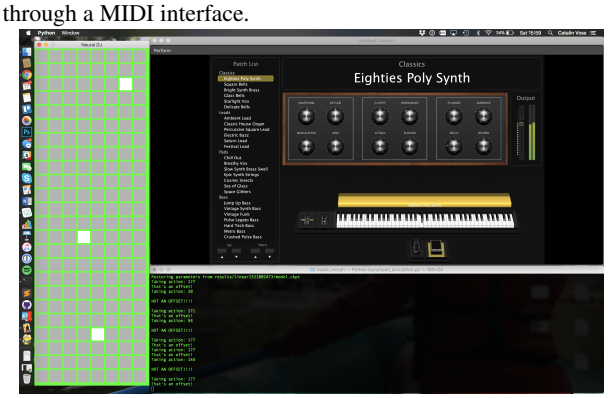
where $\mathbf{w}$ are the Q-network's parameters and $\mathbf{w}^-$ are the target network's parameters and decay our learning rate α from 0.0025 to 0.0005 in the linear and 0.00025 to 0.00005 in the multilayer case. We update the target network parameters every $1,000$ training steps.

3. Interactive Setup

Once a policy is learned, it can be executed beginning from a start state. At each step, we can construct the MIDI representation of the current state and output it in real time using RTMidi (rtm) to a synthesizer such as MainStage. We can then visualize the state space as a 'launchpad' in on the screen and allow a composer to take steps in between the bars.

We have developed a demonstration in PyGame (pyg) that was demonstrated at the Stanford CS 234 poster session,

Figure 4. PyGame-based interface for joint composition. A musician and the RL agent can take actions together on the same launchpad, which outputs the composition bar by bar in real time through a MIDI interface.



illustrated in Fig. 4. The policy is executed in real time and one bar is buffered at a time for playback. A user can take actions at any time that will affect the state. The RL agent takes a single action for every bar. As the policy is optimized for latching onto local rewards, a composer can thus 'guide' the composition process by dictating the key of the song beginning in the start state, for example.

4. Experiments

Figure 5. A comparison of linear approximated Q-learning and Multi-layer perceptron and the corresponding average reward and evaluation reward graphs. Note that the multilayer perceptron at the start has a lower average reward than using linear approximation. However, we see that during the evaluation step the Multi-layer perceptron achieves a much higher reward.

4.1. Q-Network Training

We evaluated two approximation functions for Q-Network training. The first was a simple linear function and the second was a multi-layer perceptron. During training we had enabled gradient clipping at a value of 10. The evaluation reward was calculated at every 250000 steps. We also adopted and modified several hyper parameters from the

original DQN paper. For starters we decided to update the target q-network at every 1000 steps instead of every 10000 steps. We found that having a slightly higher rate of updating the target network proved to be advantageous for both training stability and evaluation purposes. 5 shows the average reward and evaluation reward of both approximation strategies.

We also chose to relax the rate of exploration from the onset. Our ε-greedy strategy starting with ε = 0.2 and ended at ε = 0.01. This was also largely motivated by the amount of time it took to improve average reward and evaluation reward. We found that the agent quickly learned what a good policy was and did not need to explore the state space as much. Furthermore, much of the beat-note state space in actual songs is sparse and so a simple policy that could yield decent rewards is turning off all active note-beat positions. 6 reflects this point as the generated image goes from a very dense representation to a very sparse representation. However, we do agree that following the approach in DQN with ε starting at 1.0 (i.e.) always exploring could have lead to greater realized reward towards the end of training. However, due to time constraints and our desire to iterate quickly we did not want to train the network for more than 3 days at a time.

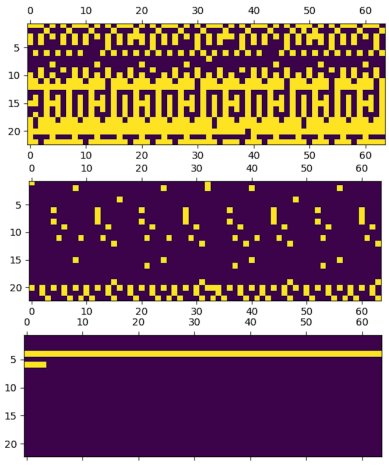


Figure 6. A shows the comparison between the generated beat-note image after 10k steps of training and a single bar from the midi set of images. Note that the image is quite dense. B shows the same comparison but after more than 1m steps of training. Here we see that the generated image is quite sparse. Note that the second image in both plots indicate a single bar from the original set of bars. C shows a sample from the actual set of midi bars. We see that an example target bar is very sparse and so it makes sense why states generated after 1m steps are also sparse.

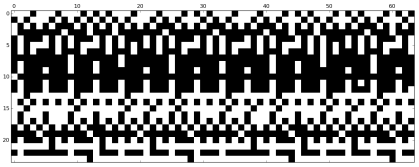
5. Experiments

5.1. Resulting Compositions

Our resulting compositions contain diverse harmonies and inherent global structure by construction.

Some synthesized example compositions are available at `stanford.edu/~catalin/rdj.zip` for reference.

Figure 7. Example of a bar generated from a random sample of the launchpad representation space. White indicates the presence of the note. Each row represents one note.



5.2. Tabular Q learning on small state space for MC exploration

We tested our music composition MDP environment on an MDP of reduced size, small enough to be quickly trained with tabular Q learning. The number of notes was reduced from 24 to 1 ($N = 1$), and the generated bar length was reduced from 64 to 4 ($B = 2$). This resulted in a state space of size 32, and an action space of size 6. The target dataset, from which the reward function is found, was reduced to just the bar of $[1, 0, 1, 1]$. Using the learning rule for tabular Q learning:

$$Q(s,a) = (1-\alpha)Q(s,a)+\alpha\left(R(s)+\gamma\max_{a'}Q\left(T(s,a),a'\right)\right)$$

Where α is the learning rate, set to 0.5, and γ set to 0.999. This learning rule is formulated for a deterministic MDP with a specific reward for a given state and a specific transition function $T(s,a)$ which returns a new state given a state and an action. The Q function was initialized to all zeros, and, given that the rewards are defined to be the negative hamming distance to the nearest bar in the dataset (in this case there is only one such bar), this is an optimistic initialization of Q. Any unexplored state and action will have a default value equal to the optimal ideal, until found to be otherwise.

This tabular Q learning approach was trained using two schemes. The first scheme uses a greedy in the limit of infinite exploration (GLIE) ϵ-greedy approach, where $\epsilon_t = 1/t$. Under this scheme, learning takes place in 300 Q-learning iterations, and remains robust to catastrophic state replacements, namely, when the state is replaced with a random

state, the agent quickly navigates back to the maximal reward state (Figure 8). The agent is able to explore the state and action spaces, as seen in the heatmap for the Q table (Figure 9).

Figure 8. Tabular Q-learning session with ϵ-greedy. Learning occurs in 300 steps and the agent is able to recover from bad reward states. States are randomized every 500 steps.

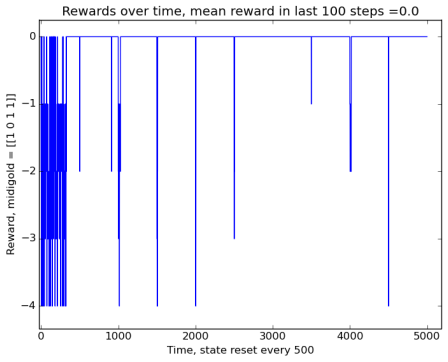
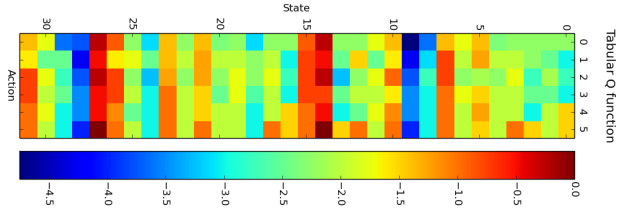


Figure 9. Tabular Q-learning heatmap, showing that exploration has replaced the initial zero values, except where that is the correct value.

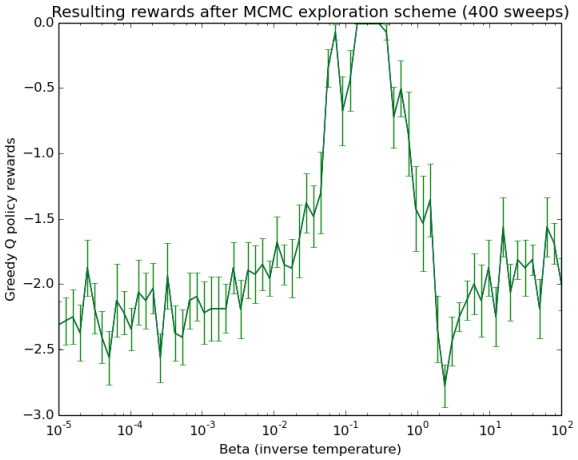


The second scheme aims to address the question of how to speed up exploration. Ideally the Q function should be learned by exploration, not necessarily in search of maximizing rewards yet, and then given to a user or customer who can then generate actions and musical tracks, at which point maximizing rewards is the desired outcome. The exploration part should be as sample efficient as possible, since with a large corpus of MIDI tracks to calculate reward from, the computation may become a bottleneck in the pipeline. Therefore the second scheme utilizes a purely exploratory phase, followed by a purely greedy exploitative phase.

The idea for the exploratory phase is to preferentially explore high reward states, since they contain the greatest information about musical structure and the agent would ideally spend more time traversing those states, rather than the larger portions of the MDP which are lower in reward, and therefore a nuanced measurement of their values is less important. This idea is formulated by having an exploration strategy which will visit each state at a frequency proportional to $\exp(\beta R(s))$. High reward states will be visited exponentially more frequently than low reward states. The exponential can be viewed as a Boltzmann factor with an inverse temperature parameter β. If β is set to 0 then all states will be visited equally (in the limit of infinite exploration). If β is set to be infinite, then the exploration will converge to the highest reward state and will never explore around it. Both extremes have downsides, leading to a hypothesis that some finite value or range of β will be optimal.

To test this idea, we ran the MCMC-based exploration by starting at the empty state and trying each action sequentially. If the action results in a higher reward, switch the state to the new state. If the action does not, then switch to the new state with probability $\exp(\beta\Delta R)$. This can also be expressed as trying actions and switching to the resulting state with probability $\min(1, \exp(\beta\Delta R))$. This approach can only be used with a simulator in which an action can be tried without permanently switching to the new state. The process is repeated until the action space is swept 400 times, at which point the Q function is evaluated with its greedy policy for an additional 400 steps. The value of 400 was selected to distinguish the small range of β values which are able to learn an effective Q function with a small amount of samples. For each value of β the process was repeated 16 times to tighten the SEM errorbars (Figure 10).

Figure 10. MCMC exploration for various values of β on a log scale. The action space is swept 400 times during the exploration phase. More sweeps results in a wider range of β values reaching the reward ceiling. The ideal β is around 0.2.



6. Conclusion

We have shown that by reducing music to be represented on a launchpad i.e. a beat-note space we can leverage Deep-Q-Learning to capture global structure when generating musical bars. Previous approaches looked at leveraging RNN's which have been show to be good at capturing global struc-

ture. Motivated by this we think that utilizing RNN's to do Q-value approximation and utilizing the launchpad environment can enable us to generate music that incorporates both local and global structure.

7. Contributions

Rahul initially adapted the Assignment 2 code for Deep Q Learning to our problem. Catalin wrote the MIDI data parser and Omer developed the rewards environment. Catalin performed training on Google cloud and built the real-time demonstration as well as audio pipeline. Omer implemented and tested tabular Q learning and MCMC exploration. All team members contributed to general debugging as well as the project write-ups and poster.

References

PyGame Library. URL https://www.pygame.org/news.

The RtMidi Tutorial. URL http://www.music.mcgill.ca/~gary/rtmidi/index.html.

Eck, Douglas and Schmidhuber, Jürgen. Finding temporal structure in music: Blues improvisation with lstm recurrent networks. In *NEURAL NETWORKS FOR SIGNAL PROCESSING XII, PROCEEDINGS OF THE 2002 IEEE WORKSHOP*, pp. 747–756. IEEE, 2002.

Graves, A. Generating Sequences With Recurrent Neural Networks. *ArXiv e-prints*, August 2013.

Jaques, N., Gu, S., Bahdanau, D., Hernández-Lobato, J. M., Turner, R. E., and Eck, D. Sequence Tutor: Conservative Fine-Tuning of Sequence Generation Models with KL-control. *ArXiv e-prints*, November 2016.

Jaques, Natasha, Gu, Shixiang, Bahdanau, Dzmitry, Hernndez-Lobato, Jos Miguel, Turner, Richard E., and Eck, Douglas. Sequence Tutor: Conservative Fine-Tuning of Sequence Generation Models with KL-control. *arXiv:1611.02796 [cs]*, November 2016a. URL http://arxiv.org/abs/1611.02796. arXiv: 1611.02796.

Jaques, Natasha, Gu, Shixiang, Turner, Richard E., and Eck, Douglas. Generating music by fine-tuning recurrent neural networks with reinforcement learning. In *Deep Reinforcement Learning Workshop, NIPS*, 2016b.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Le Groux, Sylvain and Verschure, Paul. *Adaptive music generation by reinforcement learning of musical tension*. January 2010.

Mann, Yotam. AI Duet by Yotam Mann - AI Experiments. URL https://experiments.withgoogle.com/ai/ai-duet.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K., Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://www.nature.com/articles/nature14236.

Raffel, Colin. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. Columbia University, 2016.