221 P-Final: Music Generation with Markov Chains and Recurrent Neural Networks

Suraj Heereguppe[a], **Nathan Dalal**[b], **Akash Mahajan**[c]
[a]ICME, Stanford University hrsuraj@stanford.edu
[b]CS, Stanford University nathanhd@stanford.edu
[c]MS&E, Stanford University, akashmjn@stanford.edu

1 Introduction

1.1 Task Definition

Our project explores building generative models, and inference methods for music generation. More specifically, using the symbolic representation of melodies in a dataset of MIDI files, can we generate melodies that make sense and sound good? We use a subset of the Reddit Unique MIDI Dataset for our purpose.

1.2 Related Work

Algorithmic music generation dates back to 1955 when the first rule based musical piece was composed by a computer. Since then a lot of advances have taken place in the field of automatic music generation. The first major progress came when music was modeled as a Markov process. A lot of research has been done in this field, and still continues to go on. However, the categorization of music as a markov process has its own limitations. With the recent advances in neural networks, recurrent neural networks have become the standard for representing musical behavior because they work very well with sequential data. Some related work in this field may be found in the References section of this paper.

1.3 Approach Overview

Our approach to this task consists of three parts:

- **Modelling:** Building a probabilistic model of a melody sequence $p(s_t|s_{t-1}, s_{t-2}, \cdots)$. We explore different state representations and commonly used language modelling approaches.
- **Inference:** Generating plausible-sounding melodies from the given model. We generate both via random sampling, and viewing this as a search problem with beam search.
- **Evaluation:** Examining generated melodies using some quantitative heuristics, along with subjective evaluation

We explored algorithms to generate such sequences and also how to evaluate these algorithms. Starting with a baseline of techniques like Pink Noise, expanded to Markov processes, and built on top of them by extending ideas from previous work using RNNs (recurrent neural networks) either for conditional generation of melodies on chords / harmonies.

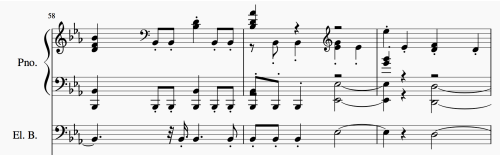

Fig 1 A sample of information contained in a MIDI file

Since music generation is very subjective, we built many simple baseline algorithms and worked on building a good evaluation function, based on a few heuristic music theory rules. Our approach during the initial phase was to get a better understanding of different ways we can formulate our problem in terms of states (for our Markov Process) and encodings (for the RNN-based approach) through many small experiments.

For our baseline we implemented a standard baseline in music generation, pink noise[1]. We then moved to a more advanced approach of Model-based Monte Carlo as an above-baseline performer. After exploring these basic models we experimented with more advanced techniques like Recurrent Neural Networks.

2 Infrastructure

2.1 Note Representation

Unlike audio files MIDI[2] format contain information, much like sheet music about notes played - their duration (start time and end time), their velocity (how hard the key is pressed on a piano), and their pitch (a note in a piano scale). As we can see in Fig 1, a file can have different "tracks" playing simultaneously, and a single track can potentially have multiple notes (chords) playing at once. We spent substantial time developing infrastructure to encode different information (pitch/duration/single melodies/simultaneous notes) in different representations in the form of states / one/many-hot vectors.

We created a very generic representation of notes in sheet music, one that can be used by both model-based Monte Carlo models and Recurrent Neural Networks. One of the ways we represented musical notes is by one-hot vectors. A standard piano has 128 notes. A note played at any given time in a song may be represented as a one-hot vector, with a one at the position which indicates its position on the piano. A guide on how piano notes map to numbers can be found here (there are 128 possible values in a pitch element of a MIDI note).

We encoded these 128-length note vectors by sampling the MIDI file to get notes and rests across tracks. A MIDI file may have multiple tracks containing multiple instruments, and what we did is merge these instruments into one track. Segmenting these notes into multiple chords (by estimating the beat length of a MIDI file), we obtain a matrix of dimension $(n_{beats}, 128)$. These are many hot vectors where each beat represents the chord at that beat (can be a many-hot vector, can be all rests as well).

[1] Douglas Eck - Talk at MUSIC 421n seminar at Stanford Oct 2017

[2] Format containing information much like sheet music https://en.wikipedia.org/wiki/MIDI

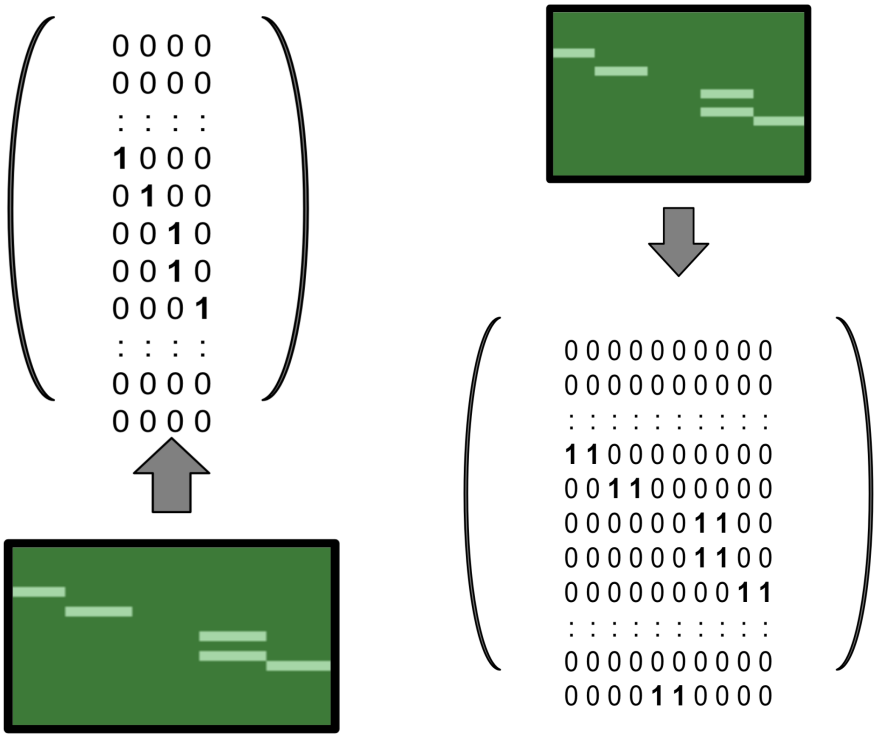

Fig 2 Left: Note-based representation, Right: Time-based representation of same window of notes

For most of our baseline and basic milestone algorithms we converted the many-hot algorithms to one-hot algorithms by picking the lowest note in the chord arbitrarily. Another useful method of thinking of representing musical notes is through intervals, as visualized in Fig 3. However, for our advance RNN models where we tried to generate harmonies, we worked with the many-hot representation of the music that was played at a given time instant.

From **Fig 2**, we see that we represented notes either as a series of notes or a series of of time steps. The note-based representation moves from note to note or chord to chord, not encoding duration of notes or silence; in the note-based representation, each vector represents a note (one-hot) or chord (many-hot). The time-based representation moves from time step to time step, encoding whatever notes appear at that time; if no note appears, silence is encoded by marking the last element with 1. Therefore, this representation encodes chords and silence and duration, but cannot encode repeated notes. This is because duration is implicitly encoded by connecting similar notes. Each vector in the time-based representation represents a window in time (the time of a 16th note, for example).

2.2 Evaluation Metrics

Naturally, due to the subjective nature of this task, evaluation is a challenge. We approach this both with a quantitative heuristic - to get a clearer picture of the difference between different melody samples, and qualitative evaluation - inspection of melodies both by listening, as well visually on a piano-roll format. As we can see in Fig. 4, we can quite effectively see the difference in structure between random generation of notes in our Pink-Noise baseline, and a much more structured output from a piece from our dataset not trained on (that we can consider an oracle).

Inspired by some methods used by Google Magenta Project,[2] we constructed evaluations functions, that given a sequence of notes, compute some basic statistics related to music theory:

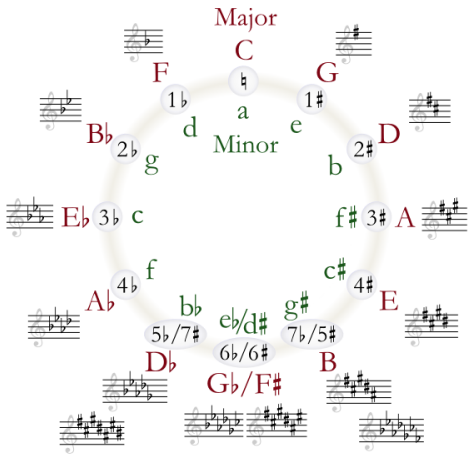

Fig 3 Circle of fifths, showing all 12 notes and related intervals

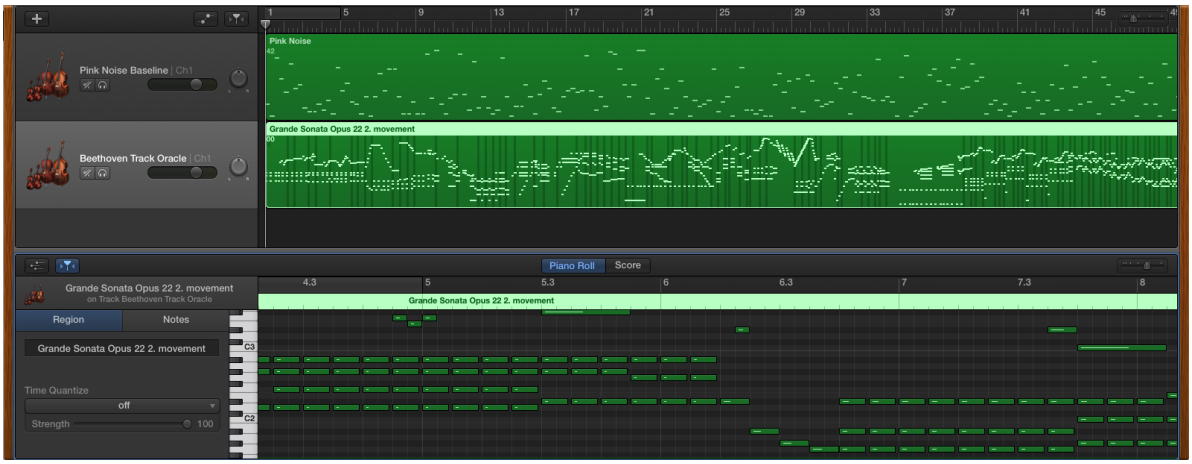

Fig 4 A sample of a subjective evaluation comparing Baseline Pink Noise (above) and an Oracle track from our dataset (below). The composed track visually has much more 'structure' than Pink Noise

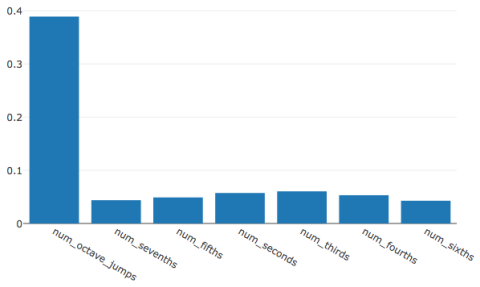

Fig 5 Distribution of intervals in Pink Noise

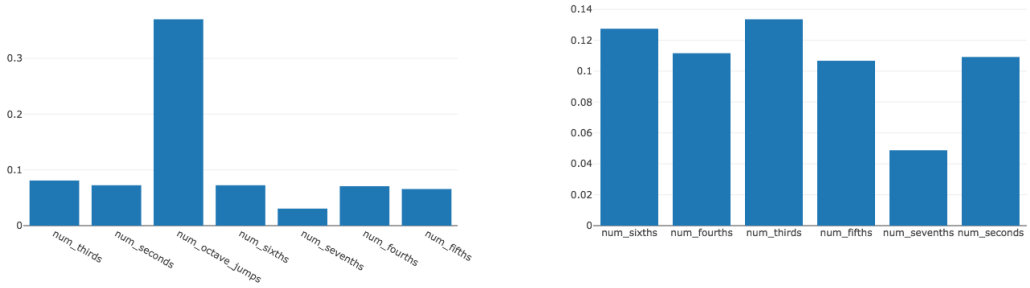

Fig 6 Distribution of intervals in Markov model generated sequences. (Left) - unrestricted sampling (Right) - restricted sampling to reduce big octave jumps

i The fraction of notes that are played 'in key' relative to the root note that was used to seed the sequence. Music typically has several different scales and variations of scales, and we used the Major/Minor scale notes (the most commonly used for this purpose)

ii The distribution of "intervals". While music consists of 12 equally spaced notes (Fig. 3), songs typically start in different 'keys' such as C/D etc. that related to the starting point. What is common to all music however are the relative jumps made (called intervals), and there are some commonly used 'pleasing' intervals. We generate a distribution of intervals in our generated sequences.

3 Data and Experiments

As shown earlier in Fig. 1, MIDI files in the Reddit dataset can have multiple notes or tracks played simultaneously. Since for the bulk of our project, we were dealing with **monophonic** melodies, we picked a subset of the dataset from **pianomidi.de** that contained well-structured **classical pieces** that had individual parts split into multiple tracks, and largely did not have the issues of gaps in parts / hard to separate monophonic melodies. This consists of totally about 400 pieces with monophonic sections, and 500 pieces with multiple parts and harmonies. Also since the dataset has a mix of many different genres, focusing on a single fairly consistent type of music would make the models easier to evaluate and train.

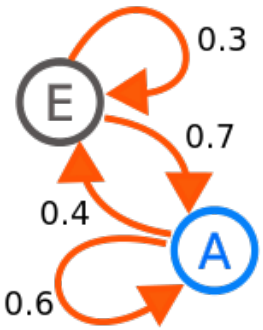

Fig 7 A visual depiction of a Markov model of transitions

3.1 Baseline - Pink Noise

Pink noise generates notes at random, with an inverse distribution of probabilites over note frequencies as below:

$$p(s) \propto \frac{1}{f(s)}$$

This means that the notes sampled produce a random distribution preferring notes with a lower jumps. This allows more notes to be in the low frequency ranges with large jumps to higher notes appearing less often. It offers some continuity in creating random music, but at its base it is still random.

3.2 Algorithm 1: Markov model-based Monte Carlo

Our first above-baseline model is a unigram/bigram based Markov Chain, an approach that's been used in the language modelling domain [TODO: add reference]. The "state" in our case is a tuple of $(pitch \in [0, 129], duration)$, with durations rounded off to multiples of a 16th note, making our states of dimension 129×16

As seen in Fig. 7, the transition probability between different "states" is modelled. More formally we estimate $p(s_t|s_{t-1})$ for order 1 and $p(s_t|s_{t-1}, s_{t-2})$.

Transition probabilities are estimated via a Monte-Carlo approach (keeping counts of transitions) from large samples of MIDI files from our dataset. We observed that it's infeasible to use higher orders than 2 as the transition matrix estimates get too sparse with the exponential growth in state space.

During the music generation phase the model is initialized to a random note in the middle octave From here on, the next note is randomly sampled from the distribution making it self-sustaining for an arbitrary length.

As a slight modification/improvement to the model, we incorporated a minor rule during the generation phase. Typically, no music has note transitions that span an octave or higher. Thus, the modification made to the code during the music generation phase is that a note to be generated is constrained to be within an octave of the previous note generated. In this way, we restrict sharp jumps, keeping the sound a little more subjectively pleasing. The evaluation heuristics in Fig. 5 and 6 give a brief picture of this.

3.3 Algorithm 2: Recurrent Neural Network-based sequence models

One drawback of modeling music as a Markov process is that we assume (in a Markov process) that a fixed window of time is completely capable of predicting what occurs next. Also probabilites However, an RNN can

- Consider a window much larger than allowed by a Markov process, since the hidden state is built up cumulatively
- Is a functional approximator, rather than rote-learning exact transition probabilities
- The cumulatively built hidden state can for account progression of events through time before making a decision about the future

Even in recent music-generation literature, RNN's have been widely used for state-of-the-art models. Our state representation for these parts uses the "piano-roll" representation of one/many hot vectors discretized to every 16th note. Longer or different durations are implicitly captured via repetition. We tried two broadly different approaches to this:

1. Modelling **monophonic** melodies with a simple Char-RNN based model
2. Modelling **polyphonic** melodies with modification to the loss function

3.3.1 Approach 1: Char-RNN Model for monophonic melodies

In the first method based on the previous notes played, the model outputs a probability distribution over the 129 notes that may be played next. During the training process, the note with the highest probability assigned was considered to be the output. This was compared against the true output (the note actually played in the song in consideration). A softmax cross-entropy loss was used to back-propagate through the network and train the model.

After training and during the music generation phase, given a starting set of five notes, the trained model would output a sixth note to be played. To generate the seventh note, notes upto sixth note generated were considered and so forth. Thus, the RNN was self sustaining in generating music. We follow two approaches:

- Randomly pick a note based on the probability distribution output by the model. We essentially followed a Gibbs sampling scheme to pick out the next note to be played
- Beam search with $k = 4$ for a sequence of length to generate a sequence with a maximum join log-likelihood $\sum_{i=1}^{n} \log p(s_i)$

Gibbs sampling introduces a touch of randomness into the generated melody, it does not account for some of the long-term variation in music. We tried to account for such behavioral patterns in music by adding specific constraints to the model in the form of a beam search. While the model learns a generic pattern or trend in music, we provide it with boundary conditions that should also be satisfied so that the melody sounds good end-to-end rather than just in some portions in the middle. What this means is that the progression of notes in the beginning and towards the ending of a song is very different from its musical behavior in between. Without giving the model information

Fig 8 Visual evaluation of Markov Chain sample

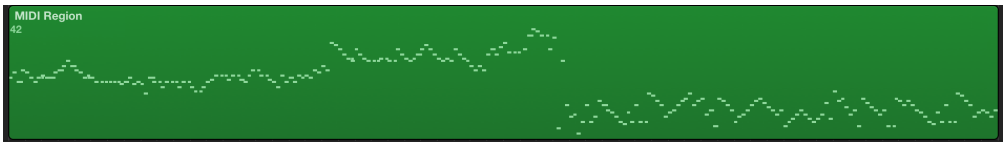

Fig 9 Visual evaluation of RNN with Gibbs sampling

about when and how to end a melody, it will always keep generating notes that sound good in the middle portion of a song. With constraints such as these, the model will find the most optimal way to order notes in such a way that the melody feels more like a song rather than just a snippet. It was one way by which we included a set of "musical rules" that the music generation phase has to follow.

The advantage of beam search is that it is completely independent of the model that generates the music. It can be plugged into any music generation method and will work just as well.

3.3.2 Approach 2: RNN Model for polyphonic melodies

In our second method, we explored the generation of harmonies rather than single melodies. In this case, we worked with the many-hot representation of music (described in Section 2.1). The modeling scheme was from the first approach was retained, except that we had multiple notes playing at any given time. Here the model was required to predict multiple notes to be played at the next time instant. The way we handled this was to train the model to predict both the number of notes to be played next and the actual notes to be played next, as well. One part of the network learns to predict how many notes n to play next. The other portion of the network assigns each of the 128 notes a probability of being played. We then select the notes with the top n values of probability assigned by the network. Mathematically, we use a sigmoid activation function to predict the probability of every note being played next and pick the notes with the top n probabilities. A key assumption here made was that the notes played at the next time instant are independent of each other, which is a very bold assumption but simplifies our task. The results of this modeling approach are also discussed in later sections.

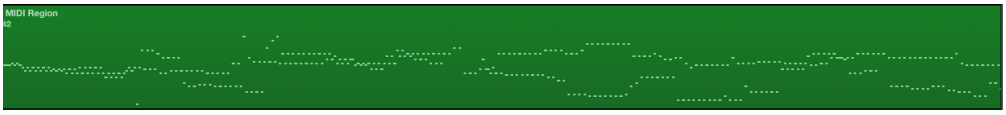

Fig 10 Visual evaluation of RNN with Beam Search

4 Results and Analysis

Aside from a subjective evaluation of the sequences generated via Fig. 8,9,10 the evaluation metrics defined also give us an objective way differentiating between sequences. Some insights noticed:

i We notice that generated melodies modeled via a Markov process $p(s_t|s_{t-1})$ can generate phrases that are 'locally' following some structure. However generating 'phrases' and recurring 'motifs' that typically occur in music need models that have some longer-term structure, not just one-note ahead.

ii A simple forward LSTM was able to capture long term repetitive musical structures better than the markov process. Transitions between notes were more rounded and slowly increasing or decreasing in pitch over time unlike the Markov process where the transition between notes were faster.

iii Both the bidirectional and stacked LSTM's were poorer performers as compared to the forward LSTM when the window size in consideration was less than 20. This may be attributed to the fact that both these networks have more hyperparameters to be learned.

iv A noticeable change from a Markov process to the RNN's was the absence of any notes for a few time instants (silence in music). This is a very important behavior because music is not always constantly being played. In any song, silence in a few places is considered to be good. Although this behavior is not very mature in our RNN, we aim to continue to explore this new promising development.

v With the addition of a new generation methodology, beam search, the music generated by our models was sounded subjectively better. This is because there are additional constraints the music generation phase has to follow, for example start and end on the same note (which sounds better usually).

vi We observed, also, that generating music based on the Gibbs sampling approach introduced a touch of creativity in the melody generated. It also sounded better than when the note with the highest probability was picked. This is because in music, given a sequence of notes, there are multiple notes that sound equally good when played after the sequence. We cannot objectively state that one note is better than the other. Picking the note with the highest assigned probability constrains the generation to stick to a pattern the network learns during training. However, there are several notes that may sound equally good that may not be picked although they have a high enough probability of occurrence. Gibbs sampling accounts for this by sampling from the probability distribution. Thus, the melody generated sounds different from the progression observed in the training dataset and adds to the creativity.

vii Dropout in the LSTM cells seemed to improve the quality of music generated. This is thought to be because the individual cells of the LSTM network are now constrained to be more generic in terms of what they "learn". This may be causing them to learn more of the long term structures in music rather than focus on short-term melodic variations that occur in those positions in the window under consideration.

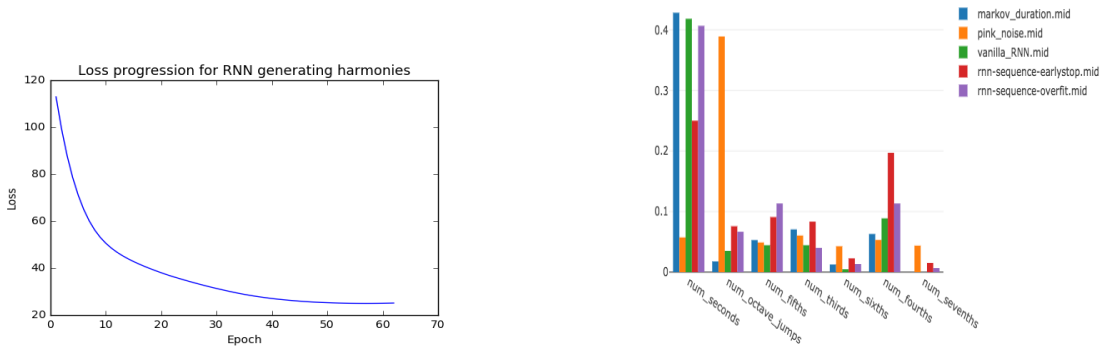

Fig 11 Loss progression for RNN generating harmonies (left), evaluation heuristics for different models (right)

The loss progression for the RNN generating harmonies is shown in Fig 11. The loss term includes both the multi-label cross-entropy loss as well as the squared error loss arising from the error in predicting the number of notes to play in the next time step.

5 Future Work

Throughout this project, for any given model, we have trained it on many songs from just any one genre. We haven't explored the possibility of training a single model with songs in our training dataset belonging to different genres. A genre corresponds to some variation in music, or variance in data in mathematical terms. In our specialized case, increased variance in our dataset (different musical genres in the training set) could actually benefit the model in being able to both learn different types of music and also be creative with respect to the kind of music it generates within a given genre.

We aim to explore note embeddings. Similar to word embeddings, we plan on using an unsupervised learning method to generate good quality note embeddings that capture some of the relationship and interplay between various musical notes. The final aim is to be able to train a model on note embeddings rather than one-hot or many-hot vectors. We want to explore this possibility because there is more signal to be received from a dense vector of real values rather than a long vector of binomial inputs. We also have lined up an entirely new approach to music generation by using Generative Adversarial Networks (GAN's). Recently, GAN's have shown much promise as a potential generative learning algorithm.[1]

6 References

References

1 L.-C. Y. Hao-Wen Dong*, Wen-Yi Hsiao* and Y.-H. Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. https://arxiv.org/pdf/1709.06298.pdf, 2018.

2 N. James. Tuning recurrent neural networks with reinforcement learning. https://magenta.tensorflow.org/2016/11/09/tuning-recurrent-networks-with-reinforcement-learning, 2017.