Memory-Enhanced Machine Learning Models for Music Generation

Ramesh Balaji, Tanmaya Dabral, Stefani Karp

Carnegie Mellon University
rbalaji,tdabral,shkarp@andrew.cmu.edu

Abstract

The problem of training a model to generate novel music falls under the broader umbrella of generative modeling for sequential data. Prior to the deep learning era, generative models for sequential data were often addressed with Markov-model-based approaches. In recent years, deep learning has taken over as state-of-the-art. We examine and thoroughly analyze a variety of different music representations and generative models (particularly Markov chains and recurrent neural networks), with the goal of learning and generating longer-term musical structure. After studying the limitations of standard Markov chains, we proceed to develop a novel hybrid approach: a kth-order Markov chain with an external memory, better-suited for learning long-term structure. We also demonstrate how a thoughtfully-chosen data representation can lead LSTMs to generate very pleasant music even without external memory.

1 Introduction

The goal of **music generation via machine learning** is to train a *generative* model to produce realistic-sounding music. More generally, music generation can be viewed as a time series density estimation problem. This broader formulation of the problem encompasses a wide range of applications with sequential data, from text generation to genomics. In particular, the problem of music generation is often compared and contrasted with text generation. However, many of the challenges of music generation are domain-specific (e.g., melody, rhythm, etc.) and - in the most ambitious cases - must address a combinatorial explosion of possibilities at every time step (e.g., chords, multiple instruments/voices, etc.).

More formally, we can understand the state space challenge as follows. Let $X_1, \ldots, X_T$ be any potentially-generated sequence. The goal is to estimate:

$$P(X_1, \ldots, X_T) = P(X_1) \prod_{t=2}^{T} P(X_t | X_1, \ldots, X_{t-1})$$

Without any additional assumptions, if the state space (i.e., the number of different values X_t can take on) has size N, the number of parameters to be learned is N^T, which quickly explodes for a standard musical piece. Thus, one of the key challenges of the music generation problem is to find appropriate representations *and* conditional independence assumptions that work together in order to reduce the number of parameters to be learned and thus reduce the complexity of the learning problem.

A variety of approaches have been explored over the years to address this problem. Typically, the training examples are musical sequences, often of a particular musical genre (e.g., classical) - though it is also possible to incorporate genre-based conditioning into the generative model. Some older techniques for music generation include Hidden Markov Models (HMMs) and Restricted Boltzmann Machines (RBMs). The state of the art right now, as in much of machine learning today,

is typically achieved using deep neural networks - specifically recurrent neural networks and their longer-term-memory variants such as LSTMs[14].

In our work, we explore a number of these approaches, along with a few novel ones, and cover both Markov models as well as deep neural networks, while at the same time explore multiple different representations each having a different representation power. The rest of the paper is structured as follows: In Section 2, we present a survey of literature already available on the topic. In Section 3, we describe our models and representations. In Section 4, we describe the datasets used. In Section 5, we describe the experiments conducted and analyze the results obtained and finally, in Section 6 we give our conclusions and outline possible directions for future work.

2 Background and Related Works

Music generation approaches in the existing literature can be broadly divided into two categories based on the input data used: approaches that use symbolic/instruction-based forms (e.g., MIDI files, piano roll files) [15; 9] and approaches that use raw audio (e.g., MP3 files) [19].

The MIDI format represents music as the start and end time for each note, along with some other properties, whereas the piano-roll format represents music as the set of notes "pressed down" at each time step. In the surveyed literature, both of these symbolic approaches are much more popular than the raw audio format. This can be attributed to the large expressivity of raw audio, thus requiring a considerably larger dataset for training purposes and more complex models(and typically signal-processing expertise as well). Examples of the symbolic approach in existing potential baseline implementations are MelodyRNN [11] and DeepJazz [18], which use MIDI files, and Huang et al. [15], which uses *both* MIDI and piano roll data [3]. In contrast, Google's Wavenet [25] and GRUV [21] use raw audio input, and Engel et. al. [24] introduced a raw audio music dataset, which they used to generate embeddings capturing musical features.

The piano-roll and MIDI datasets are more readily used for music generation, the most popular being the Lakh Pianoroll Dataset [1]ye and the Lakh MIDI Dataset [5], which are collections of about 175,000 tunes in piano-roll and MIDI formats, respectively, and will be the primary focus of the project. Another popular MIDI dataset is the Classical MIDI Piano repository [2]. Most of the literature utilizing these formats exploits the temporal nature of music via recurrent neural networks. Liu et al. [20] propose a Long Short-Term Memory (LSTM) network trained with RProp to generate MIDI music by treating the generation problem as a multi-class classification problem over the set of possible signals at each time step. Dong et al. [8], on the other hand, use a Generative Adversarial Network (GAN) and take into account the hierarchical structure of musical pieces to model the distribution of the dataset. Similarly, Roberts et al. [24] use a Recurrent Variational Autoencoder (RVAE) with a bidirectional encoder and a hierarchical decoder.

There exists an inherent problem of quantifying the subjective quality of the music generated. To circumvent the issue, most of the surveyed literature uses the log-likelihood of the songs in a test set as a surrogate.

3 Methods

3.1 Markov Models

3.1.1 1st-Order (Memoryless)

We begin with 1st-order time-homogeneous Markov chains, enabling a single state transition matrix to characterize all time steps. We use all songs in the training set to learn the initial probability matrix and the transition probability matrix.

The MLEs of the probabilities all have well-known closed-formed solutions. Let p_{ij} represent the probability of transitioning to state j for time $t+1$ given that the system is in state i at time t, and let p_i represent the probability of a new sequence starting in state i. Let n_{ij} be the total number of transitions from state i to state j, and let n_i be the total number of sequences for which state i is the

starting state. Suppose there are m states. Then, we have:

$$p_{ij} = \frac{n_{ij}}{\sum_{k=1}^{m} n_{ik}} \tag{1}$$

$$p_i = \frac{n_i}{\sum_{k=1}^{m} n_k} \tag{2}$$

where the denominator in (2) is simply the total number of sequences in our training set. (See [23] for our own numpy-based implementation and Section 5.1 for empirical results.)

3.1.2 kth-Order (Weak Memory)

We then explored higher-order Markov chains. Higher-order Markov chains are generally stored using a multidimensional array, with $k+1$ dimensions for kth-order chains. However, if N is the size of the state space, then the number of entries in the multidimensional array grows as $O(N^{k+1})$, which quickly exceeds the storage capacity of most machines. We thus re-implemented our Markov models using *dictionaries* (see [23]) to take advantage of matrix sparsity. We did not find any open-source packages that seemed suitable and thus implemented our dictionary-based Markov chain from scratch. At inference time, we used Laplace smoothing (i.e., adding a constant pseudocount to every possible transition) to ensure that the likelihood of every sequence was > 0.

3.1.3 Markov Chains with External Autoassociative Memory (Strong Memory)

We quickly discovered that simply increasing the order of our Markov chains was insufficient to learn long-term structure. For example, suppose that a song has the note sequence *ABCABCDEF* beginning at index t_1 and that this sequence repeats again at index t_2. If t_1 and t_2 are separated by an arbitrary distance, it is unlikely that even a k-th order Markov chain will learn to exactly repeat the t_1 sequence at t_2. Furthermore, making k large enough to even be *capable* of capturing all such repetitions in a song is, in general, impossible for a fixed k of reasonable magnitude. We wanted to design a *more robust memory system* that could learn this repetitive behavior, which we hypothesized is a key component of both musicality and predictive power, especially for the melodies of the Nottingham dataset. We considered various possible extensions of a kth-order Markov chain (including some HMMs results that we are leaving out of the final report due to space constraints). Ultimately, we took inspiration from the growing cognitively-inspired trend of incorporating *external* memory systems into neural networks [12; 13; 22]. We were particularly interested in the idea of building an autoassociative memory for our Markov chain (i.e., using just part of a sequence to look up a longer sequence in memory) as we thought that this captured the nature of the *ABCABCDEF* problem. We considered a variety of different implementations and converged on the following:

Our overall *learned* model is still a kth-order Markov chain, trained in the standard empirical-counts based manner (with the addition of Laplace smoothing). This is described in further detail in Section 3.1.2. The *creativity* and power of our approach is in the inference stage, which is where the memory comes into play. Each song builds its own separate memory as inference progresses. We use a trie to store all n-gram sequences seen so far so in the song and compute the probability of a new note using a combination of the trie (learned using only the current song) and the Markov model (learned using all training data). Specifically, the weight of the trie's prediction is proportional to whether the immediately previous notes match an existing sequence in the trie and how *long* that sequence is. Thus, in the *ABCABCDEF* example, the weight of the trie will be higher when predicting t_2's F than when predicting t_2's first C. We believe this captures well the nature of repetition in a musical piece. Please see [23] for the complete implementation.

Instead of representing notes as pitches, we represent notes as pitch offsets from one note to another. This allows the trie to store *patterns*, decreasing the size of the state space and making each path in the trie applicable to a wider range of sequences in the song.

To the best of our knowledge (after a best-effort literature review), we believe that combining an external autoassociative memory with a Markov chain in this manner, to learn long-term musical structure, is a very novel approach. Once the melody is generated using our memory-augmented Markov chain, there are numerous methods one could then use to generate a chord-based harmony, to enhance the overall sound of the generated piece.

3.2 LSTM-based language models

The problem of music generation can be seen as equivalent to modeling a language over the vocabulary of representation-dependent musical states. A natural choice for modeling temporal sequences are Recurrent Neural Networks(RNNs), which are neural networks with loops, thereby allowing information to persist in them across time-steps. Long Short Term Memory (LSTM) Networks [14] are RNNs specifically designed to capture long term dependencies. LSTM networks have been successfully used to model natural languages in the past, often setting the state-of-the-art. We therefore explore various different ways to represent music as a language and and train LSTM-based language models using them.

3.2.1 LSTM - MIDI Pitch and Chords

A simple and intuitive way to represent music from a piano would be to encode the music as a series of events with each event representing either a note or a chord i.e a collection of notes. The set of all such unique events would then form the vocabulary of our language model which is saved as a dictionary. At the first time-step the first event is encoded as a one-hot vector and passed through an embedding layer to generate a feature vector of size 400 and is input to the LSTM. Our LSTM has a 2-layered architecture with 1250 hidden units. At each time-step outputs a probability distribution over the output vocabulary. We sample from this output probability distribution and feed it as input in the next time-step thus generating a series of events. This series of events are then decoded back to the midi form by the same dictionary used for encoding.

3.2.2 LSTM - MIDI Pitch, Offset and Duration

The previous approach represents music using only the pitch and ignores the other characteristics of music such as the duration for which a note is played and spacing between the notes. To overcome these shortcomings we devise a representation based on previous work in [6], which uses the pitch, the duration, and the time offset from the last note. Thus we represent the music as a series of events with each event represented as a tuple of (pitch,offset,duration). In the first time step, an embedding of size 256 is generated for each of the elements for the first event. These embeddings are then concatenated and fed into the LSTM which has 3 layers and 512 hidden units per layer. We have three scoring layers after the LSTM, each of which produces a probability corresponding to one of the quantities from which we sample the event for the next step.

3.2.3 LSTM - Binary Piano Roll

A simple, intuitive representation of the piano roll format is a series of binary vectors, representing time-steps. At each time-step, a 1 in the k^{th} dimension corresponds to the k^{th} key being pressed.

An interesting direction to explore was to perform Independent Component Analysis (ICA) [16] on these time-series of vectors and analyzing the components independently. However, the ICA algorithm does not converge, suggesting that the independent components of piano music share a more complicated relationship than a simple linear combination.

We further explore this representation by training an LSTM language model that models a probability distribution over all possible configurations of the piano keys, given configurations of all the previous time-steps. To make the state space tractable, we enforce independence of each key given the output of the LSTM at each time-step.

Let $X_1, \ldots, X_T$ be a sequence of d dimensional binary piano roll vectors. Using the assumptions mentioned above, we can then denote the joint probability as follows:

$$
\begin{aligned}
P(X_{1:T}) &= P(X_1) \prod_{t=2}^{T} P(X_t | X_{1:t-1}) \\
&= P(X_1) \prod_{t=2}^{T} \prod_{k=1}^{d} P(X_t^k | X_{1:t-1}) \text{ where we model } P(X_t^k | X_{1:t-1}) \text{ using an LSTM}
\end{aligned}
$$

3.2.4 LSTM - Pitch-Pattern Piano Roll

We hypothesize that decomposing each piano-roll time-step into its base pitch, that is, its lowest note and the pattern relative to it should yield better results than the vanilla binary representation. We base this hypothesis on the fact that similar class of chords share the same key patterns.

To test this hypothesis we train another language model that models the base note and the relative pattern separately. The base note is modelled as a categorical distribution, while the relative pattern is modelled as a set of conditionally independent Bernoulli distributions. Note that depending on the first note, the number of possible patterns changes.

Let $X_1, \ldots, X_T$ be a sequence of d dimensional binary piano roll vectors. Let the 1-based index of the first note at time-step t be u_t. We can then calculate the joint probability as:

$$P(X_{1:T}) = P(X_1) \prod_{t=2}^{T} P(X_t | X_{1:t-1}) \tag{3}$$

$$= P(X_1) \prod_{t=2}^{T} P(u_t | X_{1:t-1}) \prod_{k=1}^{d-u_t} P(X_t^{u_t+k} | X_{1:t-1}) \tag{4}$$

where we model both $P(X_t^{u_t+k} | X_{1:t-1})$ and $P(u_t | X_{1:t-1})$ using an LSTM.

Note that even though this decomposition biases the joint distribution, the total state space remains the same and therefore, the likelihood values are comparable to the ones obtained using the previous representation.

3.3 Modeling Latent Space Representations (Piano Roll)

We tried to project all the melodies onto a fixed dimensional space using a sequence-to-sequence autoencoder. We hypothesized that modeling the distribution of the fixed dimensional embeddings would be easier than modelling the variable lengthed songs. However, the sequence-to-sequence model was unable to learn the latent space representations, with the average negative log-likelihood for the validation dataset saturating at ~ 10.3 on the Nottingham dataset which we have empirically found to correspond to the network learning unconditional distributions. We believe that since the state space per time-step is exponential in the number of keys, it is infeasible to project the entire melody onto a fixed dimensional space.

4 Dataset

As mentioned previously, we explore two different symbolic representations of music: the MIDI format and the piano roll format. We considered a variety of different datasets, but ultimately chose to focus on the following four datasets available in both MIDI and piano roll formats, since most of the literature that we surveyed provide benchmarks for them:

- **Piano-midi.de** is an archive of classical piano pieces [2]. Note that this was the dataset provided to us.
- **Nottingham** is a collection of 1200 folk tunes. [10]
- **MuseData** is a collection of orchestral and classical piano music from CCARH [4].
- **JSB Chorales** is a corpus of 382 harmonized chorales by J.S. Bach [17].

For each dataset, we use the train-test-validation split provided by [3].

To process MIDI files, we used the open-source music21 [7] Python library.

5 Experiments and Results

In this section we describe the experiments performed using the models described in section 3. We also provide quantitative comparison based on log-likelihood values over the test set. Note however, that the likelihood values are generally not comparable across representations with different state space sizes.

Dataset	# States	Uniform Random NLL	Train NLL	Test NLL
Nottingham	392	5.97126184	2.51536437	2.800220907
JSB Chorales	1695	7.43543802	2.965651723	7.220007994
Piano MIDI	6661	8.804024902	3.342980938	7.900961281
MuseData	37010	10.51894343	3.738933387	8.375618867

Table 1: Results for a 1st-order Markov chain with pitch and chords representation.

5.1 Markov Models for Melody

All experiments here refer to the Nottingham dataset, and we introduce α to refer to the pseudocount added for Laplace smoothing. We performed a search over various settings for k, α, c by computing the average train and test log-likelihood per time step for various settings of these parameters: $k \in [1, 15]$ at intervals of 1, $\alpha \in [0.1, 1]$ at intervals of 0.1, $c \in [0.1, 1]$ in intervals of 0.1.

The best average NLL for the Markov models enhanced with external memory is 1.7301 (achieved by $k = 2, \alpha = 0.2, c = 0.8$). The best average NLL when $c = 1$ is 1.9457, which effectively ignores the memory and thus reduces to a non-memory-enhanced order-k Markov chain. Thus, even in its current, non-fuzzy form, the memory *does* appear to improve the average log-likelihood at least a bit. This best log-likelihood for $c = 1$ also occurs when $k = 2, \alpha = 0.2$, implying that these hyperparameters *might* be the overall optimal setting.

Figure 1 summarizes how average log-likelihood changes as a function of k for different settings of c (on both train and test sets). There are a few very interesting conclusions to be drawn from this graph. Training set log-likelihood continues to increase through roughly $k = 4$ and then declines, whereas test set log-likelihood begins to decline after $k = 2$. This indicates that, above $k = 2$, higher values of k lead to overfitting. This supports our original hypothesis that simply increasing k is insufficient for learning longer-term melodic structure, thus motivating our development of more sophisticated memory methods. Further, we compare the behavior for different values of c (noting that $c = 1$ is functionally equivalent to not using the external memory at all). We note that, on both the training set and the test set, $c = 0.8$ and $c = 0.6$ consistently outperform the other values of c (thus indicating a consistent benefit from using the external memory, as long as it is not relied upon *too* heavily). We see these results as encouraging validation of the utility of our external memory-augmented Markov model, and we would be excited to work on extending them even further in the future (e.g., using fuzzy tries, etc.).

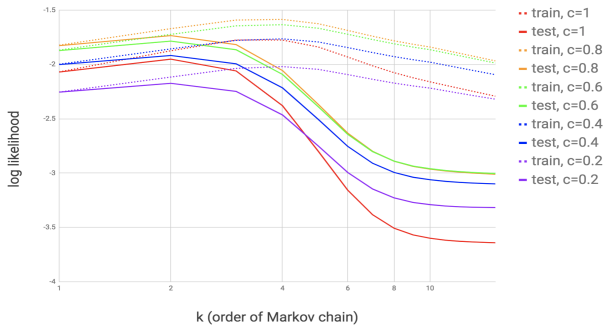

Figure 1: Average log-likelihood per time step for different values of k and c (all at $\alpha = 0.5$, chosen because it is in the middle of the range of α-values we explored).

5.2 LSTM - MIDI Pitch and Chords

Our baseline for comparison is a 1st-order Markov chain, with results displayed in Table 1. Note that this is different from the Markov chains in 5.1, as the state space here is much larger (incorporating chords in addition to just single-note pitches).

Note that for all LSTM-based architectures moving forward, we estimate the Uniform Random NLL as the performance of the untrained network. We train the model on the Nottingham and the Piano

MIDI datasets. The LSTM has 1250 hidden units and 2 hidden layers. We use cross-entropy as the loss function and ADAM optimizer for training the model. The obtained results are shown in Table 2

Dataset	Train NLL	Test NLL
Nottingham	1.86	2.02
Piano MIDI	2.1331	2.3805

Table 2: Results for the Language model with pitch and chords representation.

5.3 LSTM - MIDI Pitch, Offset, Duration

We train the model on the Nottingham and the Piano MIDI datasets. The LSTM has 512 hidden units and 3 hidden layers. We use cross-entropy as the loss function and ADAM optimizer for training the model. The obtained results are shown in Table 3. The training and validation losses as the network trains are shown in Figure 2.

Dataset	Uniform Random NLL	Train NLL	Test NLL
Nottingham	11.4543	1.2869	1.8775
Piano MIDI	15.0623	4.6857	5.6992

Table 3: Results for Pith-Offset-Duration representation. Note that the Random values are different for the two datasets because the NLL of the untrained network depends on the vocabulary size. The piano-midi dataset has a larger vocabulary size.

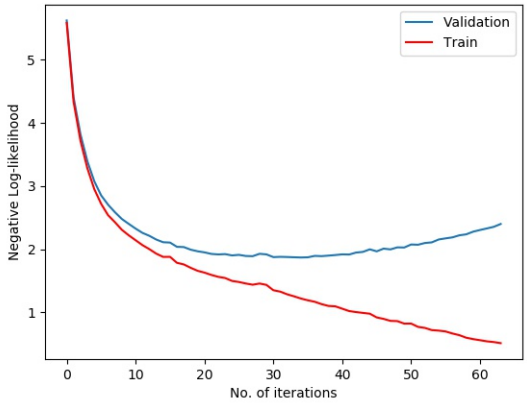

Figure 2: A plot of Training-NLL vs Time and Validation-NLL vs Time for comparison. Note that the network starts overtraining after a particular point. We use the model corresponding to the minimal validation NLL to generate music and test results.

5.4 LSTM - Binary Piano Roll

We train the model on the Nottingham and the Piano MIDI dataset. The LSTM has 256 hidden units and 3 hidden layers. We use cross-entropy as the loss function and ADAM optimizer for training the model. The obtained results are shown in Table 4.

5.5 LSTM - Pitch-Pattern Piano Roll

Once again, we train the model on the Nottingham and the Piano MIDI dataset. The LSTM has the same structure as the previous experiment except that it models a categorical distribution for the first note and a set of conditionally independent Bernoulli distributions for the pattern. Note that the number of Bernoulli distributions in consideration changes depending on the first note. Also note that

Dataset	Uniform Random NLL	Train NLL	Test NLL
Nottingham	61.2963	2.6052	3.8333
Piano MIDI	61.3057	6.7642	8.7459

Table 4: Results for Language model trained on Binary Piano Roll representation.

the Uniform Random NLL is different from the previous model not due to a change in the size of the state space, but due to the inherent bias introduced. This representation yields results as shown in Table 5

Dataset	Uniform Random NLL	Train NLL	Test NLL
Nottingham	41.9906	2.0020	3.1236
Piano MIDI	41.9929	6.7999	8.9946

Table 5: Results for Language model trained on the Pitch-Pattern Piano Roll representation.

6 Conclusion and Future Work

We have presented a thorough investigation of the problem of music generation and modeling using Markov models and recurrent neural networks and have presented various standard as well as novel representations and models for the same. We also present a novel variant of the traditional Markov chain which augments it with an external autoassociative memory component in the form of a trie. We showcase how this variant, even in its current form (with room for further extensions) performs quite well.

Furthermore, our experiments in Section 5 showcase how, given the same representations and state spaces, LSTMs perform significantly better than Markov chains both qualitatively and quantitatively. Our experiments with MIDI-based LSTM language models yielded the best numbers across all models and representations, and the model trained using the pitch-offset-duration representation generated the most subjectively pleasing music.

We also experimented with two different kinds of representations for the piano roll format: the naive binary representation, as well as a more sophisticated pattern-based representation. As hypothesized, the pattern-based representation quantitatively performs significantly better than the naive representation on the chord-rich Nottingham dataset. However, it performs marginally worse on the Piano MIDI dataset. We believe this discrepancy is due to the simplistic chord accompaniment present in the Nottingham dataset.

We also delineate a few possible directions for future efforts:

- The autoassociative memory augmentation of Markov chains has yielded promising results and is worthy of further exploration, even with more complex representations. Some other possible extensions using fuzzy tries, prefix arrays, and KMP string search can be explored.
- We find that the representations used make a significant difference in the subjective quality of the music produced. Therefore, we believe that more sophisticated representations are worthy of exploration.

Please see the following link for our generated music samples:
`https://drive.google.com/drive/u/1/folders/1pB2oYyZXnh9QptYn7w1ijpzZbSabIOPb`.

References

[1] Academia Sinica. Lakh pianoroll. https://salu133445.github.io/lakh-pianoroll-dataset/.

[2] Bernd Krueger. Classical piano midi. http://www.piano-midi.de/.

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

[4] CCARH. Musedata. http://www.musedata.org/.

[5] Colin Raffel. Lakh midi. https://colinraffel.com/projects/lmd/.

[6] Florian Colombo and Wulfram Gerstner. A general model of music composition. *arXiv preprint arXiv:1802.05162*, 2018.

[7] Michael Scott Cuthbert and Christopher Ariza. music21: A toolkit for computer-aided musicology and symbolic music data. 2010.

[8] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment.

[9] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Symbolic-domain music generation and accompaniment with multi-track sequential generative adversarial networks. *arXiv preprint arXiv:1709.06298*, 2017.

[10] Eric Foxely. Nottingham database. ifdo.ca/~seymour/nottingham/nottingham.html.

[11] Google. Melody rnn. https://github.com/tensorflow/magenta/tree/master/magenta/models/melody_rnn.

[12] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[13] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538 (7626):471, 2016.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[15] Allen Huang and Raymond Wu. Deep learning for music. *arXiv preprint arXiv:1606.04930*, 2016.

[16] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4-5):411–430, 2000.

[17] J. S. Bach. Allan harmony. https://github.com/johndpope/allan-harmony.

[18] Ji-Sung Kim. Deepjazz. https://github.com/jisungk/deepjazz.

[19] Vasanth Kalingeri and Srikanth Grandhe. Music generation with deep learning. *arXiv preprint arXiv:1612.04928*, 2016.

[20] I Liu, Bhiksha Ramakrishnan, et al. Bach in 2014: Music composition with recurrent neural network. *arXiv preprint arXiv:1412.3191*, 2014.

[21] Aran Nayebi and Matt Vitelli. Gruv: Algorithmic music generation using recurrent neural networks. *Course CS224D: Deep Learning for Natural Language Processing (Stanford)*, 2015.

[22] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.

[23] Ramesh Balaji, Tanmaya Dabral, Stefani Karp. Music-generation. https://github.com/many-facedgod/Music-Generation/tree/dev.

[24] Adam Roberts, Jesse Engel, and Douglas Eck. Hierarchical variational autoencoders for music. In *NIPS Workshop on Machine Learning for Creativity and Design*, 2017.

[25] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, page 125, 2016.