Enhancing Classical Music Generation with RNN

Alex Sokolov
sokolov@ucsd.edu

Vishaal Prasad
v2prasad@ucsd.edu

Patrick Liu
ppliu@eng.ucsd.edu

Thomas Andy Keller
takeller@eng.ucsd.edu

Abstract

Recurrent neural networks (RNN) have been used previously for purposes of creating music, a task that is complicated by the implicit complexities of the subject as well as limitations in RNNs themselves. Advances of RNNs, such as the advent of Long Short-Term Memory (LSTM), have sparked renewed interest in the capabilities of RNNs in learning music. In this paper, we examine different network structures and representations of music in an attempt to create ”better” classical music. We also explore the application of curriculum learning to music generation and observe that it improves the generated music according to multiple metrics. As comparing the quality of generated music is a subjective task, we also explore objective metrics that evaluate the equality of musical output. We discover that the best results stemmed from using complex networks with high computational capabilities and simplifying the learning task through steps such as curriculum learning and key normalization.

1 Introduction

Artificial creativity is a topic that follows naturally from artificial intelligence. In recent years, there have been many efforts in this subject exploring areas such as computer-generated art, handwriting, writing, and so forth. Within the arts, the sequential nature of recurrent neural networks (RNN) have found it utility in modeling music. The structure of RNNs allow for the output of a portion of the model to be its own input, which can be thought of as the model retaining a form of memory. The usefulness of this property in modeling and creating music should be intuitive – determining the next note to be played and how it is played requires a knowledge of the history of the piece or song up to that point in time. In theory, RNNs should be able to internally represent this history and use it to influence its output. However, this is an over-simplification from multiple standpoints.

Music itself has features that range from the fundamental (duration, dynamics, pitch of individual notes) to the more complex (tempo, chord progressions, phrasings, melodies). These complex features can vary both with or without interdependence and often are irregular; this gives music flexibility that allows for emotional expression, but makes the problem of modeling and generation significantly more difficult. These complexities require us to examine the limitations of RNNs. The vanishing or exploding gradient problem is a well-studied problem that arises when the number of layers of a neural network is increased and causes the potential for a signal to be lost during back propagation, complicating the training of these networks. To properly model music we may need to increase the complexity of our RNN, but this has its own costs.

Another difficulty is that the evaluation of music quality is largely subjective and can vary widely for any particular piece of music. Without a definitive standard for qualitatively or quantitatively evaluating a piece of music, it is difficult to judge the quality of generated music.

For this project we explore some ways of potentially addressing these issues, through work done by ourselves, and previous work and ideas by others for the purpose of music generation, as well as for other tasks which still can be applied to this subject. To better represent music, we experiment with two data transformations on the input space: one for the purpose of capturing some local structure by incorporating temporal information such as held notes, and one to change the ”goal” of model by reversing the time axis of the input.

To address the issue of model complexity versus vanishing and exploding gradients, we evaluate two separate models of RNNs: a simple dual-layer LSTM model for its long-term memory properties immune to vanishing/exploding gradients and a more complicated multiple layer LSTM model coined the ”biaxial RNN”. To improve results, we also experiment with pre-training through curriculum training and post-processing with a windowed RBM, described in more detail later.

Lastly, to address the problem of evaluating music, we reduce the problem from ”is the generated music good?” to ”does the generated music sound like the music used for training?”. To do this, we apply a quantitative measure based off of kernel density estimation (KDE) called Indirect Sampling Likelihood (ISL). With this approach, the probability of a held-out test set is computed under the probability distribution generated by the model, returning the log-likelihood of the test set. For models where the generative distribution is not explicitly defined, we can estimate it with kernel density estimation. While this is an estimation that has recently been shown to be removed from the true log-likelihood and occasionally favor samples generated from trivial generative models (sampling from k-means centroids), we hope that the ISL can provide us with a way to rank output. Lastly we perform PCA on measures taken from our training and generated music as a way of visual evaluation.

2 Methods

2.1 Data

While some research has been done into generative RNNs which work directly with raw audio recordings, and subsequently synthesize raw frequency-space output, we found the majority of prior work used more serialized forms of music such as MIDI. We decided to follow this direction, and therefore use the MIDI encoding for all our models' input and output files, in order to pursue our main goal of teaching the network larger-scale musical structure from rudimentary elements to complex elements, without requiring it to learn how to form or parse each note independently. The MIDI protocol includes information about the notation, pitch, velocity, volume, vibrato, and tempo of each note in a serialized form, giving our network access to all the basic building blocks of a musical piece.

We decided to focus our network training on classical piano and acquired training data from a variety of classical composers [12]. Our main training corpus therefore consisted of 256 MIDI piano compositions ranging in length from 30 seconds to several minutes. This also includes the data used to train one of the networks [7] studied in this project with satisfactory results, so we considered it to be comprehensive and sufficient for our purposes. Besides this, we have used the Nottingham dataset [14] and a set of easier classical pieces for curriculum training.

For our curriculum learning experiments, we needed a set of files which were significantly simpler in musical structure, and organized according to complexity. We found such a set of 80 MIDI files, grouped according to 4 levels of difficulty, at [13]. None of the pieces were complex compared to the proper train set, so they worked well for this purpose.

2.2 Data Transformations

Neural networks, despite the theoretical capability to be universal function approximators, work much better when the data is provided in a specific formats. Just like image data may be whitened, raw MIDI data can be ”normalized” by being put into the same key. This key step of pre-processing should improve output most significantly.

A further transformation is changing the input space to incorporate more information. Encoding the previous note value allows the network to more easily use its immediate temporal surroundings.

Encoding not just absolute pitch of the 88 possible notes on a piano but also the relative pitch in terms of pitch class allows a network to more easily learn chords. Further details can be found at [7].

It is also possible to limit the types of input coming in to facilitate training. Tonal music can broadly categorized into two primary types: major and minor. Though many minor pieces modulate to major at some point in their duration, and vice-versa, restricting a network so it can see mostly one of major or minor pieces in theory would allows the network to learn these underlying tonal structures more easily.

A final input transformation is reversing the training data. The teaching signal remains the next note in the (now-reversed) input sample; however, the temporal dependencies the network learns are on the opposite direction across the time axis. If it is then seeded with some input and allowed to compose, the reverse of its composition again follows the standard axis of time. This allows the network to build towards a seeded resolution, rather than hearing a network diverge from a given beginning. This should not change output similarity to the train data, but it is a new angle to evaluate the output music. It also may provide better subjective results. Most familiar Western music ends with a tonic ending. If that ending is seeded and guaranteed to occur that person may be more satisfied overall with the piece than otherwise, due to disproportionate attention being placed on the ending.

2.3 Curriculum Learning

For curriculum learning, we downloaded a set of classical music ranging from ”very easy” to ”hard” [13], but all of them were simpler than the complicated classical music of the main training set. Prior to training on the main training corpus, the models were trained on 500 iterations on each category, in order of difficulty. The goal of this was to emphasize basic musical structure in the model before training on the more complex pieces of the main corpus. The theory behind curriculum learning and an empirical analysis can be seen in more detail in [11].The number of iterations is simply a heuristic. We determined the number of iterations by the subjective quality of the final output of the trained network, as well as by the training error in an attempt to avoid overfitting.

2.4 Models

2.4.1 LSTM

As a baseline model, we implemented a basic LSTM network in Keras composed of two layers of 512 LSTM nodes each, separated by a dropout layer with drop-probability 0.2, and topped with a dense sigmoid activation layer. The input vector to the network was a one-hot encoding of midi-notes from 21 to 109 (the standard piano range), and thus had 88 ’features’. The output of the network was a similar length 88 vector, where each element denotes the probability that note should be played.

The network was optimized with RMSProp to minimize the categorical cross-entropy. The learning rate was set to 0.001, ρ was set to 0.9, and ϵ was set to 10^{-6}.

The sigmoid activation layer was chosen over alternatives, such as softmax, to allow the network to easily play multiple notes simultaneously (like in chords), without having to fight against a normalized probability.

The network was additionally trained and tested in a ’stateful’ manner, meaning that the state of the LSTM nodes was preserved between batches, allowing notes beyond a single batch to affect the output of the network. This is as opposed to the standard batch technique which limits the advantages of the LSTM units.

2.4.2 Biaxial RNN

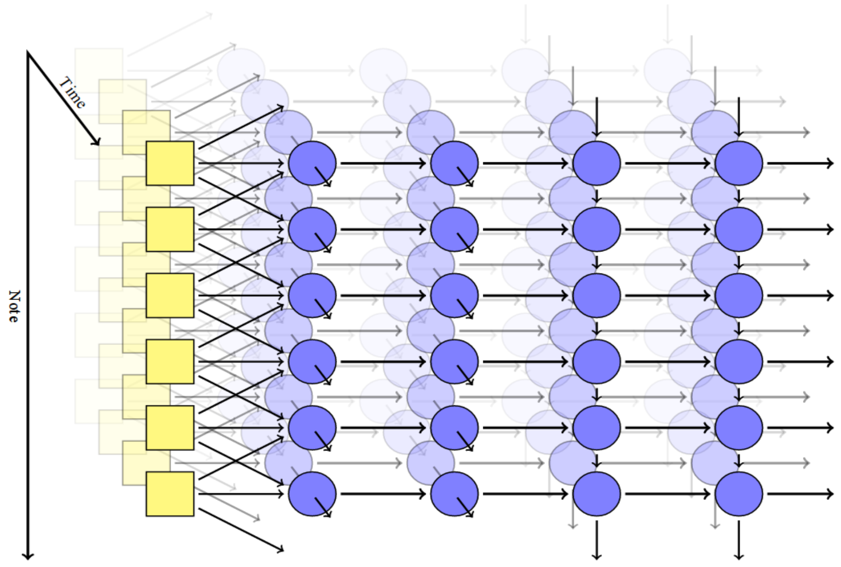
This was the best available implementation of piano music generation we were able to find and its results are impressive. The key advantages of biaxial compared to previous music generation implementations are:

1. Time-invariance - ability to compose indefinitely, being identical on each time step.

2. Note-invariance - to let music be transposed up and down with identical structure of the neural network for each note.
3. Allow multiple notes to be played simultaneously, and allow selection of coherent chords.
4. Allow the same note to be repeated at a single time step.

In this architecture each note has two LSTMs. The first has recurrent connections that connect temporally that takes an input matrix. Its output therefore contains a stack of note activations across time. This is fed, along with lateral information from its two neighbors, into the second LSTM which contains recurrent connections along the note axis. This, in turn, is sent feed-forward into a non-recurrent output layer which indicates whether a note should be played and also newly that is invariant both along time and note-space.

Figure 1: Note and time invariant architecture of biaxial RNN [7]



2.4.3 Postprocessing with RBM

Inspired by the lectures on Restricted Boltzmann Machines we were eager to embed it somewhere in our models. One of the ideas was that due to the RBMs proven ability to reconstruct slightly corrupted or incomplete data, we might be able to use it to make our networks output sound more like the training data, fixing small problems, like little dissonance or inappropriate note in a given context. Even better usage of it would be to train RBM on some variant of music, like major/minor pieces and try to convert input into that. This would be interesting post-processing.

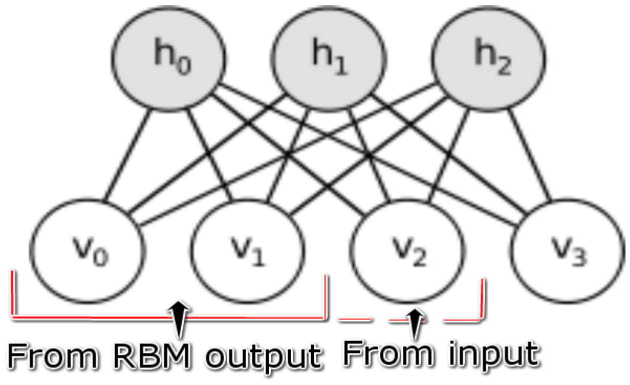
Since a simple RBM doesnt play well with sequences and we never bothered to try CRBM or TRBM, we have decided to try mapping the time into space, particularly a sliding window of 16 time steps. It would start from gibbs sampling whole first window of RBM visible units given the output sequence. And then slide one step further, taking 15 values from RBM recent output and 1 last value from the original output sequence. This way RBM will only have 1 value from unfamiliar distribution and would presumably reconstruct that. Then we reconstruct two outputs where we update either one step or the whole window per time step.

Unfortunately this ended unexpectedly - for any input it reconstructed the same sequence with very little deviation in the beginning.

2.5 ISL

To better quantify the impact of the different model architectures and data transformations on the outputs of our generative models, we decided to evaluate the validity of the Indirect Sampling Like-

Figure 2: Standard RBM with 2496 visible-and 1500 hidden units



lihood (ISL) metric, and other similar metrics, on generated music data. More specifically, for our compositions, the ISL was computed in the following manner:

1. The generative network is trained, and a series of samples is generated of size at least equal to that of the training corpus.
2. The PDF of the generated distribution is estimated using a kernel density estimator. (In our case we used the Python StatsModels Multivariate Kernel Density Estimator).
3. The log-likelihood of the held-out test set is then computed under this PDF, giving the ISL.

As can be seen, in theory, if a generative model is able to reproduce the test set exactly, then the likelihood of the test set under this distribution will be almost exactly 1.0, meaning the log-likelihood will be close to 0. For generative models which produce something which is far from the test set, the hope is that the likelihood similarly decreases. In the following sections we show a comparison of ISL to training error, and show it's computed values for a variety of model architectures.

3 Results

3.1 Curriculum Learning

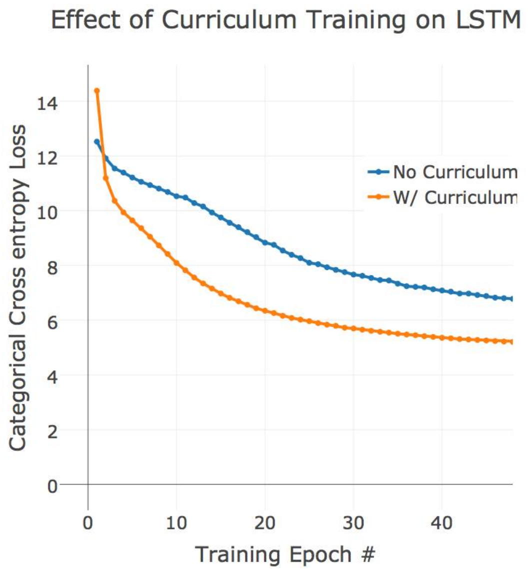
To evaluate the effects of curriculum learning, we compared the training error as a function of training iterations for a network which was pretrained with a desired curriculum, and one which was not. A plot of this error comparison for both the baseline LSTM model and the more complicated biaxial RNN is seen in Figure 3 below.

Further, we compared the lowest achieved ISL of a network which was trained without using curriculum learning (-400) with the ISL of a network which was pretrained with a specified curriculum (-377) and found a result that was in line with the training error observations: Curriculum learning results in better local minima and faster convergence. This is seen to be true for both recurrent network architectures we tested.

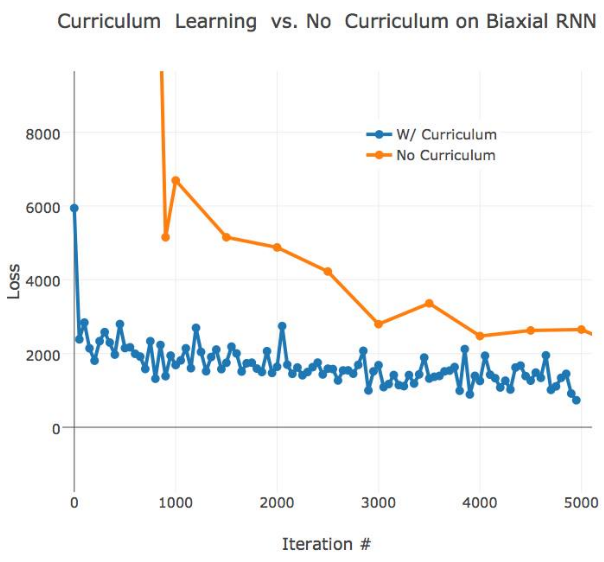
3.2 ISL Evaluation

To evaluate the validity of the ISL metric, we first verified that it decreased as a function of training iterations for our biaxial RNN, as can be seen in figure 4.

Although it can be seen that the ISL roughly corresponds to training loss, we more importantly wanted it to correspond to a subjective interpretation of quality of the music. To test this, we compared the ISL of our baseline LSTM model at a range of stages during training. The epochs we chose to evaluate our model at were 10, 30, and 70. These were chosen due to the fact that the note sparsity increased dramatically as the number of training epochs increased, leading to subjectively far inferior music quality. However, the ISL values actually contradict this subjective opinion (Table 1).



(a) Effect of curriculum training on baseline LSTM training error.



(b) Effect of curriculum training on Biaxial RNN training error.

Figure 3: The figures above clearly show the increase in the speed of convergence, as well as the lower ending local minimum obtained by training our models with curriculum as opposed to no curriculum.

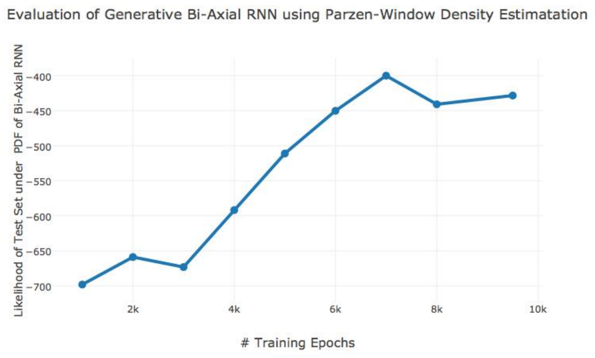


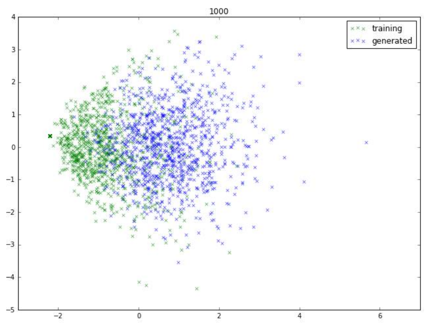
Figure 4: Indirect Sampling Likelihood of biaxial RNN as a function of the number of training epochs.

Epoch	**ISL**
10	-425.1
30	-249.9
70	-55.1

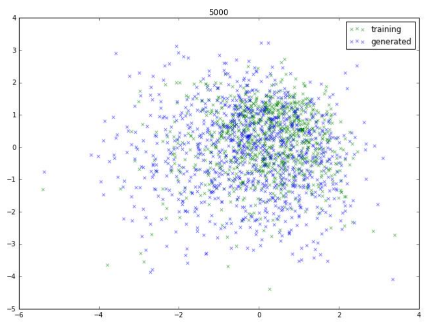
Table 1: ISL measures from various epochs of training of the LSTM model.

These do indeed correspond to the training loss, but the model which outputs mostly silence (70 Epochs), has dramatically lower ISL than any model we tested. We hypothesize that this is due to the sparsity of each of our input note vectors, and the fact that each note is considered independently in the ISL computation, causing music which is mostly silence to bear the most resemblance to the test set on a note-by-note level. This is a significant limiting factor for the ISL measure, and agrees with prior research indicating the severe drawbacks to using ISL. [10]

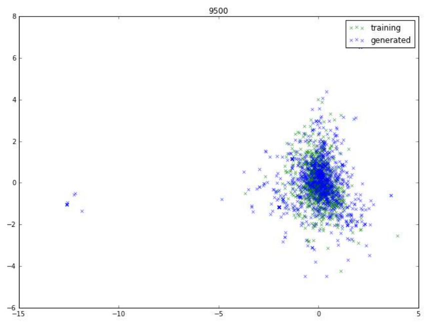
3.3 PCA



(a) 1000 Iterations



(b) 5000 Iterations



(c) 9500 Iterations

Figure 5: Projection of high-dimensional vectorized music measures from both the training corpus and generated music onto principal components 1 and 2 for 1000, 5000, and 9500 training iterations of the biaxial RNN model.

PCA of measures sampled from the corpus of training music and generated music from the biaxial RNN allow us to plot this high-dimensional data (Figure 5). With the resulting plots, we confirm visually what we concluded from listening to the generated music: the generated music sounds nearer to the music the model was trained on as iterations grow.

4 Discussion and Conclusions

As a preface to this discussion, we state that model performance and quality of the output was evaluated by ear and as a result is highly subjective, as mentioned and discussed in the introduction.

We find that unsurprisingly the more complex biaxial RNN outperforms our dual-layer LSTM model. The simple model became more and more sparse with its outputs as it trained. While it performed better than random, its outputs were not too coherent. The former model generally sounded pleasant to the ear, and was rather dense in terms of note output. It lost its path rather quickly and did not learn global structure. However, on a local level, it performed well.

Curriculum data improved the biaxial RNN's performance. As discussed prior, the network gave better error measures and ISL measures. The output was also noticeably different: in particular, the network, which trained on simpler music, also gave a more reasonable density of notes than it would do otherwise. It never reached the sparsity of the dual-layer LSTM but also avoided the overwhelming cascade of notes that the biaxial RNN without curriculum training sometimes outputted. This warrants future investigations using curriculum training. Curriculum training a network on frequent basic chords progressions might facilitate the network better learning underlying chords and provide a better harmonic foundation.

The ISL measure seems to generally correlate with the training loss, and with a subjective notion of quality given the generated output is not sparse. If the generated music is sparse, we see that the ISL measure breaks down and is no longer valid. We hypothesize this could be fixed with an alternaive density estimation technique or an alternative encoding for the music which is naturally less sparse per input vector.

Using the biaxial RNN as the baseline, we also experimented with changing the input music data.

To our surprise, the results of training the biaxial RNN on major corpuses solely did not improve network structure. In fact, the output, perhaps due to fewer training examples, sounded far worse. This bears investigating, as it is possible that beyond misclassified pieces, correctly classified pieces that switch tonality from major to minor or vice-versa also inhibited learning. Furthermore, it is possible the network simply got stuck in suboptimal minima, and that retraining the network would yield better results. Due to time restraints, we could not validate this ourselves.

However, subjectively the reversed output on the RNN with the opposite time-axis produced results that our group found to be more appealing than the original output. In particular, the network learned cadences with a higher degree of consistency. With a simple 9-measure output (roughly 10 seconds of music), the reversed network learned to end pieces with more resolution than the original network. More samples are needed to draw stronger conclusions, however. ISL did not show difference in terms of the musics similarity to the training corpus, which agrees with our hypothesis.

5 Contributions

Andy and Patrick wrote the code for implementing the dual-layer LSTM network. Andy wrote the code for calculating ISL. Patrick performed the PCA. Vishaal organized the dataset into component subdivisions for experiments and wrote the code to allow for curriculum training. He also wrote the code to normalize MIDI files into C Major or A minor. Alex ran and compared performance of biaxial network on various combinations of the input and models parameters, speed up the Theano implementation, and implemented windowed RBM for reconstruction. All authors worked on the report together.

6 Acknowledgements

This project was completed with input from the TAs of CSE 253. We would also like to acknowledge user ”nlaeae” from the machine learning forum on the website Reddit for the idea of reversing music input.

References

[1] Breuleux, O., Bengio, Y., and Vincent, P. (2011). Quickly generating representative samples from an RBM-derived process. Neural Computation, 23 (8), 20532073.
`http://www.iro.umontreal.ca/ lisa/pointeurs/breuleux+bengio_nc2011.pdf`

[2] Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In Proceedings of the Twenty-nine International Conference on Machine Learning (ICML'12). ACM.
`http://www-etud.iro.umontreal.ca/ boulanni/ICML2012.pdf`

[3] Eck, D., Schmidhuber, J. (2002) A First Look at Music Composition using LSTM Recurrent Neural Networks. Technical Report No. IDSIA-07-02
`http://people.idsia.ch/ juergen/blues/IDSIA-07-02.pdf`

[4] Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. (2014) Generative adversarial nets. In NIPS 14, 26722680.
`http://arxiv.org/abs/1406.2661`

[5] Python MIDI library
`https://github.com/vishnubob/python-midi`

[6] Generating Sentences from a Continuous Space
`http://arxiv.org/abs/1511.06349`

[7] Classical music generation with RNN implementation
`https://github.com/hexahedria/biaxial-rnn-music-composition`

[8] `https://www.reddit.com/r/MachineLearning/comments/3gaosx/composing_music_with_recurrent_neural_networks/ctwr9dt`

[9] KernScore classical music database
`http://kern.humdrum.org/`

[10] Theis, L. van den Oord, A. Bethge, M. (2016) A Note on the Evaluation of Generative Models (ICLR '16)
`http://arxiv.org/pdf/1511.01844v2.pdf`

[11] Yoshua Bengio , Jrme Louradour , Ronan Collobert , Jason Weston, Curriculum learning, Proceedings of the 26th Annual International Conference on Machine Learning, p.41-48, June 14-18, 2009, Montreal, Quebec, Canada
`http://ronan.collobert.com/pub/matos/2009_curriculum_icml.pdf`

[12] Piano midi files source.
`http://www.piano-midi.de/`

[13] Set of progressive difficulty chords for Curriculum learning.
`http://www.freesheetpianomusic.com/`

[14] Nottingham dataset of folk music
`http://abc.sourceforge.net/NMD/`