Towards Music Generation With Deep Learning Algorithms

Marko Docevski, Eftim Zdravevski, Petre Lameski, Andrea Kulakov
Faculty of Computer Science and Engineering
Sts. Cyril and Methodius University, Skopje, Macedonia
E-mails: mdocevski@gmail.com, {eftim.zdravevski,petre.lameski,andrea.kulakov}@finki.ukim.mk

Abstract—**Computer music generation has application in many areas, including computer aided music composition, on demand music generation for video games, sport events, multi-media experiences, creating music in the style of passed away artists, etc. In this work we describe our approach towards music generation. We trained a deep learning model on a corpus of works of several authors. By priming the model with a snippet of an authors work we used it to create new music in their style. The dataset consists of music for guitar in midi format, containing only 1 part/instrument. We gathered more than 2000 files, of which we used from 5 to 300 per experiment. The data for the deep learning model is represented in piano roll format, a binary matrix where one axis represents the time and the other axis represents midi notes. Two deep learning architectures were evaluated, a 2-layer recurrent neural network of LSTM (Long Short Term Memory) cells and an Encoder-Decoder (Auto-Encoder) architecture for sequence learning, where both the encoder and decoder are built as recurrent layers of LSTM cells. The models were implemented in the Keras deep-learning library. The results were evaluated on a subjective basis, and with the evaluated datasets both architectures produced results of limited quality.**

Index Terms—**music generation, midi, deep learning, recurrent neural networks, LSTM, auto-encoder**

I. Introduction

Computer music generation is an aspect of computer/artificial intelligence that has peaked interest in recent years. Opinions are split on whether computer generated music is just imitation of human work or if computers can show signs of actual creativity [1]. The opponents of computer creativity claim that no computer model can create anything novel because it would be based on a corpus of human work or human defined rule-set, which it would only recombine in arbitrarily abstract or complex way. On the other hand, most human musical composition is achieved in a similar way. People are exposed to music long before they can contribute their musical compositions, and therefore heavily influenced by other works, either consciously or subconsciously. By mimicking the human creative process and leveraging the ability of modern computers to process massive amounts of data, and inspired by the, by now legendary, article [2] by Andrej Karpathy, we would like to contribute to the discussion with our experiments with deep learning models and try to show computer creativity.

Other than adding to the philosophical discussion, we see practical applications for a music generation system. Exemplary applications include:

- Video games music, i.e., procedural generation of music based on the setting and gameplay tempo.
- Exercise music, e.g. cardio workouts, where the music's rhythm is synchronized to the users vitals and/or an exercise tempo scheme.
- Computer aided music composition, where the model would help human composers by "filling in the gaps" with candidate sequences of which the user would select the most appropriate, to augment their work or give them inspiration.
- Creating new music in the spirit or style of passed away musical performers and actors.

However, any attempt of creating a deep learning model for computer musical creativity is severely limited by the lack of a way to objectively evaluate the model. The most common approach is subjective evaluation of the results by a human, usually qualified, with musical training and experience. This not only limits evaluation of the final results, but also limits the selection and training of deep learning algorithms and can prove a severe bottleneck.

We decided to limit the scope to training models on one track/voice of polyphonic music, with an arbitrary sequence of notes or chords. We built the dataset from MIDI files due to their availability and ease of parsing.

In this paper we provide details of two experiments that we conducted, a multilayer Long-Short Term Memory (LSTM) Recurrent Neural Network (RNN) and feed-forward network, based on [3] and [4]; and a LSTM based Encoder-Decoder architecture as baseline models based on [5]; as well as outline how we aim to improve them and other experimental architecture we intend to use in future works.

The paper is organized as follows: in section II we give a brief overview of literature in the field of computer music generation; in section III we describe how we obtained the dataset, how it was processed and in subsection III-A the data representation we chose for the learning model. Then, in section IV we describe the deep learning architectures we used in the experiments, in section V we discuss and interpret

This work was partially financed by the Faculty of Computer Science and Engineering at the Sts. Cyril and Methodius University, Skopje, Macedonia. We also acknowledge the support of Microsoft Azure for Research through a grant providing resources for this work.

the obtained results, in section VI we outline how we intend to improve the current experiments and the architecture we plan to test out next, and finally in section VII, we conclude this paper.

II. Overview of Similar Approaches to Music Generation

This work was inspired by the legendary article by Andrej Karpathy - The Unreasonable Effectiveness of Recurrent Neural Networks [2]. In it Karpathy shows how a neural network consisting of 2 layers of 512 LSTM cells can generalize over several datasets of textual data: Paul Graham essay, Shakespeare plays, XML and Linux C/C++ source code. The resulting model can generate new sequences that are reasonably close to the source material and can almost fool a human, and in the case of C/C++ code some of the results compile. It is a character-level language model, i.e., it learns the texts letter by letter. Our first experiment was to adapt this model to the task of music generation.

Eck et al. in [6] describe the first application of deep learning to music generation. Due to the limited computational power at the time they defined a very simple experiment, generalization over 12 bar blues in written form, with a maximum of 8 notes per bar. They represented music in slices of note-time, i.e., one unit of time is $\frac{1}{8}$-th note long. This representation was later named as piano roll. The more complex of their two experiments consisted of learning in parallel a chord progression and a melody line, allowing them to generate both. Their architecture consists of 8 blocks of 2 LSTM cells, of which 4 are dedicated to learning melody the other 4 to learning the chord progression. The chords sections have output connections that influence the melody section but not vice versa. This work is the inspiration for most of the following research in the filed, including our own.

As an expansion to the previous work in [7] Eck et al. improve their model by using musically important temporal information from the pieces, allowing it to learn structure at different granularities. The said information is metrically significant and is provided to the network as time-delayed copies of the network inputs at intervals that depend on the piece's meter. For pieces where no meter information is provided, the authors developed a technique for algorithmic meter extraction. The architecture consists of parallel LSTM layer and standard feed-forward layer, which is supposed to speed-up local dependency discovery. We plan to investigate how and whether having metrical information embedded into the learning model impacts performance.

The architectures presented in [8] and [3] are specifically designed to tackle Bach chorales. A hybrid architecture was presented in [8] consisting of 2 LSTM subnets, of which one learns the sequence in standard order and the other in reverse order and a feed-forward subnet that learns the sequence in standard order. The three subnets are combined in a single feed-forward network. There is a copy of the architecture for each voice in the chorales. The music is generated by pseudo-Gibbs sampling procedure. They treat the 4 voices separately, as monophonic music generation problems, and combine them into a polyphonic melody. In contrast, we aim to generate mixture of chords/single notes from a single network output. To augment the dataset they transposed all the chorales to all transpositions that fit in the tonal range present in the corpus. We intend to do similar transposition schemes. The authors of [3] approached the same problem with a 3-layer LSTM network, to which they applied Grid-search hyper-parameter tuning extensively to find optimal network and parameter configurations. They used a very different data representation, encoding all the voices in a single sequence, ordered by descending pitch.

Boulanger-Lewandowski et al. in [9] attempted to combine the RNNs' ability to model sequences and Restricted Boltzmann Machines' (RBMs) ability to model probability distributions and apply them to music generation, in an architecture dubbed RNN-RBM. In this architecture the RBM part models note co-occurrence probability distributions at each time-step and the RNN part models the temporal sequences. We aim to achieve polyphony by modeling the probability of each output of the network independently of one another, even though realistically they are not, and try to avoid using a hybrid solution like RNN-RBN, due to it being difficult to train and computationally expensive.

In [4], another hybrid deep learning model, the LSTM layers try to learn the temporal dependencies in the music, followed by a set of feed forward layers that try to learn note co-occurrences (or the harmonic part). They used a piano roll very similar to our own, enriched with additional meta-data extracted from the pieces.The meta-data depends on the music being of very regular form, which our dataset is no, so we did not include any.

The work presented in [10] approaches music generation as a word-level language model, where each unit (composed of 1 to 4 bars of music) is treated as word. The words are embedded in a descriptor vector. A library of word (or units) is then created from the complete set of embeddings. A LSTM network learns sequences of the embeddings. Music is generated by first sequencing embeddings, which are then decoded to actual note sequences by selecting the most appropriate representative out of the library of units. We used character level embeddings in out second experiment.

The implementation presented in [11] is very close to the one of [2]. The authors transformed a set of folk songs in ABC notation into a modified version of the ABC notation that's more suitable for deep learning, that makes it very easy to unambiguously distinguish the various types of tokens, dubbed folk-rnn. On top of that they built a character-level model RNN deep learning model. Those tokens were encoded in one-hot vectors fed to 3 layer LSTM network, which outputs probability distribution over the unique token set. The authors of [12] used an Auto-encoder architecture to learn chord latent space embeddings. A sequence learner model generalizes over sequences of latent space embeddings. They presented a comparison of several types of chord representations. One of the compared representations is very similar to what we used in our experiments. We discuss more in section VI.

III. MIDI Music Dataset

We created a dataset out of MIDI music files of classical guitar music, scraped from http://classicalguitarmidi.com. We chose MIDI files as the source format due to their ready availability and the variety of music available on the Internet in MIDI format. There is a number of formats that convert to MIDI, like Guitar Pro files. MIDI computer-human interfaces can allow us to extend the system to accept direct human input in a relatively straightforward way. A total of 4009 pieces were scraped, which we then filtered by number of midi tracks per file to pieces with only 1 track, resulting in a set of 2158 unique pieces from more than 70 authors (e.g., Aguado, Bach, Carcassi, Duarte, Guiliani, Mertz, Paganini and Segovia), while some files contain no author information.

The total length of all pieces is 336,878 seconds, the shortest piece is just 9 seconds, the longest 1,546 seconds, and the average length of pieces is 84 seconds. A histogram of max polyphony per piece can be seen in Figure 1. Most pieces fall between 4 and 6 notes playing at the same time, which was expected, however there are pieces where more than 6 notes are playing at the same time. This means that either more than one guitar track was encoded on the same MIDI track, or some pieces contain music played on different instruments and were mistakenly uploaded to the web site or were scraped by mistake. As seen in Figure 2, which shows tempo information from the dataset, most pieces fall in the 100 to 240 tempo range, which covers all classical tempos. Figure 3 shows the distribution of note lengths in multiples of quarter note, showing that the overwhelming majority of notes is of expected durations from $\frac{1}{16}$-ths to whole notes. However a significant number of notes are of very long duration, e.g., more than ten whole note durations, which is unusual for guitar, and is probably an instrument that can sustain a tone for longer durations, or an effect that the MIDI file author wanted to achieve and not a part of the original composition. A note range between MIDI pitch 32 (G#1) to 98 (D7) is present, which also suggests that the dataset contains parts with different instruments, as a 24 fret guitar can achieve the highest note E6 or MIDI pitch 88.

A. Data Representation for the Deep Learning Models

We chose to use piano roll representation for the data. A piano roll, in its most basic form, is a binary 2 dimensional matrix, where one dimension represents linear time, and the other the set of all possible notes, usually ordered by pitch. A non-zero value indicates a certain note played at a certain time. We used multi-hot encoding to allow for polyphony. It is the most commonly used data representation in this field, as seen in [4], [6]–[9], [13]. However we approach the piano roll transformation from a different perspective. In the other works selection and regularization steps were taken to obtain datasets consisting of 4/4 meter [4], [6]–[8] and the tempo was either ignored or used as a filtering parameter. The musical sequences were discretized according to note time, to the smallest possible note duration preset in the dataset, as seen in [6]–[8], or a preset minimal note duration, where smaller notes were simply ignored as seen in [7]. We decided to use real time instead of note time in discretizing the musical pieces, at a resolution of 16Hz (16 samples a second, i.e., 1s of music is represented as as a sequence of 16 elements). We chose this sampling frequency so we can represent $\frac{1}{16}$-th notes at a very fast tempo of 240 bpm, when the notes are properly aligned to $\frac{1}{16}$-ths of a second. This sampling technique effectively captures the vast majority of note sequences in the dataset, however it is not lossless. Notes that lie on several slices can

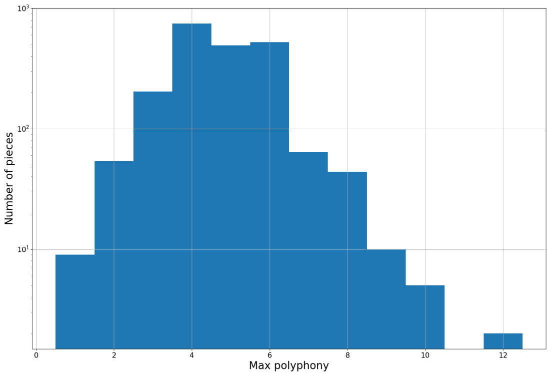

Fig. 1. Histogram of Max Polyphony.

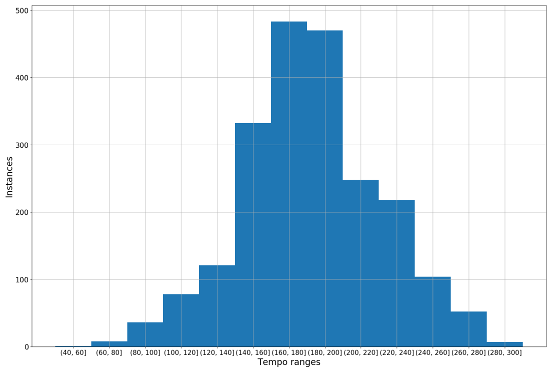

Fig. 2. Histogram of Tempos.

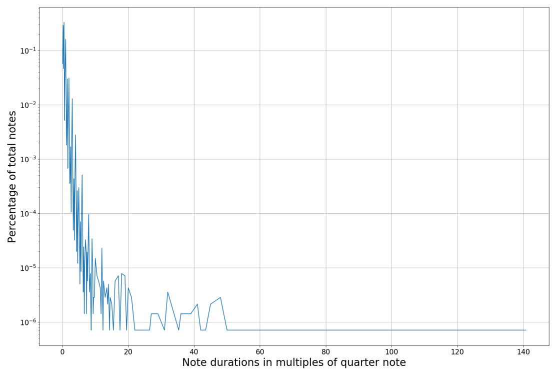

Fig. 3. Distribution of Note Durations.

be extended or contracted, or disappear entirely, depending on how far they span in the beginning and end slices. This leads to perceptive jitter/stutter when music is converted back to midi form for fast and dynamic pieces. Just like in the aforementioned works, our version of piano roll has no explicit note ending representation, which sometimes leads to several consecutive notes of the same pitch blending into one longer note. Since only a subset of the MIDI note pitches were present in the dataset, the range from MIDI pitch 32 to 98, we ignored the part of the midi specter that did not appear in any piece. MIDI files also contain velocity information for the notes played, which can be used in the piano roll representation, but we chose to ignore them as characteristics of performance and not composition, and instead focus on generating correct musical sequences.

IV. Experimental Architecture

We used the Keras deep learning library with the Tensorflow backend to implement our models. We conducted experiments with two types of architectures in mind. On the basis of the BachBot architecture presented in [3] and the architecture presented in [4], we tried 3 variations of a sequence learner:

- 2 recurrent layers with 256 LSTM cells with 0.2 dropout followed by fully connected output layer
- 3 recurrent layers with 1024 LSTM cells with 0.4 dropout followed by a fully connected output layer. Very close to the implementation in [3].
- 3 recurrent layers with 1024 LSTM cells with 0.4 dropout followed by 2 fully connected feed-forward layer, the second being an output layer. The second feed-forward fully connected layer was inspired by [4].

The activation function of the output layer is sigmoid, the model was trained with the ADAM optimizer using binary cross-entropy loss function. In Keras parlance the models were not stateful, relying on the Truncated Back-Propagation Through Time (TBPTT) to provide the network with the ability to learn sequences. We tried values between 16 and 64 timesteps for TBPTT (or between 1 and 4 seconds of look-back). The more timesteps the more context the model has, but it also becomes both harder to train, needs more training time to minimize training loss and requires more resources. We chose 16 timesteps as the value for TBPTT timesteps.

The other architecture we tried is inspired by the one described in [5], a Encoder-Decoder technique for sequence to sequence learning, where we limited the output sequence to be of length 1. We tried the following configuration:

- An encoder of one LSTM layer of 256 cells, a decoder of one LSTM layer of LSTM cells, with a fully connected output layer with sigmoidal activation.

This model was also trained with the ADAM optimizer using binary cross-entropy loss function. Each of the outputs from the final layer is treated separately, as if its activation probability is statistically independent from the rest, even though that is not strictly true. An output is considered active if the sigmoid activation value is greater than 0.5. This allow us to have multiple outputs activated at each time-step, allowing for polyphony, just like in the "bilinear model" defined in [12].

V. Results and Discussion

Training was done on a desktop PC equipped with an Intel i7 processor, 32GB of RAM and a GTX 750Ti NVidia GPU with 2GB of VRAM. All models were trained up to 100 epochs or until they did not improve on the loss function for more than 5 epochs. We used a small subset of 5 pieces from the whole dataset to do a baseline comparison between the model configurations. The pieces were chosen by familiarity to the experimenters to give us the ability to spot elements and patterns in the generated data. They were transformed to piano roll representation at a sampling frequency of 16Hz and all of them were concatenated into a single sequence with no special symbols for start or end of subsequence. With an input sequence length of 2 seconds, or 32 time slices, we got 26336 samples for training. No data was withheld for validation. The longest training time per epoch was 12 minutes for the most complex variant of the first architecture. Generation is done in a iterative fashion, i.e. a feed-forward generation strategy. We start by giving the models a primer, a 5 second sequence from the start of each of the pieces. The network is fed from the primer one sample at a time, discarding the predictions until the original primer boundary is reached. Afterwards the predictions are concatenated to the primer. We generate sequences of fixed length of 60 seconds. An overview of the results is given in Table I. An example generated sequence visualized in it's piano roll representation is shown in Figure 4. This is the best case result we have achieved, where the model repeats a subsequence several seconds in length. It was generated by the simplest model.

TABLE I
Overview of the generation results by architecture

Architecture	Result
2 layer LSTM	Up to 2 seconds of valid sequence of polyphonic music followed by either a constant polyphonic tone, a repeating pattern or silence
3 layer LSTM	Silence
3 layer LSTM and 2 feed-forward layers	Silence
Encoder-Decoder Architecture	A constant tone of several notes for the whole duration of the generated sequence

One limitation that affects all models is the character of the data. The music is not structured or very similar from piece to piece. That coupled with the polyphony make it hard for the model to generalize over the dataset.

The 2 layer LSTM model is limited in expressiveness due to the small network capacity and over-fits the data. It learns small sequences extremely well and tends to get stuck in loops when they get activated from the primer. In other cases it doesnt get activated at all and results in silence.

The 3 layer LSTM model on the other hand, is too expressive and does not have enough quality data to learn, so it never gets activated at all. This was clear from the training process,

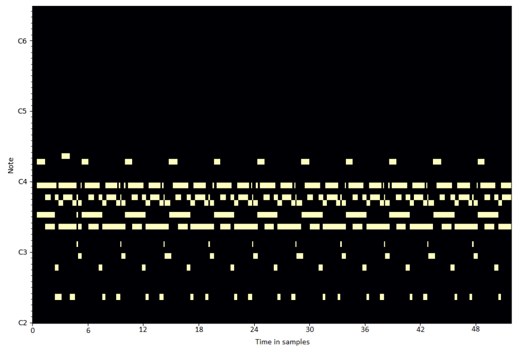

Fig. 4. Exemplary generated sequence.

where the 3 layer LSTM model plateaued much more quickly and at a higher loss value than the 2 layer LSTM model. The model with 3 LSTM and 2 feed forward layer had the same issues as the regular 3 layer LSTM model.

The Encoder-Decoder model had the same problems as the 2 layer LSTM model, however it has a more limited arbitrary memory capability when compare to the pure LSTM model. This leads to over-fitting even shorter sequences and therefore gets stuck repeating the same note. According to [12], the way we structured the outputs, with multi-hot encoding and independence assumption of the output probabilities, the model is expected to perform poorly compared to one-hot encoding or more advanced encoding schemes.

To get a time estimate for training we ran the simplest variant of the first architecture with a subset of 300 randomly chosen pieces from the dataset. An epoch took 52 minutes to train, which would take the model several days to be fully trained. This proves impractical for training models on the complete dataset.

VI. Future Work

We suggest the following ways in which the results could be improved:

- Adjust the parameters of the models, the layer sizes, activations, dropout values, input sequence sizes. Investigate applicability of Grid Search, as used in [3], or other optimization algorithms and techniques [14].
- Augment dataset with transpositions to every key present in the dataset. This is a technique used in [8], [10], and [13]. It will result in providing more samples that are musically correct, as well as sequences with the same relative pitch movement, allowing the models to better capture patterns across the dataset.
- Create a more efficient training method as well as an efficient batching method so that we can train the models on much larger subsets of the dataset.
- Try a different data representation as seen in [3] or [13].
- Try an architecture that learns to embed the input data into a latent space, i.e. chord embedding, and then have a multi-layer LSTM model learn embedding sequences similar to [12].
- Transform the problem to univariate predictions to greatly simplify the learning difficulty, like in [13].

VII. Conclusion

The work so far did not fulfill the set goal of generating a 60 second long sequence of polyphonic music. We discussed our interpretation of the limitations of the models we used in section V. They need further refinement before being able to generate actual musical sequences. We gave a brief overview of how the results could be improved. In section V we also noted that training on the complete dataset is not practical on the current hardware. However thanks to Microsoft Corporation, we have received a grant for Azure Cloud for use with machine learning, which we intent to leverage after refining the architecture.

References

[1] F. Ghedini, F. Pachet, and P. Roy, “Creating music and texts with flow machines,” *Multidisciplinary Contributions to the Science of Creative Thinking*, pp. 325–343, 2015.

[2] Andrej Karpathy, “The Unreasonable Effectiveness of Recurrent Neural Networks,” 2015. [Online]. Available: https://karpathy.github.io/2015/05/21/rnn-effectiveness/

[3] F. Liang, M. Gotham, M. Johnson, and J. Shotton, “Automatic Stylistic Composition of Bach Chorales with Deep LSTM,” *Proceedings of the 18th International Society for Music Information Retrieval Conference*, pp. 449–456, 2017.

[4] D. D. Johnson, “Generating polyphonic music using tied parallel networks,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10198 LNCS, pp. 128–143, 2017.

[5] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” 2014.

[6] D. Eck and J. Schmidhuber, “A First Look at Music Composition using LSTM Recurrent Neural Networks,” *Idsia*, pp. 1–11, 2002.

[7] D. Eck and J. Lapalme, “Learning musical structure directly from sequences of music,” *University of Montreal, Department of Computer . . .*, no. 1983, pp. 1–12, 2008.

[8] G. Hadjeres, F. Pachet, and F. Nielsen, “DeepBach: a Steerable Model for Bach Chorales Generation,” 2016.

[9] N. Boulanger-Lewandowski, “Modeling High-Dimensional Audio Sequences with Recurrent Neural Networks,” p. 145, 2014.

[10] M. Bretan, G. Weinberg, and L. Heck, “A Unit Selection Methodology for Music Generation Using Deep Neural Networks,” pp. 1–13, 2016.

[11] B. L. Sturm, J. F. Santos, and I. Korshunova, “Folk Music Style Modelling By Recurrent Neural Networks With Long Short Term Memory Units,” *Ismir*, 2015.

[12] S. Madjiheurem, L. Qu, and C. Walder, “Chord2Vec: Learning Musical Chord Embeddings,” no. Nips, 2016.

[13] C. Walder, “Modelling Symbolic Music: Beyond the Piano Roll,” 2016.

[14] P. Lameski, E. Zdravevski, R. Mingov, and A. Kulakov, “Svm parameter tuning with grid search and its impact on reduction of model over-fitting,” in *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*. Springer, 2015, pp. 464–474.