Contemporary AI Topics
Computational creativity for music

The influence of structure, datasets and other parameters on machine understanding of music

Stijn Janssens, Tristan Van Thielen

The goal of this project was to get a deep technical understanding of the field of music generation and investigate the influence of structure in compositions on the machine understanding of the music. Generative Adverserial Networks (GAN), Restricted Boltzmann Machines (RBM) and Long Short Term Memory networks (LSTM) were investigated and the latter was chosen to conduct the experiments. Different styles of music were considered based on the amount of musical structure and logic present. Afterwards, the influence of the sequence length on the coherence of the generated piece was investigated. In this report we will discuss shortly what our intentions and ideas were and the way we proceeded to execute these tasks.

Introduction

Music is a topic that has always been regarded as creative. These creative fields are tempting when trying to achieve artificial intelligence. Because of this, music generation is a highly researched subject with projects like Google's Magenta (1) and multiple fully computer generated albums as proof.

A difficulty specific to this domain is the definition of good music. This is not only highly subjective and differing from person to person, but correct and well structured music does not always mean that it is enjoyable. For this reason a popular way of judging generated music is a Turing test for music. If a piece sounds like it could be generated by a human being, then the result is considered good.

The rest of this report is structured as follows. First, an overview is given of the state of the art fields and techniques in music generation. Afterwards, the technique that was used for the experiments is explained more in depth. The text will conclude with the experiments that were conducted and their results.

State of the art

To start of this project, we read the overview paper by Colton and Wiggins (2) to get a grasp on the state of the art of this field, although interesting for the concepts behind computational creativity, this paper did not go into detail for music composition and was therefore not really useful. Diving deeper into the sub-field of music generation led to the papers by Hilscher and Shahroudi (3) and Eck and Schmidhuber (4) on understanding MIDI datasets and using LSTM networks for generating computer compositions. We found three main techniques for generating music from MIDI files, as discussed below. On the evaluation moment we were told to focus on just one of these. We chose LSTM networks to perform our experiments. The other techniques that we encountered were GANs in the paper by Mogren (5) and RBMs in the paper by Boulanger (6). We read the last two papers but did not go into detail on the exact implementation here.

The introductory slides on musical creativity by Toivanen (7) provided us with a clear overview of the field, but unfotunately did not lead us to any interesting papers, as the provided references only handled on the subject of algorithmic composition, which was not our intended field of focus.

We decided to talk about the different categories of music generation in our presentation, because leaving it out would not give a clear overview of the advances in this field. The research for these other categories led us to tools like Nyquist (8) for algorithmic composition and a quick look at what could be done with it and the syntax of this programming language in Simoni and Dannenbergs book (9) out of pure interest. Another field, computer aided music generation, was well represented in Google's Magenta project (1).

We chose to only focus on the the third category of musical creativity: automated stand-alone music generation. Within this third field there are many techniques that have proven successful. Most of these techniques are based on some variation of neural networks. Three of these were investigated more in depth. These are Generative Adversarial Networks (GANs) (5), Restricted Boltzmann Machines (RBMs) and Long Short Term Memory networks (LSTMs).

Generative Adversarial Networks are based on two different networks that interact during training. A generator network is tasked with generating music that is

indistinguishable from the given train data. On the other hand, the discriminator is tasked with discriminating between generated data and training data. *"The training becomes a zero-sum game for which the Nash equilibrium is when the generator produces data that the discriminator cannot tell from real data"*. (5) In practice however, the discriminator's task is too hard when applied to music generation. This was proven when we ran experiments with different datasets on the GAN based music generation github by Mogren (10). Experiments were conducted using the Final Fantasy and Bach databases, but in both cases the GAN did not achieve good performance or results. The generated music was very incoherent and sounded a lot like random notes. This problem seemed to also have occurred in the results of the paper mentioned above. Results of this paper can be found at (11).

A standard Restricted Boltzmann Machine consist of two layers. One input layer and one hidden layer. Contrary to a normal neural network approach, the RBM uses the joint-probability distribution in the prediction it makes. The hidden layer is used as a representation of the latent factors (for key of the piece, styles, ...). The first step in prediction is going from input to hidden layer by sampling from a Bernouilli distribution. The second step uses the new values of the hidden layer to go from hidden to the input space. The newly given input vector is the prediction. A very interesting approach was taken in (6). Restricted Boltzmann Machines were combined with Recurrent Neural Networks to achieve very good performance. Results can be found here (12). This implementation was however a bit too complex for implementing our experiments.

Creative part

Our creative part is a combination of some of the ideas suggested in the Wiki for this course and will be discussed below. We ran a set of experiments in order to create a meaningful demo. These experiments were executed on already existing github code, but we made some modifications to the code.

Chosen technique. To execute the experiments we wanted to conduct, we searched for an existing github project that could generate computer compositions in the style of a midi database without the need of a human input. We encountered (and tested) many github projects, but most of them either did not work, where needlessly complicated or did not use LSTMs like we wanted. In the end we chose a github project by Sigurður Skúli: Classical Piano Composer (13). One of the reasons we have chosen this project is because of the excellent article that was published on Medium (14). Unfortunately lots of interesting projects where found online that had better published results compared to the github we used, but they did not publish their code.

However we did not want to just completely copy the github project without making some (minor) changes to the code, aside from conducting the experiments. The project is written in Python3 using the Keras framework (15), a popular deep learning framework. The optimize that was originally used in the project is RMSprop, we changed this to Adam, both because the creators of Adam say it works better and faster and because we empirically discovered that Adam indeed was faster for our problem. We also increased the batch size of the training to 512, because we had the availability of GPU's to train on. This made it much more feasible to run experiments with large datasets. The last thing we did was switch the dropout layers of the model to BatchNormalization layers, these layers add normalization to the network which allows it to train faster whilst still allowing some regularization (which dropout is for) to occur. For one of the experiments we did more changes to the network as can be read below.

Experiments. The experiments can be divided into 3 sets of individually conducted experiments. First we will discuss our experiments regarding the influence of style and structure of a database on the resulting generated piece. Second, the influence of the parameter "sequence length" is researched. Lastly, we performed a new experiment by trying to incorporate an embedding layer into the network. More information can be found in the subsections below. The different experiments we wanted to perform for our creative part where decided on fairly quickly and from then it was just a question of thinking of a good dataset and collecting it so that it could show of the desired effects.

.1. Results. Although the results are discussed in detail during our presentation, we included one summarizing figure: figure 1, for refernce for when we talk about the different results below. Results were plotted using Keras' history function, numpy and matplotlib in Python3.

.2. Music styles. The idea of researching the influence of music styles on the ease with which neural nets are trained to predict songs in the same style originated from the popular conception that Bach's music is very mathematical and that Bach can be described perfectly with a couple of formulas (studied in e.g. (16)).

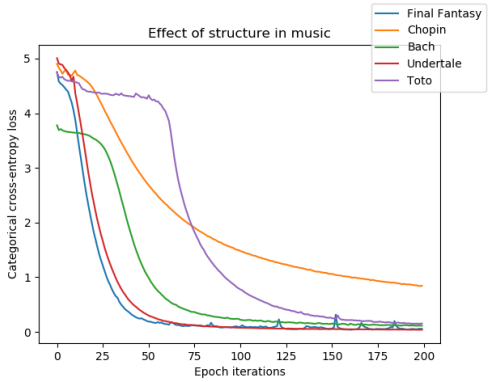

Fig. 1. The loss convergence of different datasets

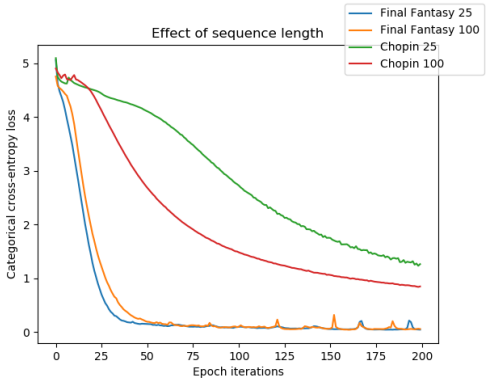

Fig. 2. The influence of sequence length on loss convergence

We wanted to extend this hypothesis by training on a database of only Bach midi files and another database of Chopin music, a composer from the Romantic era who has written much more complex pieces of music that is more difficult to explain mathematically. The hypothesis was that the training of a neural network would therefore be much harder on the database of Chopin music, then it would be on the Bach dataset, since the recognition of patterns in Chopin music is not an easy task. The results satisfy our hypothesis completely, both in the generated music as in the loss plot. The songs that are generated in the style of Bach, can fool a not so experienced person into thinking that it's Bach's music. On the contrary, the songs generated by the Chopin dataset are often incoherent and do not resemble any particular Chopin patterns. As can be observed in the loss plot, the dataset of Bach is much more easy to learn useful patterns from (earlier drop in loss) and can even learn better in general (lower loss at the end).

The same observation can be made for other databases. A field of music which is very structured is the typical 8-bit game soundtracks (represented in our experiments by a database of *Final Fantasy* and *Undertale* midi files. This style of music is very repetitive and has a lot of the same rhythmic patterns going on (mostly eighth notes). Again a human judgement can also give a good evaluation here. The pieces generated from Final Fantasy or Undertale sound a lot like the actual soundtrack. It can be seen that that the training of the Toto dataset did not go as well as the other ones, the explanation is the following: The classical piano composer github works by learning one-track midi files or selecting the first track from a multi track midi file and training on that one. All our datasets consisted of single instrument (one track) midi files, except for the Toto database. This would not make any difference where it not that the first track of the Toto database was not always the same instrument. This means that the model would train on a bassline for one song, a piano score for another or the guitar solo for yet another song. The model therefore needed a lot more time to generalize on the different patterns it saw. This unintended experiment shows the need of a balanced and curated dataset and the difficulty with learning patterns in very differing pieces.

.3. Sequence Length. The results of this experiment are summarized in figure 2.

The second experiment performed for the creative part is to research the influence of the sequence length parameter on the generated prediction of the trained model. The hypothesis for this experiment is that a lower sequence length means a lower coherence of the generated music. Sequence length is the amount of times a recurrent neural network will unroll, or the length of the input (the history) that can be remembered in the internal state. In this context, a sequence is just a pattern of sequential notes in the input music and will be cut off at the maximum sequence length. It is intuitive to think that a larger sequence length will result in a better predicted result, because the trained model will remember longer phrases of music and will hopefully be able to translate this in a consistent piece of music. To test this hypothesis, we trained a model on the *Final Fantasy* dataset for both a sequence length of 25 and a sequence length of 100 (the default of the original model). The results are difficult to measure with metrics, but can be evaluated by human judgement. Our conclusion is that the sequence length of 25 can still predict individually coherent bars, but fails to connect these bars into a meaningful way and sometimes makes random "musical jumps". The sequence length of 100 can remember a lot more patterns and learns a model that can incorporate a coherent musical history. The generated prediction is one piece of consistent music, with much less weird "jumps". The same experiment was ex-

ecuted on the more complex Chopin dataset. The results were more conclusive with this experiment. The training loss converges more quickly with a larger sequence length because it is easier for the network to memorize longer patterns. Once again the difference is mostly noticeable in the predicted end results.

.4. Musical embedding layers. A final experiment executed as a part of our creative section for this project was a research oriented question, that we would have liked to answer. The idea originated from the analogy of the neural network models used in this field of Artificial Intelligence compared to the ones used in Natural Language Processing (NLP), another very interesting research field. Both research areas use LSTM's in many of the models and work with the same kind of integer streamed input as used in the neural network that we use. To understand the setup of this experiment, a little bit of background in NLP is necessary.

Sentences are parsed by a so called tokenizer into a stream of integers were each unique token (or word) gets a unique integer. These integer streams can be inputted directly into a LSTM layer to be processed and hopefully get the wanted results, but most of the time the LSTM layer is preceded by an Embedding layer. Word embeddings are fixed-size real valued vectors that describe a word by looking at the context where the word occurs. These embeddings can be trained as part of the neural network, but they can also be loaded from a set of pretrained embedding weights that are trained on this task specifically. The most common networks that are used for this task today are *Skipgram*, *CBow* and *GloVe*.

The intuitive explanation of a word embedding is that it describes how similar two different words are by measuring the distance between their corresponding word embeddings. Similar words will have similar embeddings because they will most likely occur in the same contexts, even if they never even occur once in the same sentence. Therefore the representation of a word by its embedding makes much more sense than representing it with an integer.

The idea for this experiment was to make an analogy by introducing embeddings for music notes. The notes are parsed from midi to a unique integer and therefore get the same representation as the words. An embedding layer could ideally learn the context and the similarity of two different notes and learn which notes frequently co-occurred for example.

After modifying the neural model to include and train these embeddings we observed that the network completely stopped learning. The main reason for this we think is the random initialization of these embeddings, combined with a database that is far too small. Word embeddings are typically learned with huge databases, and such a database was not available for the training of musical embeddings. We still believe that this idea might work, but would need a far bigger dataset to actually achieve it.

Conclusion

As we are both musicians, this topic peaked our interest. Not knowing anything upfront, the journey was enlightening. We are glad that our intuitive hypothesis were all proven by our experiments in the creative part, although we are not claiming that we produced statistically significant results. As with many problems in Artificial Intelligence, a lot of the results are very depending on datasets. Which is proven by the fact that almost all available academic literature uses Bach datasets, a very easy learnable composer, as we have shown.

References

1. Google magenta. https://magenta.tensorflow.org/. Accessed: 22/02/2019.
2. Simon Colton, Geraint A Wiggins, et al. Computational creativity: The final frontier? In *Ecai*, volume 12, pages 21–26. Montpelier, 2012.
3. Moritz Hilscher and Novin Shahroudi. Music generation from midi datasets.
4. Douglas Eck and Juergen Schmidhuber. A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 103, 2002.
5. Olof Mogren. C-rnn-gan: Continuous recurrent neural networks with adversarial training. 2016.
6. Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. 2012.
7. Introductory slides on musical creativity. https://www.cs.helsinki.fi/webfm_send/1685. Accessed: 22/02/2019.
8. Nyquist homepage. https://www.cs.cmu.edu/~music/nyquist/. Accessed: 22/02/2019.
9. Mary Simoni and Roger B Dannenberg. *Algorithmic Composition: A Guide to Composing Music with Nyquist*. University of Michigan Press, 2013.
10. Music generation crnn - gan. https://github.com/dhruvsharma1992/Music-generation-cRNN-GAN, . Accessed: 22/02/2019.
11. Results from (5). http://mogren.one/publications/2016/c-rnn-gan/. Accessed: 22/02/2019.
12. Results from (6). http://www-etud.iro.umontreal.ca/~boulanni/icml2012. Accessed: 22/02/2019.
13. Classical piano composer: Github. https://github.com/Skuldur/Classical-Piano-Composer, . Accessed: 22/02/2019.
14. Classical piano composer: medium. https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5. Accessed: 22/02/2019.
15. Keras: The python deep learning library. https://keras.io/.
16. Rahul Siddharthan. Music, mathematics and bach. *Resonance*, 4(5):61–70, 1999.