RADBOUD UNIVERSITY

BACHELOR THESIS

Drum and Melody Generation using LSTM - based Neural Networks

Author:
Clemens Carl Christopher BEISSEL

Supervisor:
Umut Güçlü and Luca Ambrogioni

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Social Science

in the

Artificial Creativity Group
Department of Artificial Intelligence

February 12, 2019

Contents

Abstract

For this project, I constructed two LSTM - based neural networks that can generate monophonic melodies and polyphonic drum patterns. As opposed to projects which were conducted in the past, this attempt was focused on a combination of genres rather than training on only one instrument from one genre. When generating melodies, the patterns that resulted from this challenge were somewhat chaotic with some potentially inspirational exceptions. Generated drums, on the other hand, oftentimes converged to an "average of all genres" when no controlled randomness was introduced (section 7.1.5). A better imitation of the training data can certainly be achieved by using only one genre. But the involvement of several different genres led to a more unpredictable and creative outcome.

1 Introduction

The creation of music has been part of the human race since the beginning of mankind. It has always been an art that can only be performed by humans. But now, in the 21th century, technology has been evolved so far that computers are showing signs of intelligence and creativity.

A creative process is usually highly dependent on a sequence of contextual actions. These actions are, by themselves, not considered creative, but it is the interaction in which they are executed that forms a creative product. Movies, paintings and dance performances are examples of creative performances, that are constructed from a sequence of actions where every action depends on the history of previous actions. Music also belongs to this category where the sequential actions can be represented by notes, ranging from the lowest C_0 to the highest C_{11}.

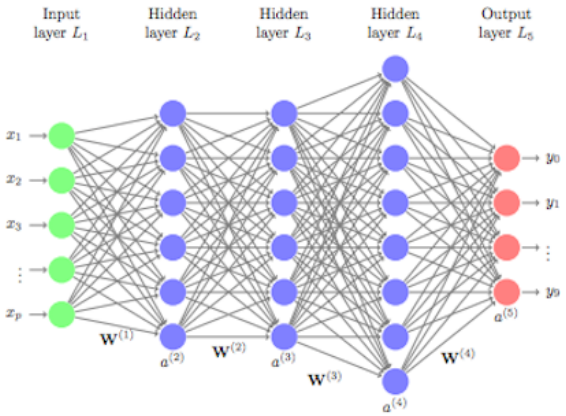

FIGURE 1: Deep neural network with 3 hidden layers (*Feed-forward Deep Learning Models*)

In order to concatenate notes in such a way that they form a coherent piece of music, a basic understanding of harmonics and note dynamics is necessary. Teaching a human about which note arrangements result in a pleasant piece of music, is rather trivial but there occur many difficulties and roadblocks when it comes to training a computational network.

1.1 Neural Networks

Machine learning works by feeding a network different input files from a specific category and then evaluating the network's output while adjusting its parameters. The input can hereby take the form of images or videos, but it can also be in the form of music. The most popular example of a neural network is the feed forward network (FFN) which consists of an input and output layer and multiple hidden layers in between where every layer consists of a varying number of perceptrons that are connected to all perceptrons of the next layer (*From perceptron to deep neural nets*).

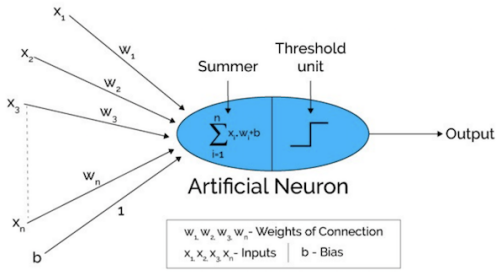

FIGURE 2: Single perceptron in a neural network (*From perceptron to deep neural nets*)

A perceptron is a single neuron / node in a network's layer that receives the accumulated result of every other perceptron of the previous layer multiplied by its connection strength (weight). Based on this information, it outputs either 0 or 1, depending on its threshold function.

A network has to be trained with values that match the dimensions of its input layer so music pieces have to be encoded into a particular datastructure. For this purpose, I have processed the information of MIDI files that contain information about starting times, ending times and pitch of all notes for every instrument in a music track, rather than rich audio signals. However, due to the length of a music track, which varies from piece to piece, I cannot use a simple FNN for generating music since the size of the input dimensions are not fixed. Therefore, I used a special type of network, the recurrent neural network (RNN).

1.1.1 Recurrent Neural Networks

Generally speaking, a RNN is a special type of neural network which computes its output based on the current input and its computational history. This makes it possible to feed the network single notes which are used to predict the next note. However, the output of RNNs can be corrupted by vanishing gradients that cause the network's information to be multiplied by very small numbers (TIME, 2015). This happens because normal RNNs have to feed their entire information through all cells to get to the cell that is currently processed. For this reason, I implemented a more complicated type of RNN strucutre, the long short-term memory (LSTM). The salient properties of this building block are discussed in section 7.1.2.

With this knowledge, I trained two RNNs with a large number of instruments from different genres. The networks should be able to generate monophonic melodic midi files and polyphonic drum patterns. Due to the numerous possible combinations of notes and the subjective quality of a piece of music, these two tasks are a challenging exercise for artificial intelligence. Especially, when the networks are fed with several genres.

This leads me to ask the following research question:
Can LSTM-based neural networks generate subjectively pleasant percussive and melodic patterns when fed with several different genres?
This question, creativity in computers, implementation, results and alternative approaches are discussed in this paper.

2 Literature Research

There has been a lot of research in the field of generative modelling of music with artificial neural networks. Most of the research is done by implementing LSTMs (long short-term memory), a special kind of RNN structure, which have been proven to be very powerful in combination with music generation and genre classification. Furthermore, the vast majority of research appears to be focused on one specific genre or

musical instrument, as opposed to my approach of combining several genres and instruments into two models.
I would say that "Google" is currently the leader, when it comes to artificial music production. Among other things, their latest project "Magenta" is capable of composing music that is pleasant to a lot of people. They have had a lot of success using RNNs as can be seen in one of their latest models "Performance RNN", which can generate classical polyphonic piano music based on live performances of real artists (Hutson, 2017). It incorporates dynamics and expressive timings in order to give the generated music a human touch. Also, they only trained their model on piano performances to keep the output in a confined range.
Another successful state-of-the-art model by Magenta is the "Music Transformer" (Huang et al., 2018). It is a sequence model that is based on self-attention in contrast to the LSTM "Performance RNN" model. Normal LSTMs cannot generate music with long-term coherence, which is quite important for whole pieces of music. They are capable of generating music that sounds familiar to a specific start sequence for a couple of moments but they deviate from the main theme relatively quickly. A transformer-based model, however, has direct access to all previous states, as opposed to a LSTM based model that compresses all previously encountered notes into a fixed-size hidden state (*music transformer magenta*). This makes it able to form long-term coherences for common song structures that contain reoccuring themes. However, when it comes to music, relative timing of notes is essential. Modulating self-attention based on pairwise distances turned out to be quite memory-expensive so that minute-long compositions were computationally intractable (Huang et al., 2018). Magenta tackled this problem by introducing relative positional information, which allows attention to be modulated by how far two notes are apart in a sequence (Huang et al., 2018). This made it possible to generate minute long sequences of coherent music due to the now linear memory dependency.
Another successful approach has been researched by Nicolas Boulanger-Lewandowski, Yoshua Bengio and Pascal Vincent. They modelled temporal dependencies using a combination of a restricted Boltzmann machine and a recurrent structure to be able to generate and predict polyphonic music (Boulanger-Lewandowski, Bengio, and Vincent, 2012). The accuracy of their models were slightly higher than the accuracy of a traditional LSTM. Thus they have shown that LSTMs might not be the best network structure when it comes to generating music.

My motivation for building a LSTM based neural network originates from the thought that drum patterns are highly repetitive and can therefore be modelled quite well by LSTMs since they do not rely on reoccuring themes of any kind. Melodies, on the other hand, are very "theme-dependent" and LSTMs are not able to make predictions based on a specific chunk of events (Huang et al., 2018). Despite this fact, I want to compare the performance of both models based on their accuracy with which they predict MIDI events and their ability of generating creative note patterns.

3 Creativity in Artificial Intelligence

3.1 Applications

Modelling human creativity is an important aspect in artificial intelligence, especially when looking at it from a psychological perspective. It can aid the process of understanding creativity in humans and it can help by specifying or altering the definition of creativity in general.

Also, creative programs could support the work in laboratories and the market place (Boden, 1998) by guiding the development of tools via a better understanding of human intelligence and creativity. When it comes to music, an artificially generated note pattern could help inspire the user in terms of producing subjectively better tracks.

3.2 Discussion and Limitations

Many people believe that creativity is a purely human phenomenon and that creative brilliance, in particular, can only be achieved by humans (Bringsjord and Ferrucci, 1999). But there are some people who think of creativity as a more rational process than first meets the eye.
Computers work in a very rational way: You can retrace a computer's output by analyzing the functions that processed the input. But if there are thousands of connections between input and output that are all functioning in a slightly different way, it is hard to determine if the output is indeed of rational nature.
It is important to note that real creativity is hard to define because it is difficult to evaluate. A network can predict the genre of a piece of music without someone questioning its creativity due to the non-ambiguous targets through which the network learns. But when it comes to generating something that matches the input space, evaluation tends to be hard just like in the real world.
There are people who are asking themselves whether humans are merely machines (Bringsjord and Ferrucci, 1999) since, just like our brains, a neural network consists of a vast amount of interconnected nodes. This is an interesting theory because of the fact that contemporary networks do not have nearly as many neurons as a real brain and would lack the computational resources for efficiently using them. The truth value of this theory can therefore better be determined in a future where the limits of a neural network are not bound to processing power.

4 General Idea

The goal of this project was, to create two artificial neural networks, that can generate drum patterns and melodic sequences. The general idea was taken from a language modelling approach, where a network is trained to predict the next word, based on the history of the previous words. This approach was simply transformed into predicting the next note (or percussive combination), based on the history of the previous events.
The models had to be able to generate a sequence of notes, when feeding the output back into the model. This is why a convolutional network, that Fourier transforms a synthesized audio file into a frequency spectrum, is out of question, despite its good classification accuracy for genres and instruments (Choi et al., 2017). Of course, a frequency representation of a sound is much more informative, than only information about the pitch, but especially when generating drums, the frequency stays quite stable over the set of percussive instruments, so pitch and timing of notes should be adequate for an accurate prediction.

5 The Dataset

I used the Lakh MIDI dataset (Raffel, 2016) for the purpose of building generative models. It contains over 170.000 MIDI files, but training on that many midi files would take too much time, therefore I used a subset of around 3.000 files that are matched to the entries of the "Million song database" (Bertin-Mahieux et al., 2011). The dataset consists of a great variety of genres, reaching from Classic to Deathmetal. When training the

models, I used MIDI files from numerous genres to achieve an evenly distributed set of possible outputs.

6 Preprocessing the Data

The processing differs slightly from model to model, but both pre-processed lists contain a large number of intervals where no note is played. This would bias the models in a direction where only successive sequences of zeros would be predicted. To prevent this, I experimented with deleting all sequences from the data that overshoot a specific amount of successive zeros.

6.1 Drum Generation

For the **drum generating network**, it is important that the input and output are polyphonic since there are a lot of times when two or more percussive instruments are played simultaneously. It is essential to capture those time points since they define the groove of a drum pattern.
Magenta provides powerful functions for this purpose: I used the `DrumExtractor` to extract drum tracks from a quantized note-sequence. The quantization was achieved by the `Quantizer.transform` function from the `note_sequence_pipelines`. I transformed the resulting drum tracks into a binary representation by using a default list of drum pitches consisting of 9 percussion types. Thus, every note from a drum track was matched to a percussive instrument in the drum list. This allowed me to squash the whole pitch range into a concise group of 9 drums, making it possible to keep the polyphonic aspect of the midi files intact. The binary representation is saved as an array with length 9, where every value can either be 1 or 0. This makes a total of 2^9 possible combinations, where every combination can be expressed as a binary number. For instance, the combination $[0, 0, 0, 0, 1, 0, 1, 1, 1]$ can be decoded into the integer 23. Since the `EmbedID` function takes an array of integers, every encoded binary time stamp has to be decoded into an integer and to be appended to a list.
This process was repeated for training, validation and test sets, where each set has its own share of midi files. This is important in order to validate the model with data that it has never seen before during training.

6.2 Melody Generation

Since melodies are very often polyphonic, it would be desirable to also represent them in a binary fashion. However, my approach made this impossible due to the enormous number of potential combinations (2^{128}). Luckily, a polyphonic representation is not mandatory since melodies can also be generated without chords. This is why the input and output of the **melody generating network** is monophonic.
The (mostly) polyphonic information of instruments has to be converted into monophonic information, which was done using the `MelodyExtractor` from Magenta, which extracts monophonic melodies from a quantized note-sequence while filtering percussive information. I squashed the pitch range of 128 notes into a range of 35 pitches to constrict the number of possible predictions while preserving the pitch relative to the octave.
The resulting one-hot matrix consists of only one active note at a time step and can therefore be converted into an array of indices where an index stands for the position of the 1 in a one-hot array. The resulting integers of every midi file are then simply concatenated after each other to form the data arrays.

Layer	In	Out	Nr
EmbedID	512	512	1
LSTM	512	256	2
LSTM	256	128	3
Linear	128	512	4

TABLE 1: Network structure of drum generation model

Layer	In	Out	Nr
EmbedID	36	36	1
LSTM	36	18	2
LSTM	18	9	3
Linear	9	36	4

TABLE 2: Network structure of melody generation model

7 Implementation of the Networks

In order to generate music, the networks had to make predictions based on their history of previously encountered notes. Therefore a recurrent neural network structure (RNN) has been used to encapsulate this idea. Given an input stream $x_0, x_1, x_2, ..., x_n$ and an initial state h_0, a RNN iteratively updates its state by $h_t = f(x_t, h_{t-1})$ (*Chainer Documentation*). For the network, this is essential in order to predict notes that stay in a particular context to each other. A traditional deep neural network would not suffice in this case, as it assumes a fixed dimensional input size, rather than an input stream of undefined length.

7.1 Network Structure

The networks consists of an `EmbedID` link, two `long short-term memory (LSTM)` layers and a `linear` output layer. Furthermore, Chainer's dropout function is called on the output of every layer except the last.

7.1.1 EmbedID

The `EmbedID` link converts the input pitch / binary decoded integer into an array that fits the number of possible notes / combinations. The possible notes for the melody generating model is 36, whereas the possible drum combinations for the drum generating model is 512, including the time step where no note is played. I used the EmbedID link (over a one-hot vector input) because integers are computationally much more efficient than one-hot vectors.

7.1.2 LSTM

The two `LSTM` instances are fully connected layers, which handle the sequential input. They are the core of the network where the internal state of the current computation is stored and the dimension of the input is compressed, like in a normal fully connected hidden layer. LSTMs enable the network to make long-term dependency-based predictions for the next combination of notes.
A LSTM link consists of multiple 'gates', which control the cell state of a layer. The first gate is essentially a sigmoid function, that takes the last output $h - 1$ and the current input x and puts out a number between 0 and 1, which is element-wise multiplied with the cell state. This tells the cell state which information to keep and which to "forget". (*Understanding LSTM networks*)
The next cluster of layers decides, what new information to add to the cell state. This is done by another sigmoid function, that determines which values to update, combined with a tanh layer, that creates an array of potentially new information. This filtered new information than is added to the cell state in order to update the state.
Finally, the output of the cell state can be computed by another gate mechanism, that filters what parts of the cell state should be put out at a particular time

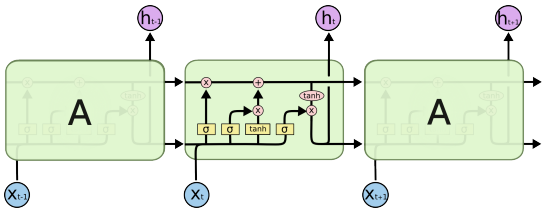

FIGURE 3: Structure of LSTM layer (*Understanding LSTM networks*)

step t. (*Understanding LSTM networks*)
A graphical representation of a LSTM is depicted in figure 3.
Chainer's `lstm` function implements this functionality efficiently via those computations:

$$c = tanh(a)\sigma(i) + c_{prev}(\sigma(f))$$
$$h = tanh(c)\sigma(o)$$

7.1.3 Linear

The linear layer acts as a normal fully connected output layer, where a predictive variable for the next note / combination is computed. Its output is computed by $f(x) = W * x + b$, where W and b are the weights and biases of the link, that are updated over time.

7.1.4 Dropout

This function drops elements of the input variable with a chance of 50% to prevent the network of getting overfitted on the training data. Overfitting occurs when a network excessively is trained on a specific dataset. So the model is biased towards predicting only values that make sense in context with the training data. That means the accuracy for other distinct datasets would be poor since the network's weights would have an increased strength towards the right values of the training data.

7.1.5 Temperature modulated Softmax

The datasets are filled with values that make out a dominating proportion of the whole set of notes. This can result in a model which is biased towards notes that have a high occurrence percentage. The result can be long sequences of one and the same note without any change at all. To prevent this, I introduced a temperature variable T that controls the entropy E of predicted note sequences. It does so by dividing the "raw" predicted values P_{pre} of the network by T in order to compress the difference between notes that have a high occurrence probability and those that probably won't occur naturally. Therefore, high values of T modulate predicted sequences to be rather chaotic and low values keep the entropy of sequences flat. T is increased if E drops under a certain threshold and vice versa.
E is computed after a certain number of time steps after which T gets adjusted in order to keep E in the optimal range. If V is a note sequence with length n, then E is computed like this:

$$E = \frac{1}{n}\sum_{i=0}^{n}\begin{cases}1, & \text{if } V_i \neq V_{i+1}\\ 0, & \text{otherwise}\end{cases}$$

After P_{pre} is divided by T it is passed to a softmax function that computes the normalized probability for every possible event $1 \dots d$ like this:

$$P_{post} = P_{pre}/T$$
$$P = \frac{exp(P_{post})}{\sum_d exp(P_{post_d})}$$

In order to generate a note / combination I sampled an index from the probabilistic distribution P that represents the corresponding event.

7.2 Training the Models

In both models, I had to implement a parallel sequential iterator that iterates over the data sets and returns mini batches

and an updater which performs backpropagation through time and updates the weights and biases of the layers. Both of those classes were copied from the Language Model from Chainer and modulated, to fit the needs of my model (*Chainer Documentation*).

7.2.1 Parallel Sequential Iterator

Built-in iterators from Chainer do not support aggregation into mini-batches from different locations (*Chainer Documentation*). This is, however, necessary in order to iterate over an extremely large sequence of notes.
The parallel sequential iterator creates mini-batches consisting of pairs of current and next notes at different positions in the data. Doing so, it creates index offsets that are equally spaced in the note sequence and it uses them to extract different chunks from the data dependent on the current iteration.

7.2.2 BPTT Updater

Since the data sequences consist of a large list of concatenated notes, it is not possible to perform backpropagation over the whole sequence. This would far exceed the memory capacities of the program. Therefore I had to think of another approach of updating the parameters of the network. This problem was solved by truncating the computation graph by unchaining the computation history of the loss variable. This way, the backpropagation can only be performed on a small range of time steps that has been controlled via a hyperparameter called "`bproblen`".
The updater works by accumulating the loss of a specific number of note pairs for the size of `bproblen` and then calling the backward function on the loss variable followed by unchaining the computation history.
Truncated backpropagation is heuristic and makes the gradients biased but it works well if the backpropagation length is long enough (*Chainer Documentation*).

8 Results

8.1 Generating Patterns

In order to generate note patterns, I fed a starting note to the networks. The predicted output note is then fed back into the networks to generate a sequence. This is repeated until the sequence reaches a desired length.

8.1.1 Drums

I decoded every predicted percussive combination back into a binary representation consisting of 9 bits. After creating the corresponding MIDI file, it was plucked into the FPC drum machine of the digital audio workstation FL Studio. The default percussive samples of the drum machine where mapped to the first 9 notes, starting from C_0, in exactly the same order in which the drum sequences were extracted in the preprocessing stage.
When generating percussive combinations by always picking the index with the highest value from the probability distribution the generated output after a while converges to drum patterns that sound like an average of all genres with which the model was trained. By that I refer to a consistent 1/16 hi hat, a 1/2 snare drum and two 1/4 kick drums at the start of each beat. This is modulated by an occasional 1/32 hi-hat hit.
At the beginning of the generated sequences, the hi-hat appears to be hitting in a quite consistent 1/8 or 1/16 pattern. The kick and snare drum mostly appear in an alternated fashion that can sound slightly random at some times. However, the drum model only generates beats that consist of a hi hat, a snare drum and a kick drum. This is probably caused by a dominating number of

appearances of those percussive instruments as can be seen in figure 6 where, for instance, the combination 64 stands for a single hi-hat hit. There certainly are tom fills and cymbal hits in the training data but they only occur very seldom. These are more likely to occur when sampling events from a probability distribution with the corresponding chance. When applying this technique, the output is a lot more unpredictable, especially when modulating the probabilities with a dynamic temperature as discussed in section 7.1.5.

8.1.2 Melodies

As opposed to drum patterns, melodies are a lot more dynamic with regard to possible note sequences due to the vast range of harmonics that fit to a particular note. Only picking the index with the highest predictive value therefore only produces sequences with very little change since the network becomes biased towards predicting long sequences of the same note. You can see that there are multiple notes to choose from when sampling from the probability distribution as shown in figure 4.

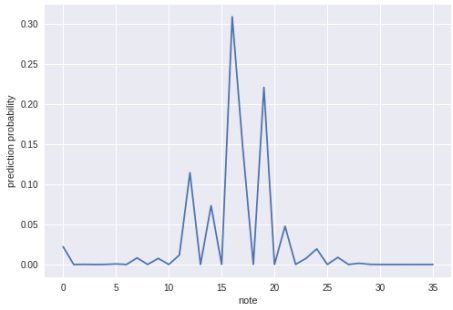

FIGURE 4: Probabilities of predicting note after softmax

Since every index in a predicted sequence corresponds to a single note, I could generate a MIDI file based on the raw samples from the softmax distribution. As for the output instrument, I used a regular sine wave with a short decay-low sustain envelope on the cutoff of a 12-dB lowpass filter accentuated with a hint of tube distortion and some reverb. I figured that the output instrument had to be of neutral electronic nature since the network was trained on so many different genres.

8.2 Analysis

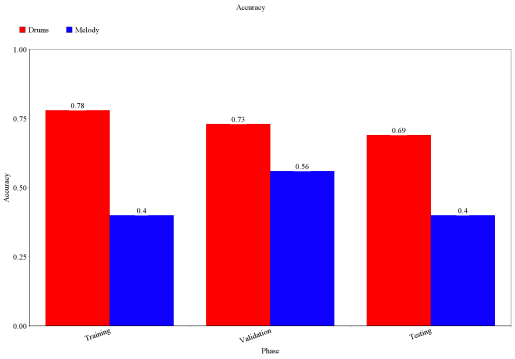

FIGURE 5: Accuracy of drum and melody models of all phases

8.2.1 Drums

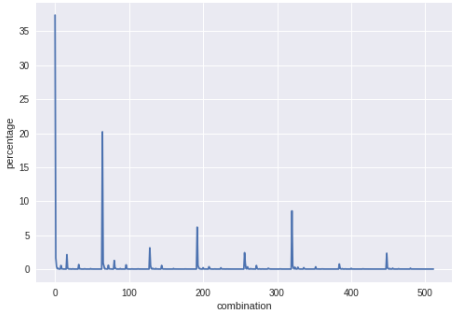

FIGURE 6: combination occurrences for all 512 possible drum events

The drum model's accuracy and loss appear to be stabilizing after only two epochs with 0.77% and 0.86 respectively. The high accuracy can be explained by the mostly repetitive structure of the

training and validation data. The occurrence probability of a percussive combination is often only modulated by the previous three combinations, which makes a correct prediction quite probable.
Predicting the next combination of percussive instruments by feeding the network some testing data works quite well with a stable accuracy of 69% which is only slightly worse than the accuracy of the training model. However, the network does not predict uncommon combinations very often due to the increased strength of weights that are connected to more common combinations. But still, modulating the entropy of sequences with a volatile temperature gives generated drum patterns a touch of controlled randomness.

8.2.2 Melody

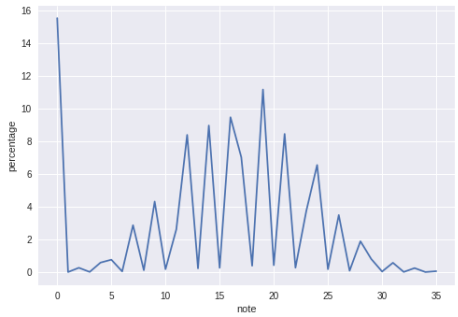

FIGURE 7: Note occurrences for all 36 possible melody events

The accuracy of the melody model is at around 40% for both, the training and testing phase. Considering the 36 possible combinations that are relatively uniformly distributed (figure 7), this is quite high. The accuracy of the validation phase lays at around 55%. This can be explained by the dropout functions that are only active in the training phase.

9 Conclusion

Looking back at this project, there are many things which could be done differently in order to generate patterns that sound more musically. But still, experimentally combining several different genres into two models gave me some insight into the creative capacities of LSTM-based neural networks. There are certainly limitations regarding genre combination since the resulting patterns exhibit some events that sound contextually unpleasant. Some sections, however, could be used for inspirational purposes to guide the production process of man-made music.
Therefore, I cannot answer the research question in a nonambiguous way. On the one hand, the networks produce passages that are definitely usable in a musical context. On the other hand, the output can sometimes be a concatenation of MIDI events that sound rather meaningless.

9.1 Discussion

The state-of-the-art networks are able to produce melodic patterns that are hard to differentiate from original performances. Only focusing on one instrument from one genre is probably the biggest reason for that. However, if I trained my network under this condition, one could argue that the resulting output is a mere imitation of the training data. By definition [1], this would defeat the purpose of programming a network capable of generating creative output since originality does not include mimicking. On the other hand, the combination of multiple genres can result in unusual ideas which are considered to be creative.
Expressive timings and dynamic velocity changes are from equal importance since they dissolve the electronic sterility

[1] https://dictionary.cambridge.org/de/worterbuch/englisch/creativity

of perfectly quantized note sequences. Implementing these features consumes a vast amount of computational resources because every note can be played at 127 different velocities, combined with several inter-positional time points at which a MIDI event can occur. This increases the training-time of the model tremendously but it is also a powerful tool for humanizing a generated piece of music.
Another successful approach involves the implementation of a transformer-based model with self-attention. Direct access to previous states is helpful especially when generating music with reoccurring themes as shown in the Magenta project "Music Transformer" conducted by Google (Huang et al., 2018). Furthermore, there is a good chance that the majority of LSTM and convolutional based networks will be replaced by attentional networks since the prediction accuracy is usually higher and the training time decreases dramatically (Vaswani et al., 2017).

9.2 Future Prospects

How will the listener's preferences change when considering the future development of artificially generated music? There is no denying that neural networks are getting better and faster with regard to music generation. As a result, the gap between artificially generated patterns and music created by humans is getting smaller and smaller. Only recently, a neural network was used to complete the two missing movements of Schubert's famous "Unfinished Symphony". According to a critical comment, the artificially added passages only seldom remind of Schubert and the patterns tend to move around an empty musical center [2]. This gives strength to the assumption that this network might be LSTM-based and that other computational structures might be a better choice for a fitting completion. But further work in this area might result in a finalization of Schubert's work - and that of other artists -, which would not be easy to differentiate from their original style. If this was the case, then there would only be little doubt about the existence of musical artificial creativity, and the bounds of humans and digital constructs would blur even further. This would also strengthen the assumption that our brains are mere machines and that our actions and thoughts are only projections of neural impulses which are controlled by environmental and biochemical processes.
Machines are still far away from exhibiting all mandatory properties to resemble human intelligence but when considering the current progress, I would say that, at some point, human-like artificial intelligence will be inevitable.

[2] Rheinische Post 6.2.19 - "Vollendung klingt anders" by Regine Müller

References

Bertin-Mahieux, Thierry et al. (2011). "The Million Song Dataset". In: *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.

Boden, Margaret A (1998). "Creativity and artificial intelligence". In: *Artificial Intelligence* 103.1-2, pp. 347–356.

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

Bringsjord, Selmer and David Ferrucci (1999). *Artificial intelligence and literary creativity: Inside the mind of brutus, a storytelling machine*. Psychology Press.

Chainer Documentation. http://docs.chainer.org/en/stable/examples/rnn.html. Accessed: 2018-12-11.

Choi, Keunwoo et al. (2017). "Convolutional recurrent neural networks for music classification". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 2392–2396.

Feedforward Deep Learning Models. http://uc-r.github.io/feedforward_DNN. Accessed: 2018-12-23.

From perceptron to deep neural nets. https://becominghuman.ai/from-perceptron-to-deep-neural-nets-504b8ff616e. Accessed: 2018-12-23.

Huang, Cheng-Zhi Anna et al. (2018). "An improved relative self-attention mechanism for transformer with application to music generation". In: *arXiv preprint arXiv:1809.04281*.

Hutson, Matthew (2017). "How Google is making music with artificial intelligence". In: DOI: http://www.sciencemag.org/news/2017/08/how-google-making-music-artificial-intelligence.

music transformer magenta. https://magenta.tensorflow.org/music-transformer. Accessed: 2018-12-30.

Raffel, Colin (2016). *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. Columbia University.

TIME, BACKPROPAGATION THROUGH (2015). "Recurrent Neural Networks Tutorial, Part 3–Backpropagation Through Time and Vanishing Gradients". In:

Understanding LSTM networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: 2018-12-19.

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in Neural Information Processing Systems*, pp. 5998–6008.