MahlerNet: Unbounded Orchestral Music with Neural Networks

Elias Lousseief and Bob L. T. Sturm
School of Electrical Engineering and Computer Science, Royal Institute of Technology (KTH), Stockholm, Sweden
`elias.lousseief@blackwinged-angel.com, bobs@kth.se`

ABSTRACT

This paper presents MahlerNet, a deep recurrent neural network that models polyphonic music sequences of arbitrary length with an arbitrary number of instruments. The data representation consists of instrument, pitch, offset and duration, which is motivated by properties inherent in both notated and performed music. It generates units of music (i.e., measures in this work) by sequentially sampling from distributions conditioned on context. This paper details experiments using two established datasets (PIANOMIDI and MUSEDATA), and a new dataset (MAHLER) consisting of all symphonies by Gustav Mahler. The smoothness of the learned latent space is explored by interpolating between two given measures of music. Results show that MahlerNet can generate music resembling its training data in many respects. Long-term structure is present in the form of instrumentation, intensity and rhythm, albeit rarely in the form of longer concrete motives and themes.

1. INTRODUCTION

The involvement of artificial intelligence in Arts practices is becoming a common activity, and music creation is no exception. Notably, AI has recently been used in the creation of popular music [1] and even folk music albums [1], not to mention start-up companies focused on the automatic generation of music for games [2] and soundtracks. [3] Ultimately, results in the field can serve many uses. AI composition tools can be used by composers to develop existing material, give inspiration to fragments of ideas, and to explore new kinds of music and ways of working.

While much research has been published about the modelling and generation of music, most of it is focused on melodies, chord progressions, or single instruments. Important aspects of music, however, are polyphony (multiple simultaneous, autonomous voices or instruments), instrumentation, and the development of musical ideas using multiple voices.

This paper presents *MahlerNet* [2], a polyphonic music model that aims to generate orchestral music with any number of instruments. The music representation of MahlerNet involves parameterizing each event in terms of a pitch, offset, duration, and instrument. MahlerNet models sequences of these events using a sequence-to-sequence network with context conditioning. These models generate new material by sampling sequentially from updated posterior distributions of the four parameters.

The rest of this paper has the following structure. Section 2 briefly reviews polyphonic music modelling and generation. Section 3 presents the representation and architecture of the proposed model. Section 4 describes several experiments with trained models. Section 5 reviews the strengths and weaknesses of the results and 6 describes the contribution of MahlerNet and presents avenues of future work.

2. PREVIOUS WORK

Computational modeling and generation of music has a long history, stretching back decades to experiments in the 1950s [3]. While these first approaches involved expert-based systems, with or without probabilistic components, later approaches take advantage of data for training music artificial intelligence (music AI) [4, 5]. Much research in the domain of music AI – a subdomain of *algorithmic composition* [6] – is concerned with individual musical voices, e.g., melodies or chord progressions. Polyphonic music modeling and generation in contrast can be more difficult, since it is concerned with multiple simultaneous, autonomous voices, perhaps on the same instrument (e.g., piano), or as an ensemble (e.g., choir, chamber ensemble, or orchestra). Though much recent work addresses polyphonic music modelling and generation, many are not concerned with instrumentation at all [7–18]. A few works use four-part choir [19, 20] or ensembles of instruments found in popular music, e.g., bass, guitar, piano and drums [21–24].

Many polyphonic approaches propose different music representations and modelling methods. A common format is “piano roll”, which is a matrix of time-ordered column vectors with rows denoting which pitches are on at a given time [7, 9–15, 18, 20–24]. One column represents activity during a fixed amount of time (typically in note length) and is used by a system to generate the activity to follow. This approach is called *time slicing*. This implies that simultaneous events are fed in at one go and that the duration and starting point of each note is implicit, which makes the representation efficient. Problems involve distinguishing a sustained long note from successive shorter ones and in the case of multiple instruments, some extra device has to take care of which instrument plays what.

[1] `https://www.helloworldalbum.net`
[2] `https://melodrive.com`
[3] `https://www.aiva.ai`

An alternative representation, similar to how MIDI works, outputs one event at a time, even if the events are simultaneous [16, 17, 25, 26]. Such a representation needs to express starting time and duration explicitly, even though sometimes, duration is handled by treating the start and end of a note as separate events [16, 25].

These sequences of note events have often been modelled by recurrent approaches, such as recurrent neural networks (RNNs) [7, 9, 13, 14, 16–18]. Restricted Boltzmann machines have also been used, both as an output function of RNNs [7, 9] as well as on their own in deep belief networks (DBNs) [10] or with convolutional neural networks (CNNs) [11]. Other attempts use CNNs [20], generative adversarial networks (GANs) [22], and sigmoid belief networks (SBNs) [8]. The variational autoencoder (VAE) has been used recently [12,15,23–25], sometimes in sequence-to-sequence networks [23–25]. These are promising approaches because they learn a structured latent space with several attractive properties, for example the ability to interpolate between musical material and to perform arithmetic, such as imposing some attribute to the output by adding a so-called *attribute vector* to the latent code [24, 25]. Further details about the VAE can be found in [27] and in [28]. Among the state of the art in music generation is MuseNet [26], which is a *transformer* model (an attention-based feedforward neural network). MuseNet seems able to generate impressive, polyphonic compositions in different styles with up to 10 different instruments.

Where instruments are handled, their number is usually restricted to at most four [19–24], with the exception of MusicVAE which handles 8 in one publication [25], and MuseNet which handles 10 [26]. Furthermore, the modelling of instruments is usually hard-coded in the architecture [19–25] and not part of the data representation only, e.g., when an instrument does not play, the network outputs a token that implies silence in that voice. This poses a restriction on the number of instruments that can be used without making changes to the actual model.

3. MAHLERNET

The overall architecture of MahlerNet is inspired by MusicVAE [23–25], and its data representation and conditioning in output layers is inspired by BachProp [17]. MahlerNet is written in python using the deep-learning library Tensorflow,[4] and the Mido[5] library for MIDI input and output. Samples of material generated by MahlerNet can be heard online at `http://www.mahlernet.se`. Attendant code and instructions can be found at the project github repository.[6]

3.1 Data representation

MahlerNet uses a data representation where each played note (an "event") is expressed by four parameters. The *offset* parameter is the duration between the last event and the current one. The *duration* parameter is the duration of the current event. Both of these parameters index into a set of 60 possible values – from the 32nd note to a 32nd note less than five tied quarter notes – plus zero. The zero duration is used in the offset parameter for simultaneous notes, and in the duration parameter to signal an advancement of time only in a non-note event (more on the use of this in Sec. 3.2). The longest duration was chosen from observing how notes are often tied over bar lines in music. The *pitch* parameter is the pitch number of the current event, which indexes a set of 96 MIDI pitches from 17 to 112 (both inclusive). Finally, the *instrument* parameter identifies the instrument playing the current pitch, indexing into a list. The number of instruments modelled by MahlerNet is arbitrary but the current preprocessor instrument plugin class makes use of 23 instruments. The number of instruments is thus not hard-coded into the architecture itself and can be altered by simply altering the plugin class. The representation used by BachProp [17] includes the same parameters except for instrument.

This data representation can be referred to as "sequential polyphony", since simultaneous pitches are modelled sequentially but with offset parameters set to zero. This is in contrast to piano roll format where simultaneous pitches are represented in parallel. The representation used by MahlerNet alleviates problems that come with some other approaches, such as time slicing (e.g., rearticulation), and offers a clear conditioning order among the output notes, from early to later in time and from lower to higher in pitch. The latter order of conditioning is reasonable from a musical perspective where lower notes affect what is played in higher registers rather than the opposite.

Apart from the described data, each sequential step is also fed additional conditioning in the form of active pitches (currently turned on notes) and active instruments (the set of the instruments of the currently turned on notes). With one input representing the start of a piece of music (the context when generating the opening of a piece) and conditioning on position in the underlying metric pulse of eighth notes, the data representation of a piece of music is a sequence of 367-dimensional many-hot binary vectors.

3.2 Preprocessing

For several reasons, MIDI is often not a straightforward conversion from symbolic (written) music to sounding music, which results in both the starting point and the length of each detected event being subject to normalization. The notion of MIDI normalization is used in [17] and implies the definition and use of a set of rules or heuristics with which we interpret the original MIDI event. The normalized MIDI event is the same event but with starting time and duration adjusted with respect to some goal, in the case of MahlerNet what written note it most likely originated from. For example, even note-writing software will make use of existing articulations (e.g., *staccato*, *tenuto*) in the score when writing a MIDI file so that the mapping between the sheet music and the MIDI becomes ambiguous (an eighth note played tenuto might be easy to mistake for a quarter note played staccato). On top of that, many MIDI files on the Internet today are manually recorded from per-

[4] `https://www.tensorflow.org/`
[5] `https://mido.readthedocs.io/en/latest/`
[6] `https://github.com/fast-reflexes/MahlerNet`

formance with MIDI instruments which introduces imprecision in many ways. In orchestral MIDI files, to make the MIDI orchestra sound more like a real orchestra, creators often manipulate the music in many ways, sometimes adding things that are not in the score, or altering things, to make the synthesis sound better.

To deal with these problems, a study of the properties of notated and performed music have been used to create a series of heuristic scoring functions that ultimately decide how to normalize an event in a MIDI file [2]. The goal of this is to pick the most reasonable candidate given a series of candidates. Other heuristics govern which candidates to choose from for a given event. The importance of normalizing MIDI files is further discussed in [17].

To train MahlerNet, a dataset of MIDI files is preprocessed into the data representation described in Sec. 3.1. This entails mapping all MIDI instruments to instrument classes pre-defined in the preprocessor instrument plugin, and then ensuring that all pitches for a given instrument are within the prescribed range ($[17, 112]$). If some pitches are outside this range, they are mapped into the range by octave transposition(s). During the normalization of the offset and duration properties of each input MIDI note event, notes are considered in their context, and not in isolation.

In the preprocessor, each input MIDI file is divided into segments of events where one segment starts with the detection of a time signature MIDI event (or the beginning of the piece) and then contains all the events up until the next time signature (or end of the piece). Fifteen different time signatures are accounted for and segments beginning with other time signatures are ignored. This can cause an interruption in the sequence of consecutive segments, resulting in a single input piece being divided into several different uninterrupted sequences of segments.

Dataset preprocessing creates a collection of data representation files, where each file corresponds to an uninterrupted sequence of segments. During training, the preprocessor reads these files and divides them, on-the-fly, to batches of sequences of either a fixed or variable total duration (hyperparameter). The preprocessor version presented here uses the variable duration of a measure of music (this unit length is variable both because any number of events can actually exist in a measure but primarily because the actual length, in duration, varies depending on time signature). For the rest of this paper, a unit of music is considered a measure.

Input note durations and offsets with normalized durations longer than the longest duration are divided into several smaller, consecutive events in the data representation. First, the full measures that the duration covers are turned into different events. Next, the remaining parts of the original duration, the beginning and end of it, are normalized like all other durations with the exception that the one in the beginning has a fixed end at the end of the measure and vice versa with the remaining piece at the end. When this duration is an offset parameter, this process gives rise to some events that are non-note events that only advance time, and nothing else.

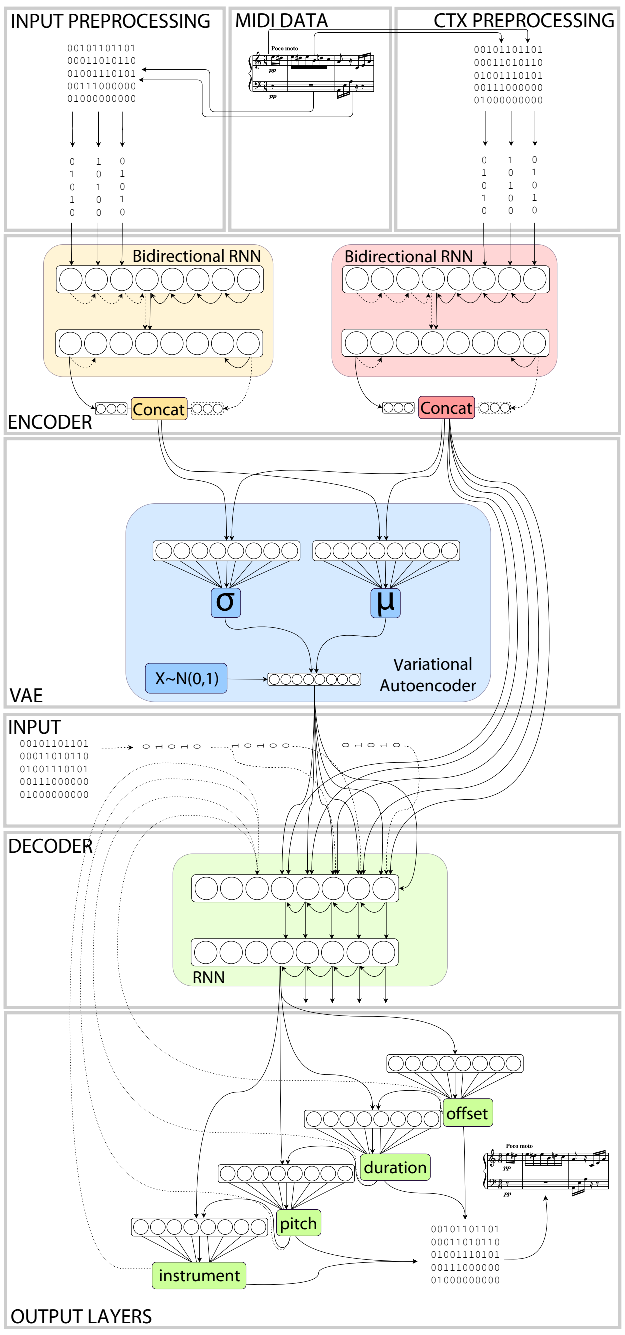

Figure 1. Schematic of the MahlerNet architecture.

3.3 Architecture

MahlerNet is a sequence-to-sequence network using a VAE to model the distribution of the latent state (like MusicVAE [23–25]). As can be seen in the schematic over the MahlerNet architecture in Fig. 1, the encoder side consists of two bidirectional RNNs where the first considers the measure to reconstruct and the second considers a context measure, e.g., the measure before the one to reconstruct. The final RNN states of both these bidirectional RNNs are concatenated and used as an input to the VAE, whose outputs are the parameters of a Gaussian distribution from which to draw a sample $\mathbf{z}$. The decoder then uses the sample $\mathbf{z}$ as a starting state, and the final states $\mathbf{c}$ of the context bidirectional RNN in the encoder as additional conditioning. This effectively results in the VAE of MahlerNet being a Conditional VAE (C-VAE) [29].

Each time the decoder RNN is advanced, it outputs first the offset parameter, then duration, then pitch and finally instrument. Sampling these parameters is done using distributions created from softmax output layers conditioned on the outputs from the previous output layers, according to the parameter order described above (which is also used in [17]). Denote an observation at state t as $\mathbf{x}_t = (o_t, d_t, p_t, n_t, ap_t, an_t, b_t)$ where o_t, d_t, p_t, n_t, ap_t, an_t, b_t denote offset, duration, pitch, instrument, active pitches, active instruments and metric position, respectively. The softmax layers output the conditional probability distributions of the parameters at the next state according to:

$$P(o_{t+1} \mid \mathbf{z}, \mathbf{c}, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t) \tag{1}$$
$$P(d_{t+1} \mid \mathbf{z}, \mathbf{c}, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, o_{t+1}) \tag{2}$$
$$P(p_{t+1} \mid \mathbf{z}, \mathbf{c}, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, o_{t+1}, d_{t+1}) \tag{3}$$
$$P(n_{t+1} \mid \mathbf{z}, \mathbf{c}, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, o_{t+1}, d_{t+1}, p_{t+1}) \tag{4}$$

Active pitches and instruments, as well as metric position in the measure, are computed beforehand for training data, and are calculated on-the-fly from the sampled parameters during generation. MahlerNet is highly configurable and comes with optional batch normalization and dropout with the possibility to use both the *β-VAE* [30] and *free bits* [31] innovations when calculating the Kullback-Leibler (KL) term of the VAE loss function. These extensions effectively guides the tradeoff between a smooth and well-behaved latent state and high-quality reconstructions.

4. EXPERIMENTS

Three datasets were used for training and testing MahlerNet models: PIANOMIDI,[7] MUSEDATA[8] and a new dataset consisting of most movements of all ten symphonies by Gustav Mahler (MAHLER).[9]. Table 1 gives some statistics of these three datasets. The music in the datasets are used as is with no transposition to a common key. A hyperparameter grid-search was conducted for each dataset whereupon activation function, learning rate and RNN cell type were established.

Trained models were evaluated in two ways: interpolations between given measures from the training datasets, and random or seeded sampling. After sampling from the VAE, the MahlerNet decoder is allowed to decode for as many steps as needed to output events with a total duration less than or equal to a full measure, as dictated by the desired time signature. Each generated measure can be used as context for the next. Teacher forcing (target output is fed as input to update an RNN instead of its actual output) was used in training the decoder.

Training was done on a GeForce GTX1080Ti GPU and took at most two hours for any individual setup and dataset.

[7] `http://www.piano-midi.de/`

[8] `https://musedata.org/` fetched from `http://www-etud.iro.umontreal.ca/~boulanni/MuseData.zip`

[9] MIDI files for MAHLER were gathered from two online sources: `http://gustavmahler.com/midi.html` and `http://midi-orchestra.xii.jp/ta/mahler.htm`

Dataset	PIANOMIDI	MUSEDATA	MAHLER
Measures	47544	37006	17450
Max length	116	158	303
Avg. length	14.16	26.25	29.01

Table 1. Statistics about the three datasets used in this work. The length of a measure is the number of events it contains.

4.1 Model training setup

MahlerNet models were trained separately with and without contextual input, after the initial hyperparameter grid search. Training was done with LSTM (long short-term memory) units in RNN layers, RMSProp optimizer and learning rate set to 0.001. Leaky ReLU (rectified lineary unit) was used as activation function with the exception of the VAE layers which used SoftPlus activation in the log-variance layer and no activation in the layer outputting the distribution mean. A batch size of 128 was maintained for all experiments except for the model trained on the MAHLER dataset with context conditioning; this model used a batch size of 96 due to size. The VAE had a single layer with 256 nodes whereas both encoder and decoder RNNs had two layers with 512 nodes in each. All models were trained with batch normalization in all dense layers, except for in the output layers and in the layers belonging to the VAE. The batch normalization takes place before the activation function but after the multiplication (as opposed to after the activation function as sometimes advocated).

We trained models without contextual input conditioning without dropout in a way similar to MusicVAE [23, 24], annealing the VAE loss function at rate 0.00001 with a β parameter of 0.2, calculated with 48 free bits. These settings favor reconstruction accuracy of training data over the quality of the latent space. By "quality" we mean that the decoder has been exposed to and has had the opportunity to be trained on all (most) points of the latent space, and that the transition between nearby points in the latent space results in gradual and small changes in the output space.

These models are meant to overfit, but to reconstruct input training samples accurately and interpolate between them in the latent space in a meaningful way. Interpolations are done by taking two endpoint measures from the training data and run them through the trained encoder and the VAE to generate a distribution for each. After sampling from both, ten steps of interpolation move from one sampled latent vector to the other. Because of the overfitting objective, no validation set was used and all models were trained for a fixed number of epochs (50).

We trained models using context conditioning with a dropout rate of 0.35 (probability of dropped connection) after the activation and with the VAE loss disregarding beta annealing or free bits. A validation set of 10% of the training data was used to determine when to abort training. These models are meant to create coherent sequences of measures sampling one measure at a time using the last measure as context. They are initialized either with a random vector with the same size as $\mathbf{z}$ sampled from a standard Gaussian

	Teacher forcing	
Dataset	*with*	*without*
PIANOMIDI	98.29%	80.94%
MUSEDATA	96.11%	45.87%
MAHLER	94.43%	27.20%

Table 2. Comparison of reconstruction accuracy of pitch with and without teacher forcing for models without contextual conditioning.

distribution (random sampling) or with a $\mathbf{z}$ vector sampled from the VAE originating from some chosen input and context vectors (seeded sampling). The context vector is either a "start" measure (only containing one step with only the "start" class set) or with some other measure, e.g., the one preceding the one to compose, for seeded sampling.

4.2 Results

Table 2 summarizes the results of experiments with the three datasets. Models trained without context conditioning were all trained for 50 epochs at which point almost no further improvement was observed. When attempting to reconstruct the training data without teacher forcing, a heavy drop in performance is visible. This is proportional to the average and maximum length of the sequences, as shown in table 1, in the different datasets.

Interpolations with these models seem successful for short sequences ($<$ 30 events), and endpoints are properly reconstructed with a plausible transition between them. An example of this can be seen in figure 2. With longer sequences, the transitions are less plausible and musical deviations from both endpoints take place. Sometimes, not even the endpoint measures are properly reconstructed even though often, the first notes are correct.

We stopped the training of models with context conditioning once the performance on validation sets started to drop. For PIANOMIDI, MUSEDATA and MAHLER, this resulted in the models training for 15, 21 and 31 epochs. With these models, exact reconstruction is worse than with previous models but quite to the contrary, random sampling works better and the decoders produce plausible music output.

We perform interpolations in 10 steps. Generated samples have 10 or 100 bars. We sampled all material at softmax temperatures around 0.9 for models trained on PIANOMIDI and MUSEDATA, and at around 0.6 for the MAHLER model.

5. DISCUSSION

The strongest argument in favour of using MIDI for training data is the availability where thousands of MIDI files can be found online. However, in principle, it is a representation of performed music rather than written. As a result, a very large responsibility is placed on the preprocessor when using this format to model written music.

The reconstruction capability of models trained without context conditioning drastically deteriorates as the number of events increases. In accord with what is said with respect to MusicVAE in [23–25] and brought up elsewhere [32], what is believed to happen here is a phenomenon named "mode collapse": the decoder learns to neglect the latent code and only base its output on the teacher-forced input. Evidently, when not even the endpoints can be properly reconstructed, it affects interpolations in a negative way. Nonetheless, often the measures resulting from these interpolations are plausible as music, however not constituting a smooth and continuous transition between the endpoints as desired.

Random sampled measures (generate a random $\mathbf{z}$ vector and decode it) from these models however do not pass off as music and neither of the output parameters, inspected in sequence, are realistic. This is quite in contrast to the result with seeded samples and it turns out that the lack of regularization of the latent space is the cause for this; the use of a β weight and free bits results in better reconstructions but less smooth latent space. Here it is obvious that it is fairly simple to sample a latent vector that the decoder has never experienced and it is thus uncertain how it will be decoded. However, mode collapse aside, interpolations are still well-behaved and since the MusicVAE is used with interpolations, the slackened regularization of the latent space appears reasonable in that case. For MahlerNet when used with context conditioning however, random sampling is an important factor and so these modifications of the VAE loss must be left out.

At the cost of exact reconstruction, the removal of these components effectively improves the outcome of sampling procedures in the models trained with context conditioning. With consecutive generation of multiple measures, no matter if the decoder is fed a random latent vector or seeded with one coming from a given input and context, the music is coherent in terms of dynamic tension, active instruments, intensity, tonality and rhythm.

We can find clear differences between outputs depending on training data. The model trained on MUSEDATA produces music that sounds Classic and Baroque whereas the model trained on MAHLER generates samples that sound more dissonant and characteristic of music of the Romantic period – both in line with the music found in the datasets. The output from the model trained with PIANOMIDI has almost always piano only whereas instruments such as tuba and trombone, which are quite common in the Romantic era but not in the Classic and Baroque eras, are commonly seen in output from the model trained on MAHLER but not in the other models.

Furthermore, instruments tend to stay within their real ranges and groups of instruments that typically play together during the different eras are active together in generated material as well. Examples of this is the chamber setting with strings, woodwinds and continuo in Baroque music and woodwind sections present in Classic music where strings otherwise dominate. In the 100-bar samples, contrasting parts are present both in terms of instrumentation and homophony versus polyphony. Sections where all instruments play often have brass, horns and timpani added much like in a real scenario with beats 1 and 3 typically

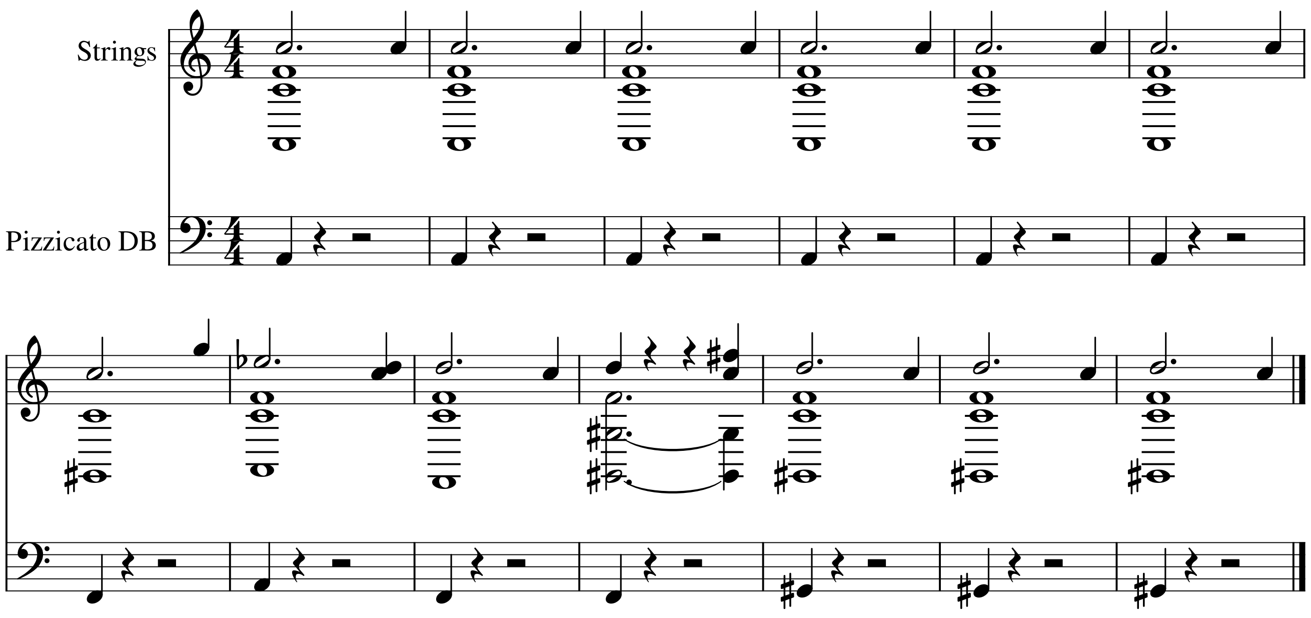

Figure 2. Interpolation (sample 3-1 at `http://www.mahlernet.se`) between two measures from the 4th movement of Mahler's fifth symphony, in a model trained without context conditioning.

emphasized.

Rhythmic consistency is also present and when MahlerNet is seeded with a measure containing particular rhythms (for example triplets) these often prevail for several measures. Finally, phrase endings with V-I motion are common, however not as common as in real Classic music.

Despite many good qualities, MahlerNet does only very occasionally produce music with motivic or thematic content that persist over many bars. This is a common defect in music AI based on neural networks. Nevertheless, in the case of MahlerNet, this tendency is almost expected since the temporal receptive field only spans one measure of history; expanding this context is necessary to improve the conditions.

6. CONCLUSIONS

MahlerNet is a new neural network based on MusicVAE and BachProp with a new data representation and preprocessor based on BachProp. MahlerNet is a first attempt at modelling orchestral music using deep neural networks with an architecture that allows for an unlimited number of instruments. Even with the limit of the current preprocessor, there are, to the authors' knowledge, many more instruments available than in any earlier publication. Music generated by MahlerNet shows coherence and long-term structure in many aspects, not the least with respect to instrumentation. Furthermore musical style is clear and correlates with training data and the music is both plausible and shows qualitative pregnancy. Reoccurring themes and motives, important aspects of music, are however rare findings in samples, perhaps due to the almost naively short context of a measure that is fed as conditioning.

The next step in developing MahlerNet is to increase its temporal receptive field by experimenting with longer contexts. The mode collapse problems in the decoder should be addressed, as well as steering the generation process with more conditioning. Further research on the impact of batch normalization on the latent space is also desirable since there are some indications that the structure of the latent space suffers from the use of it. On a more general level, an interesting direction of the future of music AI is hierarchical and transformer-based architectures.

Acknowledgments

The authors would like to thank the creators of the Mahler symphonies in MIDI format found online. [10] EL wrote most of this paper, and wrote all computer code. BLTS supervised of EL's master's thesis research, and initiated and edited this paper.

7. REFERENCES

[1] B. L. Sturm and O. Ben-Tal, "Let's Have Another Gan Ainm: An experimental album of Irish traditional music and computer-generated tunes," KTH Royal Institute of Technology, Tech. Rep., 2018.

[2] E. Lousseief, "MahlerNet - Unbounded Orchestral Music with Neural Networks," Master's thesis, Royal Institute of Technology, Stockholm, Sweden, 6 2019.

[3] L. Hiller and L. Isaacson, *Experimental Music: Composition with an Electronic Computer.* New York, USA: McGraw-Hill Book Company, 1959.

[4] J. D. Fernández and F. Vico, "AI methods in algorithmic composition: A comprehensive survey," *J. Artificial Intell. Res.*, vol. 48, no. 1, pp. 513–582, Oct. 2013.

[5] J.-P. Briot, G. Hadjeres, and F. Pachet, *Deep learning techniques for music generation.* Springer, 2019.

[10] `http://gustavmahler.com/midi.html` and `http://midi-orchestra.xii.jp/ta/mahler.htm`

[6] R. Dean and A. McLean, Eds., *The Oxford Handbook of Algorithmic Music*. Oxford, UK: Oxford University Press, 2018.

[7] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription," *Proc. Int. Conf. Machine Learning*, 2012.

[8] Z. Gan, C. Li, R. Henao, D. Carlson, and L. Carin, "Deep Temporal Sigmoid Belief Networks for Sequence Modeling," in *Proc. Int. Conf. Neural Information Process. Systems*, 2015, pp. 2467–2475.

[9] Q. Lyu, Z. Wu, J. Zhu, and H. Meng, "Modelling High-dimensional Sequences with LSTM-RTRBM: Application to Polyphonic Music Generation," in *Proc. Int. Conf. Artificial Intell.*, 2015.

[10] F. Sun, "DeepHear - Composing and harmonizing music with neural networks," https://fephsun.github.io/2015/09/01/neural-music.html, 2015, accessed: 2019-10-29.

[11] S. Lattner, M. Grachten, and G. Widmer, "Imposing higher-level Structure in Polyphonic Music Generation using Convolutional Restricted Boltzmann Machines and Constraints," *CoRR*, vol. abs/1612.04742, 2016.

[12] J. A. Hennig, A. Umakantha, and R. C. Williamson, "A Classifying Variational Autoencoder with Application to Polyphonic Music Generation," *CoRR*, vol. abs/1711.07050, 2017.

[13] D. D. Johnson, "Generating Polyphonic Music Using Tied Parallel Networks," in *EvoMusArt2017*, 2017.

[14] F. Liang, M. Gotham, M. Johnson, and J. Shotton, "Automatic Stylistic Composition of Bach Chorales with Deep LSTM," in *Proc. Int. Symp. Music Info. Retrieval*, 2017.

[15] R. Sabathé, E. Coutinho, and B. Schuller, "Deep recurrent music writer: Memory-enhanced variational autoencoder-based musical score composition and an objective measure," in *Proc. Int. Joint Conf. Neural Networks*, May 2017, pp. 3467–3474.

[16] I. Simon and S. Oore, "Performance RNN: Generating Music with Expressive Timing and Dynamics," https://magenta.tensorflow.org/performance-rnn, 2017, accessed: 2019-10-29.

[17] F. Colombo and W. Gerstner, "BachProp: Learning to Compose Music in Multiple Styles," *CoRR*, vol. abs/1802.05162, 2018.

[18] H. H. Mao, T. Shin, and G. W. Cottrell, "DeepJ: Style-Specific Music Generation," *CoRR*, vol. abs/1801.00887, 2018.

[19] G. Hadjeres and F. Pachet, "DeepBach: a Steerable Model for Bach chorales generation," *CoRR*, vol. abs/1612.01010, 2017.

[20] C.-Z. Huang, T. Cooijmans, A. Roberts, A. Courville, and D. Eck, "Counterpoint by Convolution," in *Proc. Int. Symp. Music Infor. Retrieval*, 2017.

[21] H. C. Chu, R. Urtasun, and S. Fidler, "Song From PI: A Musically Plausible Network for Pop Music Generation," *CoRR*, vol. abs/1611.03477, 2016.

[22] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment," *AAAI 2018*, 2017.

[23] A. Roberts, J. Engel, and D. Eck, "Hierarchical Variational Autoencoders for Music," in *Workshop on Machine Learning for Creativity and Design, NIPS*, 2017.

[24] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music," *CoRR*, vol. abs/1803.05428, 2018.

[25] I. Simon, A. Roberts, C. Raffel, J. Engel, C. Hawthorne, and D. Eck, "Learning a Latent Space of Multitrack Measures," *CoRR*, vol. abs/1806.00195, 2018.

[26] C. Payne, "MuseNet," https://openai.com/blog/musenet/, April 2019, accessed: 2019-09-19.

[27] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *CoRR*, vol. abs/1312.6114, 2013.

[28] D. Jimenez Rezende, S. Mohamed, and D. Wierstra, "Stochastic Backpropagation and Approximate Inference in Deep Generative Models," *CoRR*, vol. abs/1401.4082, 2014.

[29] K. Sohn, H. Lee, and X. Yan, "Learning Structured Output Representation using Deep Conditional Generative Models," in *Proc. Advances in Neural Info. Process. Systems*, 2015, pp. 3483–3491.

[30] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework," in *Proc. Int. Conf. Learning Representations*, 2017.

[31] D. P. Kingma, T. Salimans, and M. Welling, "Improving Variational Inference with Inverse Autoregressive Flow," *CoRR*, vol. abs/1606.04934, 2016.

[32] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Józefowicz, and S. Bengio, "Generating Sentences from a Continuous Space," *CoRR*, vol. abs/1511.06349, 2015.