Modelling Symbolic Music: Beyond the Piano Roll

Christian Walder
CSIRO Data61
7 London Circuit, Canberra, 2604, Australia.
christian.walder@data61.csiro.au

June 7, 2016

Abstract

In this paper, we consider the problem of probabilistically modelling symbolic music data. We introduce a representation which reduces polyphonic music to a univariate categorical sequence. In this way, we are able to apply state of the art natural language processing techniques, namely the long short-term memory sequence model. The representation we employ permits arbitrary rhythmic structure, which we assume to be given. We show that our model is effective on four out of four piano roll based benchmark datasets. We further improve our model by augmenting our training data set with transpositions of the original pieces through all musical keys, thereby convincingly advancing the state of the art on these benchmark problems. We also fit models to music which is unconstrained in its rhythmic structure, discuss the properties of this model, and provide musical samples which are more sophisticated than previously possible with this class of recurrent neural network sequence models. We also provide our newly preprocessed data set of non piano-roll music data.

1 Introduction

Algorithmic music composition is an interesting and challenging problem which has inspired a wide variety of investigations [1]. A number of factors make this problem challenging. From a neuroscience perspective, there is a growing body of evidence which links the way we perceive music and the way we perceive natural language [2]. As such, we are highly sensitive to subtleties and irregularities in either of these sequential data streams.

The intensity of natural language processing (NLP) research has surged along with the availability of datasets derived from the internet. Within the field of NLP, the broad range of techniques collectively known as deep learning have proven dominant in a number of sub-domains — see *e.g.* [3]. Much of this research may find analogy in music, including algorithmic composition. Indeed, the very problem of data driven natural language generation is currently the subject of vigorous investigation by NLP researchers [4, 5].

Nonetheless, algorithmic composition presents unique technical challenges. In contrast to the words that make up text data, musical notes may occur contemporaneously. In this way, viewed as a sequence modelling problem, music data is multivariate. Moreover, the distribution of valid combinations of notes is highly multi-modal — while a number of musical chords may be plausible in a given context, arbitrary combinations of the notes which make up those chords may be rather implausible.

The outline of the paper is as follows. In section 2 we begin by providing an overview of the relevant work in NLP and algorithmic composition. Section 3 introduces the key elements of our approach: Long Short-Term Memory (LSTM) sequence modelling, our reduction to univariate prediction, our data representation, and our data augmentation scheme. In section 4 we present our experimental results, before summing up and suggesting some possible directions for future work in the final section 5.

2 Related Work

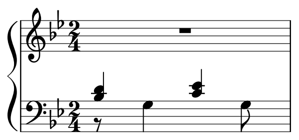
2.1 Natural Language Processing

Early NLP work focused on intuitively plausible models such as Chomsky's (arguably Anglo-centric) context free grammars, and their probabilistic generalizations. More recently however, less intuitive schemes such as *n-gram* models have proven more reliable for many real world problems [6]. In just the last few years, an arguably even more opaque and data driven set of techniques collectively known as deep learning have been declared the new state of the art on a range of tasks [3]. A key building block is the so called recurrent neural network (RNN), the example of which we are most concerned with is the LSTM of Hochreiter and Schmidhuber [7]. Like all RNNs, the LSTM naturally models sequential data by receiving as one of its inputs its own output from the previous time step. As clearly articulated in the original work [7], the LSTM is especially designed to overcome certain technical difficulties which would otherwise prevent the capturing of long range dependencies in sequential data. Nonetheless, the power of the model has been largely dormant until the ready availability of massively parallel computing architectures in the form of relatively inexpensive graphics processing units (GPUs). In a basic LSTM language model, the output $\boldsymbol{o}_t$ of the LSTM at time t is used to predict the next word $\boldsymbol{x}_{t+1}$, while the unobserved state $\boldsymbol{s}_t$ stores contextual information. The $\boldsymbol{x}_t$ typically encode words as *one hot* vectors with dimension equal to the size of the lexicon and a single non-zero element equal to one at the index of the word. Before inputting to the LSTM cell, the $\boldsymbol{x}_t$ are transformed linearly by multiplication with an *embedding matrix*, which captures word level semantics. This embedding may be learned in the context of the language model, or *pre-trained* using a dedicated algorithm such as [8].

2.2 Algorithmic Composition

Polyphonic music modelling may be viewed as a sequence modelling problem, in which the elements of the sequence are, for example, those sets of notes which are sounding at a particular instant in time. Sequence modelling has a long history in machine learning, and a large body of literature exists which studies such classic approaches as the hidden Markov model (HMM) [9]. Recently, more general RNN models have been shown to model musical sequences more effectively in practice [10].

RNNs have long been applied to music composition. The CONCERT model of Mozer [11] models a melody line along with its rhythmic structure, as well as an accompanying chord sequence. CONCERT uses the five dimensional pitch representation of Shephard [12] in order to capture a notion of perceptual distance. Another interesting feature of CONCERT is the treatment of rhythmic information, in which note durations are explicitly modeled. Aside from CONCERT, the majority


(a) staff notation

midi 63	0	0	0	0	0	0	1	1	1	0
midi 62	0	0	1	1	0	0	0	0	0	0
midi 60	0	0	0	0	0	1	1	1	1	0
midi 58	0	1	1	1	0	0	0	0	0	0
midi 55	0	0	0	1	1	1	1	0	1	0
t	0	0	0	0.5	1	1	1	1.5	1.5	2
Δt_{event}	1	1	1	0	1	1	0	0.5	0	0
Δt_{step}	0	0	0.5	0.5	0	0	0.5	0	0.5	0
ϵ_{onset}	0	1	0	0	0	1	0	0	0	0
ϵ_{offset}	0	0	0	0	1	0	0	1	0	1
target	70	74	67	-	72	75	-	67	-	-
lower bound	0	70	0	-	0	72	-	0	-	-

(b) our representation

Figure 1: The first bar of Tchaikovsky's *Song of the Lark* — see section 3.3 for a description.

of models employ a simple uniform time discretization, which severely limits the applicability of the models to realistic settings.

Allan and Williams used an HMM with hidden state fixed to a given harmonic sequence [13], and learned transition probabilities (between chords) and emission probabilities (governing note level realizations).

The suitability of the LSTM for algorithmic composition was noted in the original work [7], and later investigated by Eck and Schmidhuber for the blues genre [14], although the data set and model parameter space were both small in the context of current GPU hardware.

Finally, we note that other ideas specifically from NLP have also been used for modelling music, including hierarchical grammars [15, 16].

3 Our Approach

3.1 LSTM Language Model Architecture

We wish to leverage insights from state of the art language models, so we take a particular multi-layer LSTM as our starting point [17]. This model broadly fits our general description of the LSTM based language model in subsection 2.1, with several non-trivial computational refinements such as an appropriate application of *dropout* based regularization [18]. We will show that the model of [17] is effective in our context as well, by reducing the music modelling problem to a suitable form.

3.2 Reduction to Univariate Prediction

It is possible, as in the RNN-Restricted Boltzmann Machine (RNN-RBM) of [10], to modify the *output* or *prediction* layer of the basic RNN language model architecture, such that it supports multivariate prediction. As we shall see, our reduction to a simple univariate problem is at least as effective in practice. Moreover, by reducing the problem in this way, we can easily leverage future advances in the highly active research area of NLP.

The reduction is motivated by the problem of modelling a density over multisets $\boldsymbol{c} \in \mathcal{C}$ with underlying set of elements $\mathcal{X} = \{1, 2, \ldots, d\}$. For our purposes, $\mathcal{X}$ represents the set of d musical notes (or piano keys, say), and $\boldsymbol{c}$ is an unordered set of (possibly repeated) notes in a chord. Since $\mathcal{X}$ contains ordinal elements, we can define a bijective function $f : \mathcal{C} \to \mathcal{O}$, where $\mathcal{O}$ is the set of ordered tuples with underlying set of elements $\mathcal{X}$. Now, for $\boldsymbol{o} \in \mathcal{O}$ such that $\boldsymbol{o} = (o_1 \leq o_2 \leq \ldots, o_{|\boldsymbol{o}|})$, we can always write

$$p(\boldsymbol{o}) = \prod_{i=1}^{|\boldsymbol{o}|} p(o_i|o_1, o_2, \ldots, o_{i-1}). \tag{1}$$

Hence, we can learn (or infer) $p(\boldsymbol{c})$ by solving a sequence of univariate learning problems. If the cardinality $|\boldsymbol{o}|$ is unknown, we simply model an "end" symbol, which we may denote by $d + 1$.

This is similar in spirit to the *fully visible sigmoid belief network* (FVSBN) [19], which is in turn closely related to the *neural autoregressive distribution estimator* (NADE) [20]. Both the FVSBN and NADE have been combined with RNNs and applied to algorithmic composition, in [21] and [10], respectively. These methods employ a similar factorization to (1), but apply it to a binary indicator vector of dimension d, which has ones in those positions for which the musical note is sounding. Our approach is well suited to sparse problems (those in which $|\boldsymbol{o}| \ll d$), and also to unrolling in time in order to reduce the polyphonic music problem to a sequence of univariate prediction problems, as we explain in the next section. Indeed, to apply the unrolling in time idea of the next sub-section to an FVSBN type setup would require unrolling d steps per chord, whereas as we shall see our approach need only unroll as many steps as there are notes in a given chord.

3.3 Data Representation

Following from the previous section, our idea is essentially to model each note in a piece of music sequentially given the previous notes, ordered first temporally and then, for notes struck at the same time, in ascending order of pitch. Similarly to the previous subsection, this defines a valid probability distribution which — in contrast to the RNN-RBM but similarly to the RNN-NADE [10] — is tractable by construction.[1]

To achieve this, we employ the data representation of Figure 1. In this work, we assume the rhythmic structure, that is the note onset and offset times, is fixed. While we could model this as well, such a task is non-trivial and beyond the scope of this work.

In Figure 1, the musical fragment is unrolled into an input matrix (first ten rows), a target row, and a lower bound row. At each time step, the LSTM is presented with a column of the input, and should predict the value in the target row, which is the midi number of the next note to be "turned on". The lower bound is due to the ordering by pitch — notes with simultaneous onsets are ordered such that we can bound the value we are predicting, so we must incorporate this into the training loss function.

Roughly speaking, the "midi" rows of the input indicate which notes are on at a given time. We can only allow one note to turn on at a time in the input, since we predict one note at a time, hence the second column has only pitch 70 turned on, and the algorithm should ideally predict that the next pitch is 74 (given a lower bound of 70), even though musically these notes occur simultaneously. Conversely, we can and do allow multiple pitches to turn off simultaneously (as in columns 5 and 10), since we are not predicting these — the notes to turn off are dictated by the rhythmic structure (algorithmically, this requires some simple book keeping to keep track of which note(s) to turn off).

[1]For example, there no known tractable means of computing the exact normalization constant of an RBM, as is required for computing say the test set likelihoods we report in section 4.

The non-midi input rows 6 to 10 represent the rhythmic structure in a way which we intend to aid in the prediction. Δt_{event} is the duration of the note being predicted, Δt_{step} is the time since the previous input column, ϵ_{onset} is 1 if and only if we are predicting at the same time as in the previous column, ϵ_{offset} is 1 if and only if we are turning notes off at the same time as in the previous column, and t represents the time in the piece corresponding to the current column (in practice we scale this value to range from 0 to 1). In the figure, the time columns are all in units of quarter notes.

This representation allows arbitrary timing information, and is not restricted to a uniform discretization of time as in many other works, *e.g.* [10, 13]. A major problem with the uniform discretization approach is that in order to represent even moderately complex music, the grid would need to be prohibitively fine grained, making learning difficult. Moreover, unlike the "piano roll" approaches we explicitly represent onsets and offsets, and are able to discriminate between, say, two eighth notes of the same pitch following one another as opposed to a single quarter note.

3.4 Data Augmentation

Data augmentation provides a simple way of encoding domain specific prior knowledge in any machine learning algorithm. For example, the performance of a generic kernel based classifier was advanced to the then state of the art in digit recognition by augmenting the training dataset with translations (in the image plane, for example by shifting all images by one pixel to the right), yielding the so-called virtual support vector machine algorithm [22]. There, the idea is that if the transformation leaves the class label unchanged, then the predictive power can be improved by having additional (albeit dependent) data for training. It can also be rather effective to encode such prior knowledge directly in the model, however this is typically more challenging in practice [23], and beyond the scope of the present work.

Here we transpose our training data through all 12 (major or minor) keys, by transposing each piece by n semi-tones, for $n = -6, -5, \ldots, 4, 5$. To the best of our knowledge this method has not been used in this context, with many authors instead transposing all pieces to some target key(s), such as C/A minor as in [13] or C/C minor as in [10]. Although simple and widely applicable to all approaches, we will see that augmenting in this way will lead to a uniformly substantial improvement in predictive power on all the datasets we consider. Moreover, by augmenting we can avoid the non-trivial sub-problem of key identification, which is arguably not even relevant for pieces that modulate.

4 Experiments

In this section, we first provide an overview of our experimental setup, before proceeding to demonstrate the efficacy of our approach. We proceed with the latter in two steps. First, in subsection 4.2 we verify the effectiveness of our model and data representation by comparing with previous approaches to the modelling of piano roll data. Then, in subsection 4.3 we demonstrate our model on the new task of modelling non piano-roll data.

4.1 Implementation Details and Experimental Setup

After representing the data in the form depicted in Figure 1, our algorithmic setup is similar to [17], with two minor differences. The first is that some of the data columns do not have prediction targets (those with a dash in the target row of Figure 1). Here, we apply the usual LSTM recurrence but

Table 1: Mean test set log-likelihoods per time step, for the piano roll problems introduced in [10].

Model	Reference	Piano.	Nott.	Muse.	JSB.
DBN-LSTM[5]	[25]	-4.63	-1.32	-3.91	-3.47
HMM (with chord annotations)	[13]	—	—	—	-9.24
TSBN	[21]	-7.98	-3.67	-6.81	-7.48
RNN-NADE	[10]	-7.05	-2.31	-5.60	-5.56
LSTM	ours	-6.67	-2.06	-5.16	-5.01
LSTM (augmented)	ours	-5.43	-1.66	-4.46	-4.34
LSTM (augmented + pooled)	ours	-4.94	-1.57	-4.41	-4.45

Table 2: Average test set log-likelihoods per time step for the piano roll problems introduced in [10].

		Test			
		Piano.	Nott.	Muse.	JSB
Train	Piano.	-5.43	-3.26	-5.90	-7.76
	Nott.	-15.28	-1.66	-17.12	-18.40
	Muse.	-5.47	-3.00	-4.46	-6.49
	JSB	-13.83	-10.03	-16.16	-4.34

without any loss function contribution (in training) or output sampling (when generating samples from the trained model). The second difference is that the target value is lower bounded by a known value in some cases as described in subsection 3.3. The correct way to implement this is trivial: only those output layer logits whose indices satisfy the bound are considered at the output layer. In line with the parameter naming conventions in [17], we used two LSTM layers, an 800 dimensional linear embedding, a dropout "keep" fraction of 0.45, mini batches of size 50, and we limited the unrolling of the recurrence relation to 150 steps for computational reasons. We used the *Adam* optimizer with the recommended default parameters [24]. We optimized for up to 60 epochs, but terminated early if the validation likelihood did not improve (*vs.* the best so far) while the training likelihood did improve (*vs.* the best so far) four times in a row, at which point we kept the model with best validation score. This typically took 10 to 20 epochs in total, requiring up to around two days training for our largest problem. After training however, it only took around a second to sample an entire new piece of music, even with highly unoptimized code. Our implementation used Google's *tensorflow* deep learning library,[2] and we did most of the work on an NVIDIA Tesla K40 GPU.

4.2 Benchmark Tests

Our first test setup is identical to [10]. In line with that work we report log likelihoods rather than the *perplexity* measure which is more typical in NLP. Four datasets are considered, all of which are

[2]www.tensorflow.org

Table 3: Mean test set log-likelihoods per note for the midi data of [10].

Model	Piano.	Nott.	Muse.	JSB.
LSTM	-2.99	-1.18	-1.71	-1.32
LSTM (augmented)	-2.28	-0.99	-1.33	-1.07
LSTM (augmented + pooled)	-2.14	-0.99	-1.34	-1.25

derived from midi files:
Piano-midi.de is a collection of relatively complex classical pieces transcribed for piano [26].
Nottingham is a collection of folk tunes with simple melodies and block chords.[3]
MuseData is a collection of high quality orchestral and piano pieces from CCARH.[4]
J. S. Bach is a corpus of chorales harmonized by J.S. Bach, with splits taken from [13].

To compare with previous results [10, 21] we used the pre-processed data provided in piano roll format, which is a sequence of sets of notes lying on a uniform temporal grid with a spacing of an eighth note, and pre-defined train/test/validation splits. To adapt our more general model to this problem, we added an "end" symbol along with each set of notes, as per subsection 3.2. We also omitted the t row of Figure 1, which contains a form of "forward information". In this way, our model solved precisely the same modelling problem as [10], and our likelihoods are directly comparable.

The results in Table 1 include the best model from [10], as well as that of [21]. The *LSTM* row corresponds to our basic model, which performs encouragingly, outperforming the previous methods on all four datasets. The *LSTM (augmented)* row uses the data augmentation procedure of subsection 3.4, which proves effective by outperforming the basic model (and all previous approaches) convincingly, on all datasets. The only exception is the DBN-LSTM model[5] Finally, in addition to the augmentation, we pooled the data from (the training and validation splits of) all four datasets, and tested on each dataset, in order to get an idea of how well the model can generalize across the different data sets. The difference here is minor in all cases but piano-midi.de, where we see further improvement, which is interesting given that piano-midi.de appears to be the most complex dataset overall. There is a small deterioration of this *LSTM (augmented + pooled)* model performance on the J.S. Bach test set, however Table 2, which is a cross table of performance for training on one dataset and testing on another, shows that this is not the complete picture. Indeed, models trained and tested on different datasets unsurprisingly do far worse than our pooled data model. This suggests that our model with pooling and augmentation as the best overall for algorithmic composition.

4.3 A New Benchmark Problem

In the previous section, we adapted our model to handle the case where time is uniformly discretised, but the number of notes sounding at each time step is unknown. In this section, we use the original midi file data to construct a different, arguably more musically relevant problem. The data, which

[3] ifdo.ca/~seymour/nottingham/nottingham.html
[4] www.musedata.org
[5] However, while not stated in [25], the DBN-LSTM likelihoods are well known merely to be optimistic bounds on the true values.

we have preprocessed as described in the supplementary material, and made available online[6] is now in the form of sets of tuples indicating the onset time, offset time, midi note number, and part number (*i.e.* midi track number) of each event in the original midi file. We model the midi note names conditional on the rhythmic structure (onset and offset times). We also ignore the part numbers, although future work should utilize these, given that individual parts in a piece of music should ideally have a coherence of their own.

Before modelling the above data, we applied a simple preprocessing step in order to clean the timing information. This step was necessary due to the imprecise timing information in some of the source midi files, and would have no effect on "clean" midi data derived from musical scores. In particular, we quantized the onset and offset times to the nearest 96th of a quarter note, unless this would collapse a note to zero duration in which case we left the onset and offset times unchanged. This was especially helpful for the case of the *piano-midi.de* data, where the onset and offset times often occur extremely close to one another. There are two distinct reasons why we apply this pre-processing step:

1. By quantizing the offset times, we reduce the number of columns in our data representation, since more notes can "turn off" simultaneously in a single input column. See Figure 1 and the corresponding description of the data representation in subsection 3.3. This in turn effectively allows the LSTM to model longer range temporal dependencies.

2. By quantizing the onset times, our ordering technique (see subsections 3.2 and 3.3), which orders first by time and then by pitch, will more consistently represent those onsets which are close enough to be considered practically simultaneous.

The results in Table 3 are now in the form of mean log-likelihood per note, rather than per time step, as before. In the setup of [10], a uniformly random prediction would give a log likelihood of $-\log(2^d)$ (we assume no unison intervals) where d is the number of possible notes. For $d = 88$ notes (spanning the full piano keyboard range of A0 to C8) as in [10], this number is approximately -61. For our new problem however, a uniformly random prediction will have a mean log-likelihood per note of $-\log(d)$ which is approximately -4.47 for $d = 88$. Overall then, the results in Table 3 are qualitatively similar to the previous results, after accounting for the expected difference of scale.

Note that it is not practical to compare with the previous piano-roll algorithms in this case. It turns out that the greatest common divisor of note durations in our data dictates a minimum temporal resolution of 480 "ticks" per quarter note, rendering those algorithms impractical (note that the piano roll format used in the previous work and in subsection 4.2 used a resolution of an eighth note, or two ticks per quarter note).

Importantly however, we can generate music with an arbitrarily complex rhythmic structure. We generated a large number of example outputs by sampling from our model given the rhythmic structure from each of the test set pieces contained in the MuseData corpus.[6] We also included a single sample output in the supplementary material, two bars of which are notated in Figure 2.

We also attempted to visually identify some of the structure we have captured. For the pooled and augmented model, we visualized the embedding matrix which linearly maps the raw inputs into what in NLP would be termed a latent semantic representation. A clear structure is visible in Figure 3. The model has discovered the functional similarity of notes separated by an interval of one or more octaves. This is interesting given that all notes are orthogonal in the raw input representation, and moreover that the training was unsupervised. In NLP, similar phenomena have

[6] Our data and output audio samples: http://bit.ly/1PqNTJ2

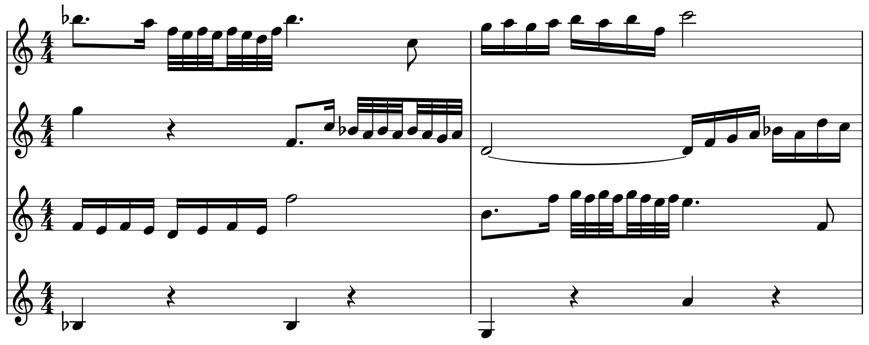


Figure 2: Two bars of the model's output, featuring a relatively complex rhythmic sequence which occurs 52 seconds into the audio sample included in the supplementary material.

been observed [8], as well as more sophisticated relations such as (in embedding space) "*Queen* - *King* + *brother* ≈ *sister*". We tried to identify similar higher order relationships in our embedding, by comparing intervals, analyzing mapped counterparts to the triads of the circle of fifths, *etc.*, but the only clear relationship we discovered that of the octave, described above. This does not mean the model hasn't identified these relationships however, since the embedding is only the first layer in a multi-layer model.

5 Summary and Outlook

We have presented a model which reduces the problem of polyphonic music modelling to one of modelling a temporal sequence of univariate categorical variables. By doing so, we were able to apply state of the art neural language models from natural language processing. We further obtain a relatively large improvement in performance by augmenting the training data with transpositions of the pieces through all musical keys — a simple but effective approach which is directly applicable to all algorithms in this domain. Finally, we obtain a moderate additional improvement by pooling data from various sources. An important feature of our approach is its generality in terms of rhythmic structure — we can handle any fixed rhythmic structure and are not restricted to a uniform discretization of time as in a piano roll format. This lead us to derive a new benchmark data set, which we believe will be useful for further advancing the state of the art in algorithmic composition. Future work could 1) model the rhythmic information, rather than conditioning on it as we do here, 2) provide a mechanism for user constraints on the output, as has been done with Markov models by applying branch and bound search techniques [27], or 3) introduce longer range dependencies and motif discovery using attentional models (see *e.g.* [28]).

6 Acknowledgments

We gratefully acknowledge NVIDIA Corporation for donating the Tesla K40 GPU used in this work.

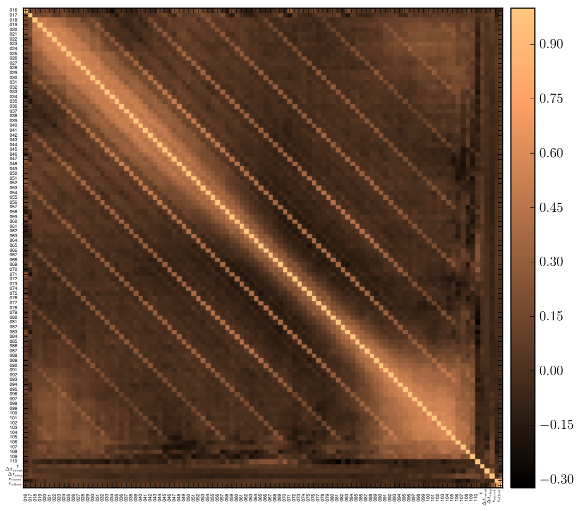


Figure 3: Correlations between the columns of the learned *embedding matrix*, which maps inputs to higher dimensional "semantic" representations. The row/column labels can be seen by zooming in on an electronic copy: the last 5 rows/columns correspond to the last five rows of the input (upper) part of Figure 1 (b), the remaining rows/columns correspond to sequential midi numbers. The off-diagonal stripes suggest that the model has discovered the importance of the octave.

References

[1] Jose D. Fernandez and Francisco J. Vico. Ai methods in algorithmic composition: A comprehensive survey. *J. Artif. Intell. Res. (JAIR)*, 48:513–582, 2013.

[2] A.D. Patel. *Music, Language, and the Brain*. Oxford University Press, 2010.

[3] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005, 2013.

[4] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[5] Alex Mathews, Lexing Xie, and Xuming He. SentiCap: generating image descriptions with sentiments. In *The Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, Phoenix, USA, feb 2016.

[6] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[8] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, October 2007.

[10] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proceedings of the Twenty-ninth International Conference on Machine Learning (ICML'12)*. ACM, 2012.

[11] Michael C. Mozer. Neural network music composition by prediction: exploring the benefits of psychoacoustic constraints and multi-scale processing. In *Connection Science*, pages 247–280, 1994.

[12] Roger Shepard. Geometrical approximations to the structure of musical pitch. *Psychological Review*, 89:305–333, 1982.

[13] Moray Allan and Christopher K. I. Williams. Harmonising chorales by probabilistic inference. *Advances in Neural Information Processing Systems 17*, 2005.

[14] Douglas Eck and Juergen Schmidhuber. A first look at music composition using lstm recurrent neural networks. Technical report, 2002.

[15] Mark Granroth-Wilding and Mark Steedman. Statistical parsing for harmonic analysis of jazz chord sequences. In *Proceedings of the International Computer Music Conference*, pages 478–485. International Computer Music Association, 2012.

[16] W. Bas De Haas, Jose Pedro Magalhaes, Remco C. Veltkamp, and Frans Wiering. HARMTRACE: Improving Harmonic Similarity Estimation Using Functional Harmony Analysis. In *ISMIR*, 2011.

[17] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014.

[18] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[19] Radford M. Neal. Connectionist learning of belief networks. *Artif. Intell.*, 56(1):71–113, July 1992.

[20] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. *JMLR: W&CP*, 15:29–37, 2011.

[21] Zhe Gan, Chunyuan Li, Ricardo Henao, David E Carlson, and Lawrence Carin. Deep temporal sigmoid belief networks for sequence modeling. In C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2458–2466. Curran Associates, Inc., 2015.

[22] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. *Artificial Neural Networks — ICANN'96*, 1112:47–52, 1996.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[25] Raunaq Vohra, Kratarth Goel, and J. K. Sahoo. Modeling temporal dependencies in data using a DBN-LSTM. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Campus des Cordeliers, Paris, France, October 19-21, 2015*, pages 1–4. IEEE, 2015.

[26] Graham E. Poliner and Daniel P. W. Ellis. A discriminative model for polyphonic piano transcription. *EURASIP J. Adv. Sig. Proc.*, 2007, 2007.

[27] F. Pachet and P. Roy. Markov constraints: steerable generation of markov sequences. *Constraints*, 16(2):148–172, March 2011.

[28] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.

Supplementary Material Modelling Symbolic Music: Beyond the Piano Roll

Christian Walder
CSIRO Data61
7 London Circuit, Canberra, 2604, Australia.
christian.walder@data61.csiro.au

June 7, 2016

A Data

A.1 Introduction

In this appendix we provide an overview of the symbolic music datasets we offer in pre-processed form[1]. Note that the source of these datasets is the same set of midi files used in [1], which also provides pre-processed data. That work provided "piano roll" representations, which essentially consist of a regular temporal grid (of period one eighth note) of on/off indicators for each midi note number. While the piano roll is an excellent simplified music format for early investigations into symbolic music modelling, it does have several limitations, as discussed in the main text. To name one such limitation, the piano roll format does not explicitly represent note endings, and therefore cannot differential between, say, two successive eighth notes, and a single quarter note.

To address these limitations, we have extracted additional information from the same set of midi files. Our goal is to represent the performance (or sounding) of notes by when they begin and end, rather than whether they are sounding or not at each time on a regular grid. The representation we adopt consists of sets of five-tuples, each of which describes the sounding (and termination) of a single note. Each such note event is represented by the following integers:

- piece number (corresponding to a midi file),
- track (or part) number, defined by the midi channel in which the note event occurs,
- midi note number, ranging 0-127 according to the midi standard, and 22-105 inclusive for the data we consider here,

[1]The data is available for download here: http://bit.ly/1PqNTJ2

- note start time, in “ticks”, (2400 ticks = 1 beat = one quarter note),
- note end time, also in ticks.

We split the pieces into the same three (train/validation/test) splits as used in [1].

Please refer to the data archive itself[1] for a detailed description of the format. A summary of the four datasets is provided in Table 1.

A.2 Preprocessing

We applied the following processing steps and filters to the raw midi data.

- Combination of piano “sustain pedal” signals with key press information to obtain equivalent individual note on/off events.
- Removal of duplicate/overlapping notes which occur on the same midi channel (while not technically allowed, this still occurs in real midi data due to the permissive nature of the midi file format). Unfortunately, this step is ill posed, and different midi software packages handle this differently. Our approach involves processing notes sequentially in order of start time, and ignoring those note events that overlap a previously added note event.
- Removal of midi channels with less than two note events (these occurred in the MUS dataset, and were always information tracks containing authorship information and acknowledgements, *etc.*).
- Removal of percussion tracks. These occurred in some of the Haydn symphonies and Bach Cantatas contained in the MUS dataset. It is important to filter them as the percussion instruments are not pitched, and hence the midi numbers in these tracks are not comparable with those of pitched instruments, which we aim to model.
- Re-sampling of the timing information to a resolution of 2400 ticks per quarter note, as this is the lowest common multiple of the original midi file resolutions (see Table 1).

A.3 Exploratory analysis

We provide some basic exploratory plots in figures 1–4.

The **Note Distribution** and **Number of Notes Per Piece** plots are self explanatory.

Note that the **Number of Parts Per Piece** (lower left sub figure) is fixed at one for the entire JSB dataset. This is due to an unfortunate lack of midi track information in these files, many of which are in fact four part harmonies. The pieces in the NOT dataset feature either one part (in the case of pure melodies) or two (in the case of melodies with associated chord accompaniments). The PMD dataset features up to six parts (for a three-part Bach fugue in which left

Dataset	Long Name	Source	Total Pieces	Midi Resolution
PMD	`piano-midi.de`	[2, 1]	124	480
JSB	J.S Bach Chorales	[3, 1]	382	100
MUS	MuseData	[4, 1]	783	240
NOT	Nottingham	[5, 1]	1037	480

Table 1: A summary of the datasets used in this study.

and right hands are tracked separately). MUS features up to 27 parts (for Bach's *St. Matthew's Passion*). Note that this information is for future reference only — we have yet to take advantage of the part information in our models.

The least obvious sub-figures are those on the lower-right labeled **Peak Polyphonicity Per Piece**. Polyphonicity simply refers to the number of simultaneously sounding notes, and this number can be rather high. For the PMD data, this is mainly attributable to musical "runs" which are performed with the piano sustain pedal depressed, for example in some of the Liszt pieces. For the MUS data, this is mainly due to the inclusion of large orchestral works which feature many instruments.

References

[1] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proceedings of the Twenty-ninth International Conference on Machine Learning (ICML'12)*. ACM, 2012.

[2] Graham E. Poliner and Daniel P. W. Ellis. A discriminative model for polyphonic piano transcription. *EURASIP J. Adv. Sig. Proc.*, 2007, 2007.

[3] Moray Allan and Christopher K. I. Williams. Harmonising chorales by probabilistic inference. *Advances in Neural Information Processing Systems 17*, 2005.

[4] `www.musedata.org`.

[5] `ifdo.ca/~seymour/nottingham/nottingham.html`.

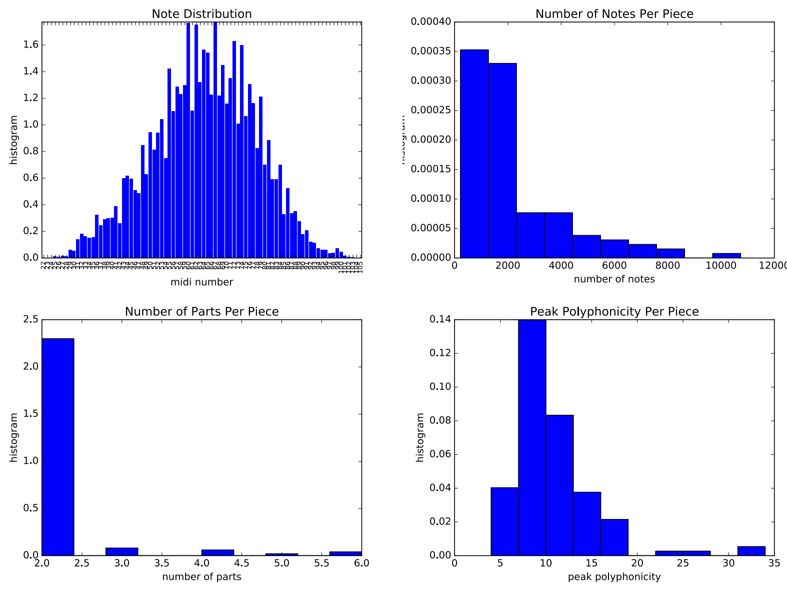


Figure 1: Summary of the PMD dataset.

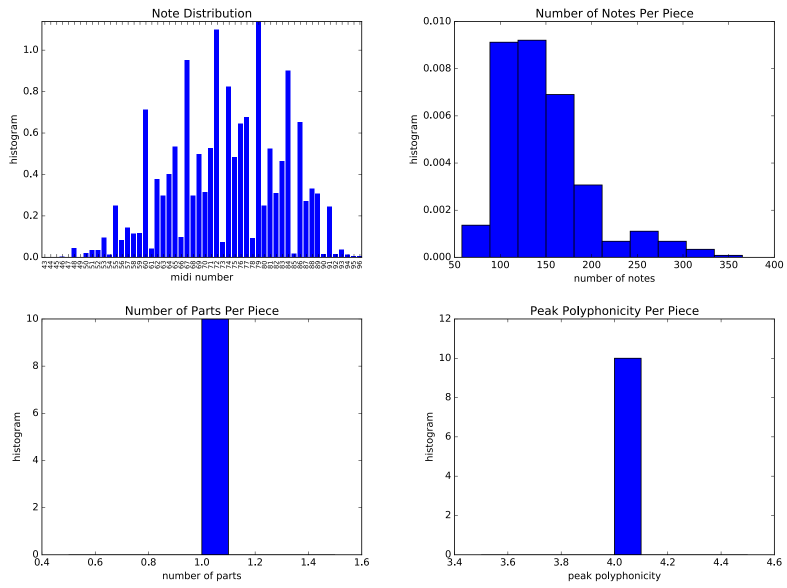


Figure 2: Summary of the JSB dataset.

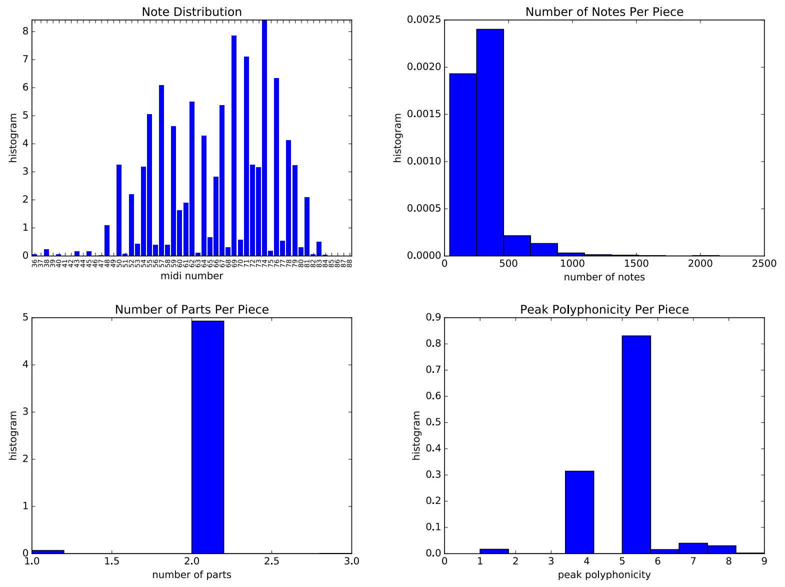


Figure 3: Summary of the NOT dataset.

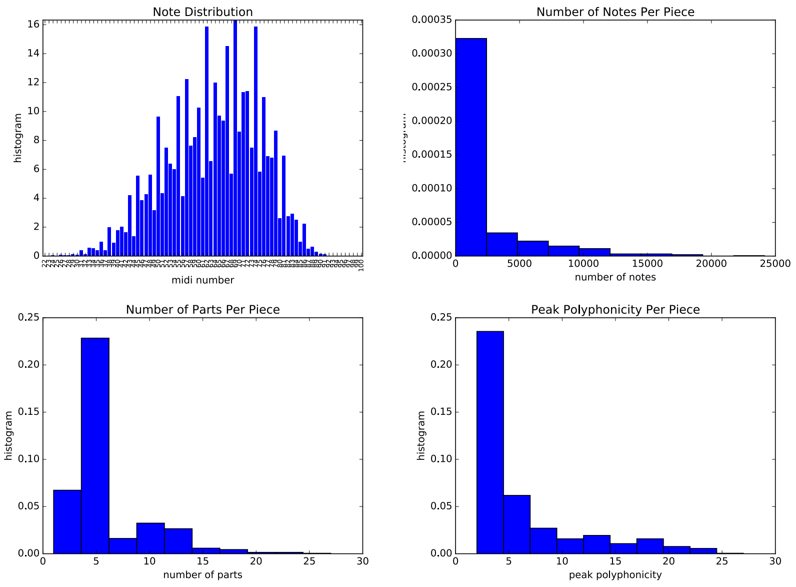


Figure 4: Summary of the MUS dataset.