MEASURE BY MEASURE: AUTOMATIC MUSIC COMPOSITION WITH TRADITIONAL WESTERN MUSIC NOTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we present a system that is capable of generating long polyphonic music given number of measures, up to hundreds of measures. This is achieved by creating a measure model that imitates the object hierarchy used in common encodings of traditional western music notation. On top of this, we construct a inter-measure context model that spans the entire composition.

1 INTRODUCTION

Music, largely, is a form of art with no tangible existence. Throughout the history, people have developed several notation systems to record and formalize different types of composing practices. Traditional Western music notation and piano roll notation (the format that MIDI uses) are two examples. Although such notation systems were invented to facilitate music composition, performance and analysis, the form of music notation itself has a non-negligible effect on these activities. For example, it is more effortless for composers to stay in a certain key in a music piece when traditional Western music notation is used: in this writing system, key-relevant concepts such as key signature, note spellings and accidentals are built-in. As a contrast, piano roll notation does not encode such concepts and it requires extra effort for users to infer such attributes.

In this work[1], for better modeling music pieces originally written in the traditional Western music notation system, we try to utilize the common object hierarchy (movement-measure-voice-chord-pitch, see Sec.3.1) as notated in traditional Western music notation to simplify the problem of automatic music composition, and also creates notations that are human-readable (we assume the piano roll notation is hard to read by a common human musician). It is motivated by observing drawbacks of music encoding inherent in many music generation models.

From the performance (or the realization of a composition) perspective, music can be conceptualized as a discrete event time series: it is a sequence of note events spreading out along the time axis. For properly encoding music for processing, most previous works either choose to represent music as a sequence of key stroke events or other composite symbols, e.g. Jaques et al. (2017); Roberts et al. (2017); Bretan et al. (2016); Walder (2016), or simply treat the entire piano rolls as an image by evenly sampling the time axis into a fixed grid with two axis being time and pitch, e.g., Hadjeres et al. (2017); Huang et al. (2017); Yan et al. (2018). The former encoding works well with monophonic melody generation; however, it is not that intuitive when it comes to polyphonic music generation. One can choose to merge multiple simultaneous streams of events into a single sequence, in which case, two simultaneous notes would be represented by two events in the sequence with inter-onset time interval 0 or equivalent, e.g. (Huang et al., 2019b). However, it would make the sequence length much longer, and eliminate explicit composability between streams. One can also simply make a slice by slice representation without discretizing the time axis by representing all concurrent notes in a key frame as a k-hot vector (Walder, 2016). However, it loses dependencies between different pitches. The latter equally sampled grid-based approach works naturally for polyphonic music, where multiple monophonic streams of notes are aligned and coupled with each other. However, as the rhythm becoming more complex, the combination between different voices makes the step size prohibitively large for constructing the grid. For example, to authentically encode Mozart's string

[1]See `https://sites.google.com/view/pjgbjzom` for generation examples

quartets, one would need at least 48 steps for a single quarter note, as there are 16th note lengths in one voice divided by a triplet in another voice.

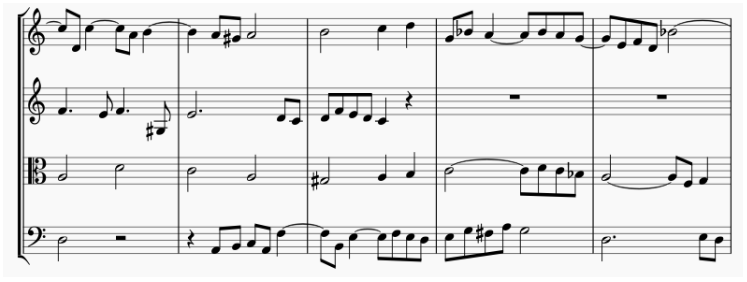


Figure 1: From the Art of Fugue, Bach. It is clear that at the inter-measure level it is a grid.

For better handling the complexity of music composition and also creating human-readable notations, we choose to model the traditional music notation system itself, which is different from all approaches above-mentioned. We noticed that traditional music notation at the inter-measure level is by itself a grid (see Fig.1), where each cell (measure) may contain a variable number of voices, and each voice may contain a variable number of note events. The concept of measure in music notation originates from annotating synchronization between voices, and later, recurring patterns of accents (meter) (ran, 2004). We utilize this fact by making a structured *measure encoder-decoder* with a hierarchical structure that directly corresponds to the object hierarchy in traditional music notation. Measures in our model resembles words in natural language generation; the difference is that in our model a measure encoder-decoder is used instead of a simple word embedding lookup and a softmax output layer. The use of a measure encoder-decoder coincides with Bretan et al. (2016), but they simply select measures from a predefined database to generate monophonic Jazz melody lines. In the proposed model, everything is handled event by event within a measure, and at the inter-measure level, this representation forms a dense grid, which significantly reduces the sequence length of a complete movement. Also, it accepts a variety of model architectures (RNN layers, Convolution layers, self-attentive layers, etc.).

The difficulties in designing such a structured autoencoder for measures are as follows: a) the model should be able to count the beats so that it knows how to generate and terminate properly; b) when it comes to music with strong polyphony (as opposed to homophony), the model should be able to attend/align a single event with the events around the same beat positions in other staves. We solve these two problems by introducing a set of techniques including duration normalization, within-beat position signals, cross-beat duration splitting, to make it easy to count, and we also create a fix-sized position-content associative memory for implicit alignment. Details of these techniques will be presented later.

This paper is structured as follows: in section 2, we give a description on the paradigm that our system is based on. In section 3, we present our proposed model on generating long polyphonic music composition. In section 4, we give out details regarding our experiments.

2 Background

In this section, we give a brief review on the generative framework our work is based on.

Conditionally specified distributions Learning and sampling using a set of conditional distributions has a long history (Hofmann, 2000; Heckerman et al., 2000; Arnold et al., 2001; Van Buuren et al., 2006). Given a set of conditional distributions $\{p(x_i|\mathbf{x}_{-i})|\forall i\}$ that predicts a single variable x_i given all its dependencies $\mathbf{x}_{-i}$, an implicit joint distribution is uniquely specified if all conditional distributions are compatible. In practice, when using an approximator, we usually relax the requirement of compatibility between conditional distributions.

To sample from this implicit distribution, (psedo-)Gibbs sampling is applied using this set of conditional distributions: firstly initialize the entire configuration and then iterate over all variables

$x_1, x_2, \ldots, x_N$ in a specific or random order multiple times to update one variable at a time from its conditional distribution.

This framework has been used for music generation in Hadjeres et al. (2017), Yan et al. (2018), etc. The advantage of this framework is that it allows more flexible conditioning, and meanwhile, this framework learns to predict a single variable with more observations, which is often easier compared to an auto-regressive model.

Ancestral Sampling given an arbitrary order For a given sequence of indices $j_0, j_1 \ldots j_{N-1}$ over all positions, we define a sequence of corresponding masks $\mathbf{m}_0, \mathbf{m}_1, \ldots \mathbf{m}_{N-1}$, where positions $\{j_k | k \geq i\}$ are masked $(= 0)$ in $\mathbf{m}_i$, i.e., values are hidden. By applying this sequence of masks to the full configuration $\mathbf{x} = x_0, x_1, \ldots, x_{N-1}$, we obtain a factorization of the joint distribution and sample data accordingly:

$$p(\mathbf{x}) = \prod_{s=0,1,\ldots,N-1} p(x_{j_s} | \mathbf{x} \odot \mathbf{m}_{j_s}), \tag{1}$$

where $\odot$ represents the masking operation. This factorization was utilized by Neural Autoregressive Density Estimation (NADE) (Uria et al., 2016), which allows sampling the whole configuration of all variables in a single sweep, given an sampling order.

In our work, during training, our model learns to predict arbitray masked positions with mask sampled from a predefined distribution;for sampling, we firstly initialize the full configuration using ancestral sampling, and then refine the result using (pseudo-)Gibbs sampling, which is similar to Huang et al. (2019a).

3 METHOD

Our model uses a measure encoder-decoder with a structure that imitates the object hierarchy of the Western music notation system. On top of it, we build a movement-wide inter-measure context model. The measure encoder-decoder serve as the basic input-output layers for our system.

In sec.3.1, we give details on the measure autoencoder that imitates the object hierarchy of traditional Western music notation. In sec.3.2 we briefly introduce the inter-measure context model used in this work.

3.1 MEASURE MODEL

Figure 2 illustrates the common object hierarchy used in most major music notation systems, e.g, Sibelius, Finale, Musescore, and also the music score open format, Musicxml.

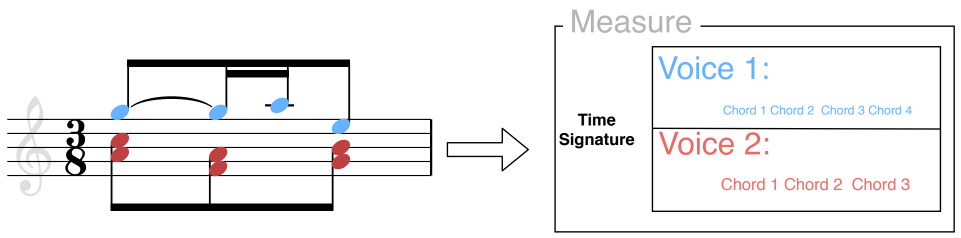


Figure 2: The common object hierarchy for most major music notation software

Each *measure* has a beat unit specified by the denominator of their time signature (e.g., quarter, eighth), and contains a variable number of *voices*. Each *voice* is a variable-length sequence where each step is a *chord-duration* tuple. A *chord* is a set consisting of *pitch-tie* tuples. A *pitch* is represented as a tuple of *diatonic steps* and *accidentals*. A *tie* indicates whether or not this note appears as a continuation of the previous note of the same *pitch* in the same *voice*. Each *pitch* is allowed to have its own *tie* status. In this work, a single note is also referred as a *chord*, as a general treatment to reflect the fact that a single note is a *chord* with only one element. Rest symbol is represented by a *chord* with cadinality zero, that is an empty set. The numerator of time signature can be calculated from the beat unit and the total number of beats within the measure; therefore it is omitted from the our model.

Figure 3 provides a concise, higher-level summary of the measure model. It contains an encoder and a decoder following this object hierarchy in an object-oriented way. Objects modeled are *duration*, *chord*, *voice*, and *measure*. Each object model implements an encoder $[\mathbf{h}, \mathbf{M}] = \phi(\cdot)$ and a decoder $p(\cdot|\mathbf{c}, \widetilde{\mathbf{M}})$, where $\mathbf{M}$ is a *position-indexed memory matrix*, $\mathbf{h}$ is the embedding vector produced by the encoder, and $\mathbf{c}$ is the input to the decoder for obtaining the resulting part(staff)-wise measure.

All decoders are autoregressive models: they invoke lower level decoders for producing lower level objects, and they invoke lower-level encoders for constructing feedback to the model on the lower level object produced (for the lowest level, they directly use embedding lookups).

For notational convenience, we reuse the same symbol $\mathbf{c}$, $\mathbf{h}$ for every sub-level object model. We will talk about the detailed design for each object model in the following subsections.

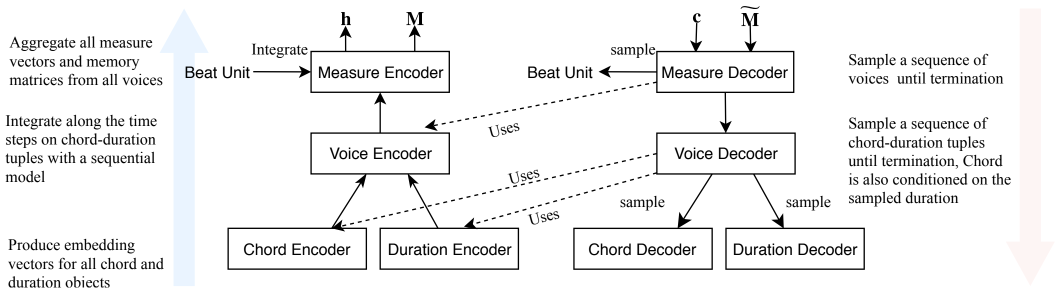

Aggregate all measure vectors and memory matrices from all voices
Integrate along the time steps on chord-duration tuples with a sequential model
Produce embedding vectors for all chord and duration obiects
h
M
Integrate
Beat Unit
Measure Encoder
Voice Encoder
Chord Encoder
Duration Encoder
c
M̃
sample
Beat Unit
Measure Decoder
Voice Decoder
Uses
Uses
Uses
sample
sample
Chord Decoder
Duration Decoder
Sample a sequence of voices until termination
Sample a sequence of chord-duration tuples until termination, Chord is also conditioned on the sampled duration

Figure 3: High level description of the proposed hierarchical measure model.

3.1.1 CHORD ENCODER-DECODER

As mentioned, a *chord* is a set of *pitch-tie* tuple, one single note is represented by a *chord* object with cardinality 1 and the rest symbol is represented as an empty set. As a basic building block, We use the same factorized pitch embedding as proposed in Yan et al. (2018), which encodes multiple aspects, i.e., octave equivalence, diatonic step, chromatic step of a single *pitch*. We denote *chord* $= \{(pitch_i, tie_i)\}_{i \in \mathcal{I}}$, where *pitch* represents the *pitch* and *tie* indicates whether or not this *pitch* is a continuation of the previous note with the same *pitch* within the same *voice*.

Chord Decoder Elements in a chord are orderless. We put these elements into an ordered sequence *chord*$_\pi$ with π denoting the order. We use $k = 0, \ldots, |\mathcal{I}| - 1$ to denote indices of this sequence. For sequentially generating set elements, we use an autoregressive model. Here we augment the candidate set of *pitch* with a special *terminate* symbol such that the last pitch in the sequence $pitch_{|\mathcal{I}|} = terminate$. Therefore we have

$$\begin{aligned} p(chord_\pi|\mathbf{c}) = \quad & p(pitch_{|\mathcal{I}|}|pitch_{<|\mathcal{I}|}, tie_{<|\mathcal{I}|}, \mathbf{c}) \\ & \prod_{k=0}^{|\mathcal{I}|-1} p(pitch_k, tie_k|pitch_{<k}, tie_{<k}, \mathbf{c}) \end{aligned} \tag{2}$$

During training we shuffle the order of elements in a chord to take permutation invariance of a set into account. We also consider the dependency between *pitch* and *tie* by factorizing

$$\begin{aligned} & p(pitch_k, tie_k|pitch_{<k}, tie_{<k}, \mathbf{c}) \\ & \qquad = p(pitch_k|pitch_{<k}, tie_{<k}, \mathbf{c})p(tie_k|pitch_k, pitch_{<k}, tie_{<k}, \mathbf{c}). \end{aligned} \tag{3}$$

We use a single layered GRU for this autoregressive model. The initial state of GRU is computed by feeding the input vector $\mathbf{c}$ into a linear projection layer, followed by a $\tanh$ activation function. During sampling, at step k, we firstly sample $pitch_k$ conditioned on the hidden state of GRU and $\mathbf{c}$, which is the input to this *chord* decoder module. If the sampled $pitch_k$ is the *terminate* symbol. terminate this process; otherwise we sample tie_k, the tie status, conditioned on the hidden state of

GRU, $\mathbf{c}$, and the pitch embedding $\mathbf{pitchEmbed}_k$ of the recently sampled pitch_k. Both of these two steps use two-layer feedforward neural networks before the softmax output layer. We then use the embedding vector ($\mathbf{noteEmbed}_k$, see below) for this recently sampled $(\mathit{pitch}_k, \mathit{tie}_k)$ tuple as the feedback to the GRU, and then repeat this process until termination.

Note embedding We obtain the embedding vector, denoted by **noteEmbed** for the *pitch-tie* tuple by

$$\mathbf{noteEmbed} = \mathbf{pitchEmbed} \odot \mathbf{tieEmbed} \tag{4}$$

where $\odot$ represents element-wise multiplication. Multiplication is used here for breaking the linearity of the sum aggregation used by the chord encoder (see below), so that a tie can be associated with a particular pitch in a chord

Chord encoder The chord encoder is more straightforward:

$$\mathbf{h} = \phi(\mathit{chord}) = \Phi(\sum_{\mathit{note}=(\mathit{pitch},\mathit{tie})\in \mathit{chord}} \mathbf{noteEmbed}(\mathit{note})) \tag{5}$$

Where $\Phi(\cdot)$ is a linear projection, which maps the summations of note embeddings to the desired dimension.

3.1.2 Duration encoder-decoder

Duration encoder *Duration* representable without a *tie* in traditional music notation is obtained by recursively dividing a *beat unit*, e.g., a quarter length, into 2,3,5...equal segments. The number of combinations allowable is huge. Therefore, for making the *duration* representation universal and better preserving the information of the division level, we directly encodes a legal note duration as a multiplication of prime exponents $\mathbf{d} = [d_0, d_1, \ldots]$ such that:

$$\text{duration} = (2^{d_0}3^{d_1}\ldots)\text{beatUnit} \tag{6}$$

We assign an embedding vector for every possible digit, and it is shared for all digit positions $d_i, i = 0, 1, \ldots$. We simply choose $[-4, +4]$ as the range for the digits. We use $[2, 3, 5, 7, 11]$ as the prime base in our experiment. The encoder of a duration is simply a linear projection of the concatenation of all digit embeddings for all prime bases.

Duration decoder For the decoder, we consider the dependency between these digits. Therefore we use a GRU to sample each digits autoregressivly.

3.1.3 Single Voice encoder-decoder

As mentioned above, each *voice* inside a *measure* is a sequence with each step being a *chord-duration* tuple $(\mathit{chord}_i, \mathbf{d}_i)$, where i denotes the index within the sequence.

Preprocessing In order to make the autoregressive encoder/decoder easy to count the beats so that it can generate sequences of correct total durations and also receive guidance for synchronizing with simultaneous events at other parts (staffs), we do the following:

- **Duration normalization:** We scale all the durations in a measure such that each integer beat corresponds to a *primary division* of the measure. For example, for time signature 6/8, we transform it into a sequence with a total length of two beats using the dotted quarter length as the beat unit, which is a conventional practice to count beats in traditional Western music theory; durations in a measure with time signature 3/4 have the same normalized total length as those in a 3/8 measure. By doing this, we only have measures with length 2(duple), 3(triple), or 4(quadruple) beats for all pieces in our current training corpus.
- **Within-beat position signal as extra input:** Since we have normalized the total duration of all measures, we then use a sinusoidal within-beat position signal $\mathbf{b}_i$ as an extra input to both encoder and decoder for every step.

$$\mathbf{b}_i = [\sin(2\pi\delta_i), \cos(2\pi\delta_i)] \tag{7}$$

  where δ_i is the normalized onset position within a measure for the step i, whose value is wrapped to $[0, 1)$

- **Cross-beat note splitting:** Since the within-beat position is circular, we desire every integer beat position to appear in the sequence to avoid aliasing issue where a note crosses multiple beats, which bypasses circles of the position signal. We add another preprocessing step: we split all notes whenever their spans cross integer beat position into multiple tied notes. Another benefit of doing this is that it limits range of durations. One thing worth mentioning is that this is also a notational convention for instrumental music (see Stone (1980)) when a note starts at a non-primarily divided position and crosses such primary position for making human musicians easy to count.

Voice encoder For encoding a single *voice* in a *measure* into a vector representation, we concatenate embedding vectors of *chord* (see Sec.3.1.1), *duration* (see Sec.3.1.2) and *within-beat position signal* for each *chord-duration* tuple, $(chord_i, \mathbf{d}_i)$, and then feed this sequence of concatenated values into a bidirectional GRU layer to obtain a sequence of contextual embeddings $[\mathbf{s}_0, \mathbf{s}_1, \ldots]$. We then pass the last hidden state of GRU through a linear projection layer to get the final embedding vector $\mathbf{h}$ for this *voice*.

We run another bidirectional GRU layer only with the concatenation of *duration* embeddings and the *within-beat position signal* to obtain a sequence of embeddings $[\mathbf{t}_0, \mathbf{t}_1 \ldots]$ purely from the perspective of rhythm/position related context. We create a fix-sized position indexed memory matrix of this *voice* by:

$$\mathbf{M} = \sum_i \text{key}(\mathbf{t}_i)\text{val}(\mathbf{s_i})^T, \tag{8}$$

where $\text{key}(\mathbf{x}) = \text{softmax}(\mathbf{W}_{key}\mathbf{x})$ and $\text{val}(\mathbf{x}) = \mathbf{W}_{val}\mathbf{x}$. This constant-sized memory matrix makes it easier for the inter-measure model to attend back into the sequence, and also, after summing up memory matrices from synchronous measures, entries with similar positional/rhythmic context get aggregated, which provides an extra reference for the decoder on the aggregated information of a certain positional/rhythmic context, e.g, the vertical sonority.

Voice decoder The voice decoder takes as input an vector $\mathbf{c}$, and a memory matrix $\tilde{\mathbf{M}}$. It autoregressively generates a sequence of *chord-duration* tuple within a voice. It uses a separate termination predictor at the end of each step. We omit this termination predictor here for simplicity.

At step i, it firstly samples a duration $\mathbf{d}_i$ and then conditions $chord_i$ on the recently sampled $\mathbf{d}_i$:

$$p(chord_i, \mathbf{d}_i | chord_{<i}, \mathbf{d}_{<i}, \mathbf{c}, \tilde{\mathbf{M}}) = p(chord_i | chord_{<i}, \mathbf{d}_{<i}, \mathbf{d}_i, \mathbf{c}, \tilde{\mathbf{M}}) p(\mathbf{d}_i | chord_{<i}, \mathbf{d}_{<i}, \mathbf{c}, \tilde{\mathbf{M}}) \tag{9}$$

The voice decoder uses a single-layer unidirectional GRU. The decoding process can be described as follows. Firstly, the initial state of the GRU is produced by feeding $\mathbf{c}$ into a linear projection layer followed by $\tanh$ activation function. Then at step i, we calculate a query vector using the current GRU state $\mathbf{h}_i$ by

$$\mathbf{q}_i = \text{softmax}(\mathbf{W}_q \mathbf{h}_i), \tag{10}$$

and then retrieve values from the matrix memory:

$$\tilde{\mathbf{v}}_i = \textbf{LayerNorm}(\mathbf{q}_i^T \tilde{\mathbf{M}}). \tag{11}$$

We then feed $\hat{\mathbf{h}}_i = [\mathbf{h}_i, \mathbf{c}, \tilde{\mathbf{v}}_i]$ into a two-layer feedforward network to produce the input to the *duration* decoder which implements the last term on the RHS of the equation 9. After $\mathbf{d}_i$ is sampled, we concatenate the embedding of $\mathbf{d}_i$ with $\hat{\mathbf{h}}_i$ and use another two-layer feedforward network to produce the input vector for the *chord* decoder which implements the first term on the RHS of the equation 9. After $chord_i$ is sampled, we feed the concatenation of embeddings of $chord_i$, $\mathbf{d}_i$, together with $\mathbf{b}_{i+1}$, the within-beat position signal for the next step, into the GRU cell to obtain $\mathbf{h}_{i+1}$. Repeat this until termination.

3.1.4 Measure encoder-decoder

The Measure encoder-decoder is the highest level in the measure object hierarchy.

Measure encoder The measure encoder aggregates information from embedding vectors/matrices of all voices it contains. It then integrates information of the beat unit.

$$\begin{aligned} \mathbf{h}_{measure} &= \mathbf{W}_{measure}[\sum_{voice \in measure} \mathbf{h}_{voice}] + \mathbf{b} \\ \mathbf{M}_{measure} &= [\sum_{voice \in measure} \mathbf{M}_{voice}] + \mathbf{k}\mathbf{v}^T \end{aligned} \tag{12}$$

where $\mathbf{b}$, $\mathbf{k}$, $\mathbf{v}$ are embedding vectors assigned to the corresponding beat unit.

Measure decoder The measure decoder decodes the input vector $\mathbf{c}$ and matrix $\tilde{\mathbf{M}}$ into a beat unit and arbitrary number of voices autoregressively. It firstly sample the beat unit given the input vector $\mathbf{c}$. It then generate each voice one by one using a GRU autoregressive model. The initial state is again calculated by a linear projection followed by $\tanh$ activation function. At each step, the hidden state is fed into a two-feedforward neural network to produce an input to the voice decoder. After a voice is sampled, the voice encoder is invoked to obtain an voice embedding and we feed this embedding back into the GRU to udpate the hidden state. Repeat this until termination.

3.2 Inter-measure context model

After encoding the score into a grid-like representation, a plenty of neural network architectures can fulfill the purpose of an inter-measure context model. The goal is to predict the content at the masked positon in a music score. Our architecture is based on dilated LSTM (Chang et al., 2017) and equivariant layers (Zaheer et al., 2017). For each stacked block, it firstly performs dilated LSTM for each part (staff) independently; then the model attends back into the memory matrix; afterwards, an equivariant layer, which is similar to Zaheer et al. (2017) Yan et al. (2018), is performed for combining information from variable number of parts. We leave details of the concrete architecture used in our experiment to Appendix A.

4 Experiment

In this section, we present further implementation details and our subjective listening test.

4.1 Dataset

We use *musicxml* as the input score format. We use the Mozart Quartets and Corelli Trios obtained from KernScore (`http://kern.humdrum.org`) . We manually corrected all wrongly tied and quantized notes for the Mozart Quartets. We also collected Bach's music offerings, the Art of Fugue, and some movements of Brandenburg Concerto online. Our dataset also includes Bach's 371 chorales including the chorale 150 which is often omitted as it has five parts. In total, our dataset has 893 pieces. We split our dataset into training, validation and testing with ratios of 0.9, 0.05, and 0.05, respectively.

4.2 Implementation details

For the measure autoencoder, we use pitch embedding size 160, duration exponent embedding size 50, duration embedding size 100, chord embedding size 200, measure memory key size 200, value size 60, voice encoding/decoding RNN hidden size 400, measure decoding RNN hidden size 400, measure embedding size 200.

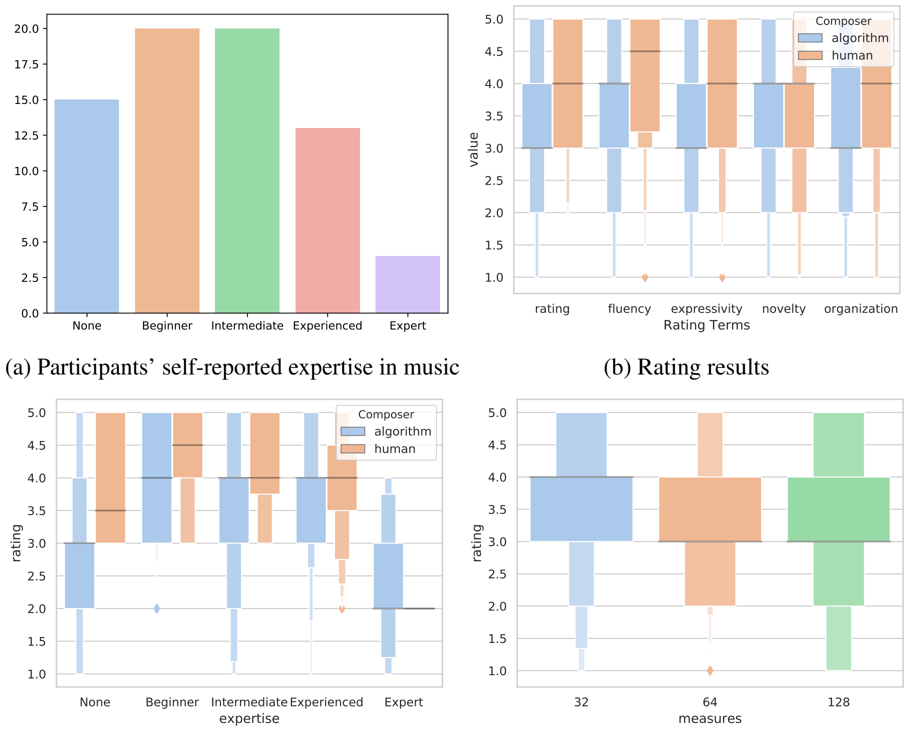
For the inter-measure context model, the dimension of both input and output of each block is the same as the measure embedding size. We use 600 as the expanding projection dimension in the context aggregation layer. We stack 7 blocks with dilations of 1,2,4,8,16,32,64.

During training, we use a dropout rate of 0.1, and each time we sample a mask from *bernoulli(0.15)*. We also apply transposition in the range of $[-5, 5]$ semitones. We also apply distortion to the input score by randomly dropping notes, perturb pitches and beat units within a measure as data augmentation. We averaged the loss per measure for every piece. The final loss is then a average loss of per-measure averaged loss over pieces We train the model with batch size of 16 (pieces) using the

Adam optimization method with learning rate of 2-e4 that is gradually decreased to 1e-5. We use the model with the best validation loss for our experiment. We perform 6 sweeps of resampling on every single position from the configuration obtained by the ancestral sampling (NADE-like) for creating each piece.

4.3 Subjective listening test

For evaluating the quality of generated pieces, we conducted an online survey. We generated string quartets with measure length 32, 64, and 128 respectively, 57 pieces in total, mixed with 15 pieces composed by Corelli, Mozart, Prokofiev, and Satie, where the former two come from our dataset. Each time 3 pieces are presented by picking 1 pieces with the least number of ratings and 2 uniformly. We require participants to rate the overall quality and whether or not they like this piece. The ratings for fluency, expressivity, novelty, and organization are optional. We used star ratings and like button for participants to rate. We received 85 responses from 72 unique participants. The participants' self-reported expertise in music is shown in Fig.4a. Rating results is shown in Fig.4b; also see Appendix B.1 for means and p-values. Fig.4c and Fig.4d demonstrate the overall ratings against participants' expertise and lengths of pieces, respectively. Our algorithm receives a like ratio of 17.7%, while the real human composer receives a like ratio of 40.5%. It is not surprising that the masterpieces of human composers outperform the generated pieces. For our system, participants with Beginner, Intermediate and Experienced expertise rate higher than those with None and Expert expertise. Short pieces receive higher average ratings than longer pieces.



(a) Participants' self-reported expertise in music

(b) Rating results

(c) Overall ratings vs. expertise

(d) Overall ratings vs. length

5 Conclusion and future works

In this paper, we present a system that can generate long polyphonic music. The benefit of modeling long pieces comes from the use of a music notation inspired measure autoencoder. Future works include collecting more clean music scores and also improve the model for more organized overall structure.

REFERENCES

Bar line [Fr. barre de mesure; Ger. Taktstrich; It. stranghetta; Sp. barra de compás]. In Don Michael Randel (ed.), *The Harvard Dictionary of Music*. Harvard University Press, Cambridge, June 2004. ISBN 0-674-01163-5.

Barry C Arnold, Enrique Castillo, Jose Maria Sarabia, et al. Conditionally specified distributions: an introduction (with comments and a rejoinder by the authors). *Statistical Science*, 16(3):249–274, 2001.

Mason Bretan, Gil Weinberg, and Larry Heck. A unit selection methodology for music generation using deep neural networks. *arXiv preprint arXiv:1612.03789*, 2016.

Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 77–87, 2017.

Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation. In *International Conference on Machine Learning*, pp. 1362–1371, 2017.

David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1(Oct):49–75, 2000.

Reimar Hofmann. Inference in markov blanket networks. *Technical University of Munich Technical Report FKI-235-00*, 2000.

Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron C. Courville, and Douglas Eck. Counterpoint by convolution. In *ISMIR*, pp. 211–218, 2017.

Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Douglas Eck. Counterpoint by convolution. *arXiv preprint arXiv:1903.07227*, 2019a.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. In *International Conference on Learning Representations*, 2019b.

Natasha Jaques, Shixiang Gu, Dzmitry Bahdanau, Jose Miguel Hernandez Lobato, Richard E. Turner, and Doug Eck. Tuning recurrent neural networks with reinforcement learning. In *International Conference on Learning Representations (ICLR), workshop track*, 2017.

Adam Roberts, Jesse Engel, and Douglas Eck. Hierarchical variational autoencoders for music. In *NIPS Workshop on Machine Learning for Creativity and Design*, 2017.

K. Stone. *Music Notation in the Twentieth Century: A Practical Guidebook*. W. W. Norton, 1980. ISBN 9780393950533.

Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016.

Stef Van Buuren, Jaap PL Brand, Catharina GM Groothuis-Oudshoorn, and Donald B Rubin. Fully conditional specification in multivariate imputation. *Journal of statistical computation and simulation*, 76(12):1049–1064, 2006.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Christian Walder. Modelling symbolic music: Beyond the piano roll. In *Asian Conference on Machine Learning*, pp. 174–189, 2016.

Yujia Yan, Ethan Lustig, Joseph VanderStel, and Zhiyao Duan. Part-invariant model for music generation and harmonization. In *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, September 23-27, 2018*, pp. 204–210, 2018.

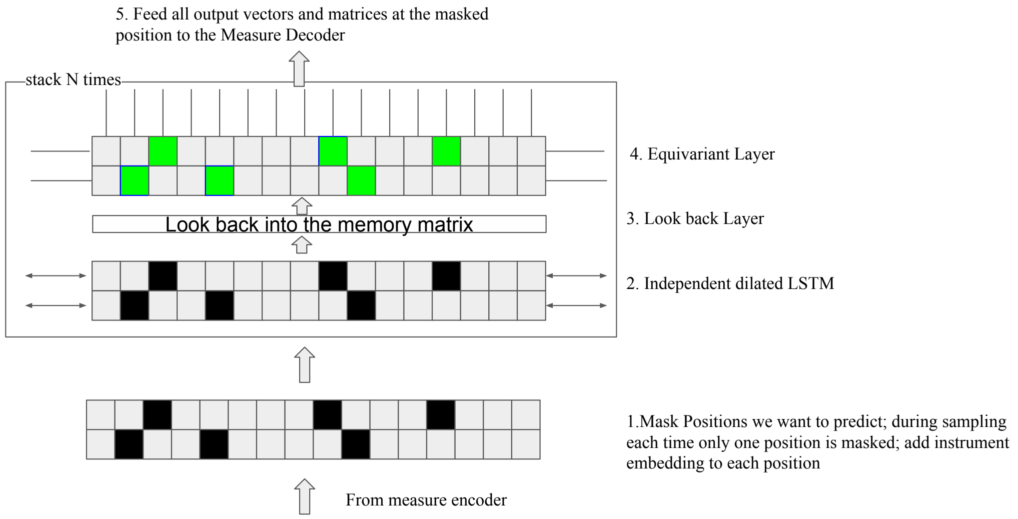

5. Feed all output vectors and matrices at the masked position to the Measure Decoder
stack N times
4. Equivariant Layer
Look back into the memory matrix
3. Look back Layer
2. Independent dilated LSTM
1.Mask Positions we want to predict; during sampling each time only one position is masked; add instrument embedding to each position
From measure encoder

Figure 5: Overview of the inter-measure context model

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pp. 3391–3401, 2017.

A INTER-MEASURE CONTEXT MODEL

Overview The overview of the entire system is illustrated in Fig. 5. Given a music score, we first encode each measure into a fixed-length vector (measure embedding) and a matrix memory, which creates a 3D tensor with dimensions [nMeasure, nStaff, measureEmbedDim], and a 4D tensor with dimensions [nMeasure, nStaff, memKeySize, memValSize], where nMeasure and nStaff are the number of measures (temporal length) and the number of staffs (parts) respectively; memKeySize and memValSize are dimensions of the memory matrix discussed in sec. 3.1.3.

We assume all input and output dimensions for all this layers are the same as the grid of measure embedding vectors. Note that for all the following layers, we apply position-wise *add and layernorm* as used in Vaswani et al. (2017).

Independent dilated LSTM layer We perform (dilated) LSTM on each staff(part) independently. After projecting the cell state of LSTM to the original input dimension, it was added to the input vector and then we apply layernorm. Denote the cell state of LSTM at a postion in the grid as $\mathbf{h}$, the input state vector of the grid at a position as $\mathbf{x}$, then the updated grid state $\hat{\mathbf{x}}$ at a position is given by:

$$\hat{\mathbf{x}} = \textbf{LayerNorm}(\mathbf{x} + \mathbf{W}\mathbf{h}) \tag{13}$$

Look back into the measure memory matrix Denote the current state vector at each measure as $\mathbf{x}$, and the memory matrix for that measure as $\mathbf{M}$, the new state $\mathbf{x}$ is obtained by:

$$\begin{aligned} \mathbf{q} &= \text{softmax}(\mathbf{W}\mathbf{x}) \\ \mathbf{v} &= \textbf{LayerNorm}(\mathbf{q}^T \mathbf{M}) \\ \hat{\mathbf{x}} &= \textbf{LayerNorm}(\mathbf{x} + \text{FNN}_{\text{lookback}}(concat([\mathbf{x}, \mathbf{v}]), dim = 2)) \end{aligned} \tag{14}$$

Context aggregation The context aggregation layer works as follows: denote the input tensor to this layer is $\mathbf{X}$ which has the same shape as the grid of measure embedding vectors, then it firstly applies position-wise projection for each vector on this grid:

$$\mathbf{H} = \mathbf{W}_1 \mathbf{X} \tag{15}$$

Afterwards, it does max reduction along rows and columns of the grid separately. Those pooled result is then concatenated with the input vector and fed into a position-wise two-layer feedforward network with the output dimension the same as the original input vector.

$$\mathbf{Y} = \mathrm{FNN}(\mathrm{concat}([\mathbf{H}, \max(\mathbf{H}, \mathrm{dim} = 0), \max(\mathbf{H}, \mathrm{dim} = 1)], dim = 2)) \tag{16}$$

Here, the column-wise aggregation resembles the context aggregation used in Yan et al. (2018) for coupling simultaneous parts; the row-wise aggregation, additionally, passes some global information to the entire sequence.

Output We send the outputs at the masked position of the last context aggregation layer to the measure decoder. The memory matrix is summed along the axis of staffs (columns) and sent to the measure decoder directly in order to provide a reference for the local vertical sonority when generating events inside a single voice.

B More details on experiments

B.1 Means and p-values of ratings in the subjective evaluation

	Algorithm	Human	p-value
Rating	3.43	4.03	0.014
Fluency	3.56	4.15	0.003
Expressivity	3.39	3.91	0.007
Novelty	3.52	3.50	0.917
Organization	3.44	3.94	0.029

Table 1: Subjective evaluation: mean and p-value