SONG FROM PI: A MUSICALLY PLAUSIBLE NETWORK FOR POP MUSIC GENERATION

Hang Chu, Raquel Urtasun, Sanja Fidler
Department of Computer Science
University of Toronto
Ontario, ON M5S 3G4, Canada
{chuhang1122,urtasun,fidler}@cs.toronto.edu

ABSTRACT

We present a novel framework for generating pop music. Our model is a hierarchical Recurrent Neural Network, where the layers and the structure of the hierarchy encode our prior knowledge about how pop music is composed. In particular, the bottom layers generate the melody, while the higher levels produce the drums and chords. We conduct several human studies that show strong preference of our generated music over that produced by the recent method by Google. We additionally show two applications of our framework: neural dancing and karaoke, as well as neural story singing.

1 INTRODUCTION

Neural networks have revolutionized many fields. They have not only proven to be powerful in performing perception tasks such as image classification and language understanding, but have also shown to be surprisingly good "artists". In Gatys et al. (2015), photos were turned into paintings by exploiting particular drawing styles such as Van Gogh's, Kiros et al. (2015) produced stories about images biased by writing style (e.g., romance books), Karpathy et al. (2016) wrote Shakespeare inspired novels, and Simo-Serra et al. (2015) gave fashion advice.

Music composition is another artistic domain where neural based approaches have been proposed. Early approaches exploiting Recurrent Neural Networks (Bharucha & Todd (1989); Mozer (1996); Chen & Miikkulainen (2001); Eck & Schmidhuber (2002)) date back to the 80's. The main variations between the different models is the representation of the notes and the outputs they produced, which typically encode melody and chord. Most of these approaches were single track, in that they produced only one note per time step. The exception is Boulanger-lewandowski et al. (2012) which generated polyphonic music, i.e., simultaneous independent melodies.

In this paper, we aim to generate pop music, where the melody but also chords and other instruments make up what is typically called a song. We draw inspiration from the Song from π by Macdonald [1], a piano video on Youtube, where the pleasing music is created from a sequence of digits of π. This video shows both the randomness and the regularity of music. On one hand, since any possible digit sequence is a subset of the π digit sequence, this implies that pleasing music can be created even from a totally random base signal. On the other hand, the composer uses specific rules such as *A Harmonic Minor* scale and *harmonies* to convert the digit sequence into a music sheet. It is these rules that play the key role in converting randomness into music.

Following the ideas of Songs from π, we aim to generate both the melody as well as accompanying effects such as chords and drums. Arguably, these turn even a not particularly pleasing melody into a well sounding song. We propose a hierarchical approach, where each level is a Recurrent Neural Network producing a key aspect of the song. The bottom layers generate the melody, while the higher levels produce drums and chords. This enables the drum and chord layers to compensate for the melody in order to produce appleasing music. Adopting the key idea from Songs from π, we condition our model on the scale type allowing the melody generator to learn the notes that are typically played in a particular scale.

[1] https://youtu.be/OMq9he-5HUU

We train our model on 100 hours of midi music containing user-composed pop songs and video game music. We conduct human studies with music generated with our approach and compare it against a recent approach by Google, showing that our songs are strongly preferred over the baseline. In our human study we also perform an ablation analysis of our model. We additionally show two new applications: neural dancing and karaoke as well as neural music singing. As part of the first application we generate a stickman dancing to our music and lyrics that can be sung with, while in the second application we condition on the output of Kiros et al. (2015) which writes a story about an image and convert it into a pop song. We refer the reader to http://www.cs.toronto.edu/songfrompi/ for our demos and results.

2 Related Work

Generating music has been an active research area for decades. It brings together machines learning researchers that aim to capture the complex structure of music (Eck & Schmidhuber (2002); Boulanger-lewandowski et al. (2012)), as well as music professionals (Chan et al. (2006)) and enthusiasts (Johnson; Sun) that want to see how far a computer can get to be a real composer. Real-time music generation is also explored for gaming (Engels et al. (2015)).

Early approaches mostly instilled knowledge from music theory into generation, by using rules of how music segments can be stitched together in a plausible way, e.g., Chan et al. (2006). On the other hand, neural networks have been used for music generation since the 80's (Bharucha & Todd (1989); Mozer (1996); Chen & Miikkulainen (2001); Eck & Schmidhuber (2002)). Mozer (1996) used a Recurrent Neural Network that produced pitch, duration and chord at each time step. Unlike most other neural network approaches, this work encodes music knowledge into the representation. Eck & Schmidhuber (2002) was first to use LSTMs to generate both melody and chord. Compared to Mozer (1996), the LSTM captured more global music structure across the song.

Like us, Kang et al. (2012) built upon the randomness of melody by trying to accompany it with drums. However, in their model the scale type is enforced. No details about the model are given, and thus it is virtually impossible to compare to. Boulanger-lewandowski et al. (2012) propose to learn complex polyphonic musical structure which has multiple notes playing in parallel through the song. The model is single-track in that it only produces melody, whereas in our work we aim to produce multi-track songs. Just recently, Huang & Wu (2016) proposed a 2-layer LSTM that, like Boulanger-lewandowski et al. (2012), produces music that is more complex than a single note sequence, and is able to produce chords. The main novelty of our work over existing approaches is a hierarchical model that incorporates knowledge from music theory to build the neural architecture, and produces multi-track pop music (melody, chord, drum). We also present two novel fun applications.

3 Concepts from Music Theory

We start by introducing the basic notation and definitions from music theory. A **note** defines the basic unit that music is composed of. Music follows the **12-tone** system, i.e., 12 is the cycle length of all notes. The 12 tones are: C, $C^\sharp/D^\flat$, D, $D^\sharp/E^\flat$, E, F, $F^\sharp/G^\flat$, G, $G^\sharp/A^\flat$, A, $A^\sharp/B^\flat$, B. A **bar** is a short segment of time that corresponds to a specific number of beats (notes). The boundaries of the bar are indicated by vertical bar lines.

Scale is a subset of notes. There are four types of scales most commonly used: *Major* (*Minor*), *Harmonic Minor*, *Melodic Minor* and *Blues*. Each scale type specifies a sequence of relative *intervals* (or shifts) which act relative to the starting note. For example, the sequence for the scale type *Major* is $2 \rightarrow 2 \rightarrow 1 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 1$. Thus, *C Major* specifies the starting note to be C, and applying the relative sequence of shifts yields: $C \xrightarrow{2} D \xrightarrow{2} E \xrightarrow{1} F \xrightarrow{2} G \xrightarrow{2} A \xrightarrow{2} B \xrightarrow{1} C$. The subset of notes specified by *C Major* is thus C, D, E, F, G, A, and B (a subset of seven notes). All scales types have a subset of seven notes except for Blues which has six. In total we have 48 unique scales, i.e. 4 scale types and 12 possible starting notes. We treat *Major* and *Minor* as one type as for a *Major* scale there is always a *Minor* that has exactly the same set of notes. In music theory, this is referred to as *Relative Minor*.

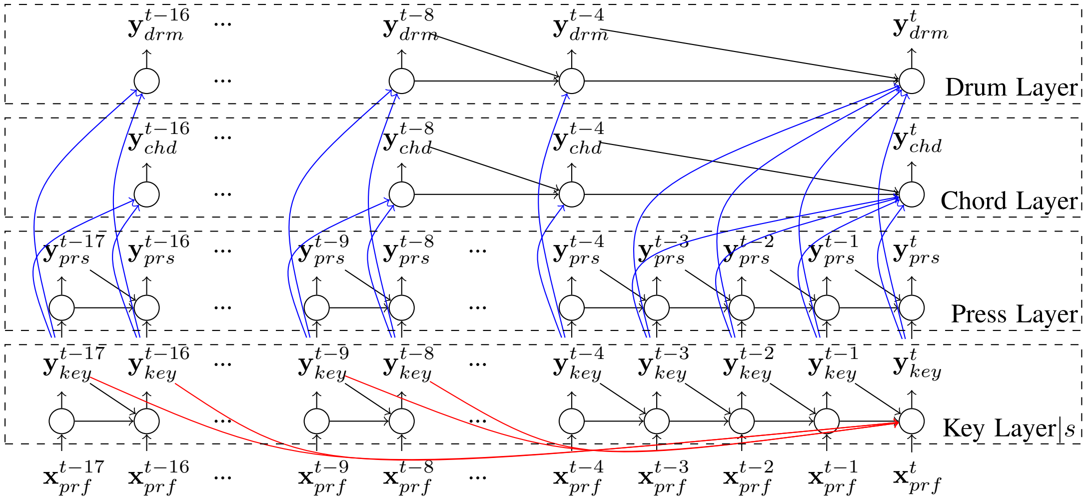

Figure 1: Overview of our framework. Only skip connections for the current time step t are plotted.

Chord is a group of notes that sound good together. Similarly to scale, a chord has a start note and a type defining a set of intervals. There are mainly 6 types in triads chords: *Major Chord*, *Minor Chord*, *Augmented Chord*, *Diminished Chord*, *Suspended 2nd Chord*, and *Suspended 4th Chord*.

The Circle of Fifths is often used to produce a chord progression. It maps 12 chord starting notes to a circle. When changing from one chord to another chord, moving to a nearby chord on the circle is often preferred as this forms a *strong chord progression* that produces the sense of harmony.

4 Hierarchical Recurrent Networks for Pop Music Generation

We follow the high level idea behind the Song from π to define our model. In particular, we generate music with a hierarchical Recurrent Neural Network where the layers and the structure of the hierarchy encode our prior knowledge about how pop music is composed. We first outline the model and describe the details and justifications for our choices in the subsections that follow.

We condition our generation on the scale type, as this helps the model to pick up the regularities in pop songs. We encode melody with two random variables at each time step, representing which key is being played (the *key layer*) and the duration that the key will be pressed (the *press layer*). The melody is generated conditioned on the scale, which does not vary across the song as is typically the case in pop music. We assume the drums and the chords are independent given the melody. Thus conditioned on the melody, at each time step we generate the chord (the *chord layer*) as well as the drums (the *drum layer*). The output at all layers yields the final song. We refer the reader to Fig. 1 for an illustration of our hierarchical model.

4.1 The role of Scale

It is known from music theory that while in principle each song has 12 tones to choose from, most of the notes are in fact only using the six (for Blues) or seven (for other scales) tone subsets specified by the scale rule. We found that by conditioning the music generator on scale it captures these regularities more easily. However, we do not enforce the notes to be generated from the subset and allow our model to generate notes outside the scale.

We confirm the above musical fact by analysing over 100 hours of pop song music from the midi_man dataset. Since scale is defined relative to a starting note, we first try to factor out its influence and normalize all songs to have identical start note. To identify the scale of a song, we compute the histogram over the 12 tones and match it with the 48 tone subsets of 4 scale types with 12 different start notes. We then normalize all songs to have start note C by applying a constant shift on all notes. This allows us to categorize any song into 4 scale types. Since this shift affects all notes at once, it does not affect how the song sounds (its harmony). Our analysis shows that for all notes in all *Major* scale songs, 94.66% are within the tone subset. For *Harmonic Minor*, *Melodic Minor*,

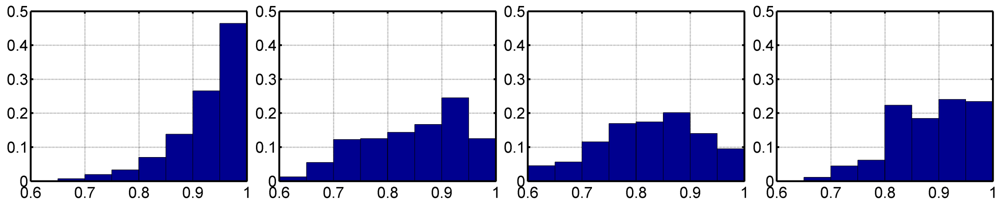

Figure 2: Distribution of within-scale note ratio for four scale types. x-axis: percentage of tones within the scale type's tone set, y-axis: percentage of songs of the scale type. (a)-(d) shows *Major*(*Minor*), *Harmonic Minor*, *Melodic Minor*, and *Blues*, respectively.

and *Blues* the percentage of notes that belong to the main tone set is 87.16%, 85.11%, and 90.93%, respectively. We refer the reader to Fig. 2, where the x-axis denotes the percentage of within-scale notes of a song, and the y-axis indicates how many songs in the dataset have that percentage. Note that the majority of the notes follow the scale rule. Furthermore, different scale types have different inlier distribution. We thus represent scale with a single random variable $s \in \{1, \cdots, 4\}$ which is fixed for the whole song, and condition the model on it. [2]

4.2 Two-layer RNN for Melody Generation

We represent the melody with two random variables per time step: which key is pressed, and the duration of the press. We use a RNN to generate the keys conditioned on the scale. Then conditioned on the output of the key layer, a second RNN generates the duration of the press at each time step.

In particular, we model the key layer with a two-layer LSTM (Hochreiter & Schmidhuber (1997)) with a 512-dimensional hidden state, which outputs a note (key) at each time step. Note that we condition on scale s, thus we have a different set of weights for each scale. We only allow notes between $C3$ to $C6$ as notes outside this range are usually too low or too high to sound good. We remind the reader that given a scale, seven (or six for blues) out of the twelve notes (per octave) are statistically more plausible, however we allow the model to choose from all 12. This results in a 37-dimensional output, as there are 36 possible notes corresponding to 3 octaves with 12 notes per octave, plus silence. Let h^t_{key} be the hidden state of the second key decoder layer at time t. We compute the probability of each key using the softmax:

$$P(\mathbf{y}^t_{key}) \propto \exp(\mathbf{v}_{\mathbf{y}^t_{key}} h^t_{key}) \tag{1}$$

where $\mathbf{v}_{\mathbf{y}^t_{key}}$ is the row of $\mathbf{V}$ (the output embedding matrix of notes), corresponding to note $\mathbf{y}^t_{key}$.

As input to the LSTM we use a vector that concatenates multiple features: a one-hot encoding of the previous generated note $\mathbf{y}^{t-1}_{key}$, Lookback features, and the melody profile. The Lookback features were proposed by Google Magenta (Waite et al.) to make it easier for the model to memorize recently produced notes and potentially repeat them. They include skip connections from two and one bar ago (a bar is 8 consecutively played notes), i.e., $\mathbf{y}^{t-16}_{key}$ and $\mathbf{y}^{t-8}_{key}$. They also contain two additional features, indicating whether the last generated key has been copied from one or two bars ago, i.e. $\mathbb{1}(\mathbf{y}^{t-1}_{key}, \mathbf{y}^{t-1-8}_{key})$ and $\mathbb{1}(\mathbf{y}^{t-1}_{key}, \mathbf{y}^{t-1-16}_{key})$. They also add a 5-dimensional feature indicating a binary encoding of the current time t. This helps the model keep track where in a $4-$bar range it is, and thus produce music accordingly.

In addition, we introduce a new feature which we refer to as the *melody profile*. Intuitively, the profile represents the high-level music flow. To get the profile for each song, we compute the local note histogram at each time step with width of two bars, and cluster all local histograms within the song into 10 clusters via k-means. We order the 10 clusters with mean note ordered from low to high as cluster 1 to 10, and apply moving averages on the cluster id sequence to encourage local smoothness. This results in a 10-dimensional one-hot vector representation of the cluster id for each time step. This additional information allows the user to set the melody's ups and downs of the song.

The keys alone are not sufficient to describe how the melody is performed. Additionally we also need to know the duration that each key needs to be pressed for. Towards this goal, conditioned on the

[2]For readers with musical background, the *Twelve-Tone Serialism* technique Schoenberg & Newlin (1951) prevents emphasis of any one tone. However, our data analysis indicates that pop music is not influenced by it.

(a) (b) (c) (d)

Figure 3: Co-occurrence of tones in melody (y-axis) and chord (x-axis). (a)-(d) shows *Major*(*Minor*), *Harmonic Minor*, *Melodic Minor*, and *Blues*, respectively.

melody, we generate the duration of each key with a two-layer LSTM with a 512-dimensional hidden state. We represent the duration of pressing as a forward counting sequence that is conditioned on the generated melody. The press outputs 1 when a new key is pressed, and sequentially outputs 2, 3, 4 and so on as the key is held on. When the current key is released, the press counter is reset to 1. Compared to the event on-off representation of Waite et al., our representation learns the melody flow and how to press separately. This is important, as Waite et al. has extremely unbalanced output distributions dominated by the repeat-of-holding event. We represent press $\mathbf{y}^t_{prs}$ as a 8-dimensional one-hot vector. The input to our LSTM is $\mathbf{y}^{t-1}_{prs}$, concatenated with the 37-dimensional one-hot encoding of the melody key $\mathbf{y}^t_{key}$.

4.3 Chord and Drum RNN Layers

We studied all existing chords in our 100 hours of pop music. Although in principle a chord can be any arbitrary combination of multiple notes, we observed that in the actual music data 99.19% of the chords belong to one of 72 chord classes (6 types $\times$ 12 start notes). Fig. 3 shows the correlation between the melody's tone and the starting note of the chord playing at the same time. It can be seen that chord is strongly correlated with melody. These two findings inspire our design. We thus represent chord $\mathbf{y}^t_{chd}$ as a one-hot encoding with 72 classes, and predict it using a two-layer LSTM with a 512-dimensional hidden state. We generate one chord at each time step. The input is $\mathbf{y}^{t-4}_{chd}$ concatenated with $\mathbf{y}^{t-3:t}_{key}$.

We look at our music dataset and find all unique drum patterns with duration of a half bar. We then compute the histogram of all the patterns. This forms a long tail distribution, where 94.60% comes from the top 100 common patterns. We generate drum conditioned on the key layer using a two-layer LSTM with 512 dimensional hidden states. Drum $\mathbf{y}^t_{drm}$ is represented as one-hot encoding with of the 100 unique one-bar-long drum patterns. The input is $\mathbf{y}^{t-4}_{drm}$ concatenated with the notes from the previous three times steps $\mathbf{y}^{t-3:t}_{key}$.

4.4 Learning

We use cross-entropy as our loss function to train each layer. We follow the typical training strategy where we make predictions at each layer and time step but feed in ground-truth information to the next. This effectively decomposes training, and allows to train all layers in parallel. We use the Adam optimizer, a learning rate of 2e-3 and a learning rate decay of 0.99 after each epoch for 10 epochs.

4.5 Music Synthesis: Putting all the Outputs Together

To synthesize music we first randomly choose a scale and a profile $\mathbf{x}_{prf}$. For generating $\mathbf{x}_{prf}$, we randomly choose one cluster id with a random duration, and repeat until we get the desired total length of the music sequence. We then perform inference in our model conditioned on the chosen scale, and use $\mathbf{x}_{prf}$ as input to our key layer. At each time step, we sample a key according to $P(\mathbf{y}^t_{key})$. We encode it as a one-hot vector and pass to the press, chord and drum layers. We sample the press, chords and drums at each time step in a similar fashion.

Figure 4: Example of our music generation. From top to bottom: melody, chord and drum respectively.

Before putting the outputs across layers together, we further adjust the generated sequences at the bar level. For melody, we first check at each bar if the first step is a continuation of a previous note or silence. If it is the latter, we find the first newly pressed note within the bar and move it to the beginning of the bar. We do similarly for the windows of two half-bars as well as the four quarter-bars. This makes the melody more likely to be on the beat, and generally sounds better. We verify this in our experiments.

For chord, we generate one chord at each half bar, which is the majority of all single step chord generations. Furthermore, we incorporate the rule of chord progression in the *Circle of Fifths* as between chords pairwise smooth terms, and compute the final chord using dynamic programming. For drum, we generate one pattern at each half bar.

Our model generates with scale starting note C, and then applies a constant shift to generate music with other starting notes. Besides scale, which instrument to use is also customizable. However, we simply set all instruments as grand piano in all experiments, as the effect and musical meaning of different instrument combinations is beyond the scope of this paper.

5 EXPERIMENTS

To train our model, we took 100 hours of pop music from midi_man which consists of user-composed pop songs and video game music. In our generation, we always use 120 beats per minute with 4 time steps per beat. However, songs in the dataset can have arbitrary speed. To neutralize the effect of this, we detect the most frequent interval between two adjacent notes for each song, and iteratively divide or multiply this interval by 2 until it falls in the range between $0.25s$ and $0.5s$. We use this as a measure of the song's beat duration. We then adjust the song's temporal axis so that all songs have the same beat duration of $0.5s$.

A MIDI file can be separated into different channels/tracks, where the 9th channel is specifically preserved for drums. We categorize the rest of non-drum tracks into melody, chord, and else, by simply setting thresholds on average number of unique notes within a bar and average number of note changing within a bar, as chords are by definition repetitive. Fig. 4 shows an example of our music generation.

To evaluate the quality of our music generation, we conduct a human survey with 27 participants. All subjects are university students who did not have any prior knowledge about the content of our project. In the survey, participants are presented with several pairs of 30-second music clips, and are asked to vote which clip in the pair sounds better. We gave no other information about what they are listening to. They are also allow to submit a neutral vote in case they cannot decide between the two choices. In our study, we consider three cases: our full method versus Magenta Waite et al., our method with melody only versus Google Magenta (Waite et al.), and our method versus our method without the temporal alignment described in Sec.4.5. We randomly generated 10 songs per method and randomly shuffled each pair. For the Magenta baseline we used its Lookback version, which was the latest version at the time of our submission.

As shown in Table 1, most participants prefer songs produced by our method compared to Magenta. Participants also made comments such as *music sounds better with percussion than piano alone*, and *multiple instruments with continuous play is much better*. This confirms that our multi-layer generation improves music quality. Few participants also point out that *drums sound too different and do not participate to the melody perfectly*, which indicates that further improvements can be still made. In the second comparison, we study if the quality improvement of our method is only caused

Method	Ours	Magenta	Ours-MO	Magenta	Ours	Ours-NA
% of votes	81.6%	14.4%	69.6%	13.6%	75.2%	12.0%

Table 1: Human evaluation of music generated by different methods: ours and Waite et al.'s Magenta. Ours-MO and Ours-NA are short for Ours Melody Only and Ours No Alignment. We allowed neutral votes, thus the sum of the pair is less than 100%.

	Human	Magenta	Ours
sub-seq	7.06	4.39	4.65
repeat	4.04	17.08	2.33

Table 2: Evaluations of the longest matching sub-sequence with training, and self repeating times.

by adding chords and drums, or is also related to our two-layer melody generation with alignment. It can be seen that without chords and drums, the score drops as expected, but is still much higher than the Magenta baseline. This is because our method *produces less recursion and silence*, and *faster and more accurate tempo* as mentioned by the participants. In the last comparison, most participants prefer our full method than the no-alignment version, since *beats are more subtle and better timed*. This proves the usefulness of temporal alignment. We performed significance tests on the evaluation results in Table 1. All comparisons passed the significance test with significance level 5%. The lowest alpha values to reject the null hypothesis are 1e-19, 1e-14, and 1e-19, respectively. Further experimental results of removing music scale in our method and adding temporal alignment to the baseline can be found on our project page.

To examine the suitability of the four scale types we use, we collected the list of all existing musical scales from Wikipedia and measured the scale distribution of the dataset. 37.8% of the data belongs to our four scales, 47.7% belongs to Acoustic, Algerion, Lydian, Adonai Malakh, and Ukrainian, while 14.5% belongs to the rest 31 uncommonly seen scales such as Insen, Iwato, Yo, and Enigmatic. We also found out that the five scales that accounts for 47.7% are either one or two degree away from one of our used scales (all notes are the same except one being one or two steps away). This experiment shows that even in the most rigorous musical setting, at least 85.5% of online songs are very close to the four scales that we use.

Finally we study our model's capabilities to generate new music. Towards this goal, we generated 100 sequences of 50 seconds of length using different random initializations. We perform two evaluations. First, for each sequence, we search for the longest sub-sequence of keys that matches part of the training data, and record its length. This evaluates how much the model copies the training data. Secondly, we break each generated melody into segments of 2-bars in length (inspired by common definition of music plagiarism). We then compare each segment to all segments in the rest of the 100 generated songs, and record the repeat time. This evaluates how much the model repeats itself. For comparison, we repeat the same evaluation for the Magenta baseline, and human composed music. Table 2 reports the results. It can be seen that our method performs similarly as Magenta in terms of copying (*sub-seq*). It is somewhat surprising that human composers in fact tend to copy more from other songs, which indicates that both generation approaches can be further relaxed in terms copying. Our method is less likely to generate recurring melodies (*repeat*) compared to Magenta, and is closer to the statistics of human-produced songs.

6 Applications

In this section we demonstrate two novel applications of our pop music generation framework. We refer the reader to http://www.cs.toronto.edu/songfrompi/ for the music videos.

6.1 Neural Dancing and Karaoke

In our first application, we attempt to generate both music and a stickman dancing to it, as well as a sequence of karaoke-like text that people can sing along with. To learn the relationship between music and dance, we download 1 hour of video from the game *Just Dance*, as well as the MIDI files for songs included in the video from different sources. We use the method in Newell et al. (2016) to track single-frame 2D human pose in the videos. We process the single-frame tracking result to ensure left-right body consistency through time, and then use the method of Zhou et al. (2016) to convert the 2D pose sequence into 3D. Example results are shown in Fig. 5. We observe that our pose processing pipeline is able to extract reasonable human poses most of the time. However, the

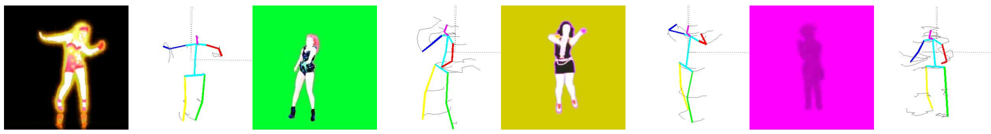

Figure 5: Examples from *Just Dance* and 3D human pose tracking result. (a) and (b) are success cases, pose tracking fails in (c), and (d) shows the defect in video which makes tracking difficult.

quality is not perfect due to tracking failure or video effects. We define pose similarity as average euclidean distance of all joints, and cluster poses into 456 clusters. We used Frey & Dueck (2007) as the number of clusters is large.

We learn to generate a stickman dancing by adding another dancing layer on top of the key layer, just like for drum and chord. We generate one pose at each beat, which is equivalent to 4 time steps or 0.5 seconds in a 120 beat-per-minute music. In particular, we predict one of the 456 pose clusters using a linear projection layer followed by softmax. We use cross-entropy at each time step as our loss function. At inference time, we further apply moving average to temporally smooth the generated 3D pose sequence.

To learn the relationship between music and lyrics, we collect 51 hours of lyrics data from the internet. This data contains 50 hours of text without music, and the rest 1 hour are songs we collected from *Just Dance*. For the music part, we temporally align each sentence in the lyrics with the midi music by using the widely-existing *lrc* format, which records the time tag at the beginning of every sentence. We select words that appear at least 4 times, which yields a vocabulary size of 3390 including unknown and end-of-sentence. Just as for dance, we generate one word per beat using another lyrics layer on top of the key layer.

6.2 Neural Story Singing

In this application our aim is to sing a song about a photo. We first generate a story about the photo with the neural storyteller Kiros et al. (2015) and try to accompany the generated text with music. We utilize the same 1 hour dataset of temporally aligned lyrics and music. We further include the phoneme list of our 3390 vocabulary as we also want to sing the story. Starting from the text produced by neural storyteller, we arrange it into a temporal sequence with 1 beat per word and a short pause for end-of-sentence, where the pause length is decided such that the next sentence starts from a new bar. As our dataset is relatively small, we generate the profile conditioned on the text, which has less dimensions compared to the key. This is done by a 2-layer LSTM that takes as input the generated profile at the last time step concatenated with a one-hot vector of the current word, and outputs the current profile. We then generate the song with our model given the generated profile. The generated melody key is then used to decide on the pitch frequency of a virtual singer, assuming the key-to-pitch correspondence of a grand piano. We further constrain that the singer's final pitch is always in the range of $E3$ to $G4$, which we empirically found to be the natural pitch range. We then replace all words outside the vocabulary with the sound *Ooh*, and play the rendered singing with the generated music.

7 Conclusion and Future Work

We have presented a hierarchical approach to pop song generation which exploits music theory in the model design. In contrast to past work, our approach is able to generate multi-track music. Our human studies shows the strength of our framework compared to an existing strong baseline. We additionally proposed two new applications: neural dancing & karaoke, and neural story singing.

In this paper, we show that incorporating knowledge from the music theory into the model, as well as capturing multiple aspects of music results in better sounding songs. However, generating appealing and interesting music that captures structure, rhythm, and mood is challenging, and there is an exciting road ahead to improve on these aspects in the future.

REFERENCES

Jamshed J. Bharucha and Peter M. Todd. Modeling the perception of tonal structure with neural nets. *Computer Music Journal*, 13(4):44–53, 1989.

Nicolas Boulanger-lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *ICML*, 2012.

Michael Chan, John Potter, and Emery Schubert. Improving algorithmic music composition with machine learning. In *9th International Conference on Music Perception and Cognition*, 2006.

Chun-Chi J. Chen and Risto Miikkulainen. Creating melodies with evolving recurrent neural networks. In *International Joint Conference on Neural Networks*, 2001.

Douglas Eck and Juergen Schmidhuber. A first look at music composition using lstm recurrent neural networks. 2002.

Steve Engels, Fabian Chan, and Tiffany Tong. Automatic real-time music generation for games. In *AIIDE Workshop*, 2015.

Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. volume 315, pp. 972–976, 2007.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. In *arXiv:1508.06576*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Allen Huang and Raymond Wu. Deep learning for music. *arXiv preprint arXiv:1606.04930*, 2016.

Daniel Johnson. Composing music with recurrent neural networks. `https://goo.gl/YP9QyR`.

Semin Kang, Soo-Yol Ok, and Young-Min Kang. *Automatic Music Generation and Machine Learning Based Evaluation*, pp. 436–443. Springer Berlin Heidelberg, 2012.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. In *ICLR 2016 Workshop*, 2016.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *NIPS*, 2015.

David Macdonald. Song from π. `https://youtu.be/OMq9he-5HUU`.

Reddit midi_man. Midi collection. `https://goo.gl/4moEZ3`.

Michael C. Mozer. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2-3), 1996.

Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, 2016.

Arnold Schoenberg and Dika Newlin. Style and idea. Technical report, Williams and Norgate London, 1951.

Edgar Simo-Serra, Sanja Fidler, Francesc Moreno-Noguer, and Raquel Urtasun. Neuroaesthetics in fashion: Modeling the perception of beauty. In *CVPR*, 2015.

Felix Sun. Deephear - composing and harmonizing music with neural networks. `https://goo.gl/7OTZZL`.

Elliot Waite, Douglas Eck, Adam Roberts, and Dan Abolafia. Project magenta. `https://magenta.tensorflow.org/`.

Wikipedia. List of musical scales and modes. `https://goo.gl/5kvXLP`.

Xiaowei Zhou, Menglong Zhu, Spyridon Leonardos, Kosta Derpanis, and Kostas Daniilidis. Sparseness meets deepness: 3d human pose estimation from monocular video. In *CVPR*, 2016.