이학석사 학위논문

HymnTron: A DNN Based Automatic Music Composer

(깊은 신경망 기반 자동 음악 작곡)

2019년 2월

서울대학교 대학원

수 리 과 학 부

박 진 만

HymnTron: A DNN Based Automatic Music Composer

(깊은 신경망 기반 자동 음악 작곡)

지도교수 강 명 주

이 논문을 이학석사 학위논문으로 제출함

2018년 10월

서울대학교 대학원

수 리 과 학 부

박 진 만

박 진 만의 이학석사 학위논문을 인준함

2018년 12월

위 원 장	______________________	(인)
부 위 원 장	______________________	(인)
위 원	______________________	(인)

HymnTron: A DNN Based Automatic Music Composer

by

Jinman Park

A DISSERTATION

**Submitted to the faculty of the Graduate School
in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Mathematical Sciences
Seoul National University
February 2019**

Abstract

HymnTron: A DNN Based Automatic Music Composer

Jinman Park

Department of Mathematical Sciences
The Graduate School
Seoul National University

With recent developments, there are many different applications for the use of Deep Neural Networks. One of which is music composition. In this thesis, we attempt to compose music that is indistinguishable from human composition. We initialize the composition with a seed note. Next, we utilize the long-short term memory network to compose numerous melodies from the given seed note and the chord progression. Finally, we use a variety of scoring functions to keep only a desired number of compositions through the beam search algorithm. Human survey is conducted for quality testing.

Key words: Music composition, Beam search, Deep learning
Student Number: 2016-22071

Contents

Chapter 1

Introduction

In 1950, Professor Alan Turing developed the reputable Turing test. It is to test a machine's ability to show intelligent behavior indistinguishable from a human. His work was proven to be both highly influential and widely criticized in the field of Artificial Intelligence. He believed that machines can think. The goal of this paper is to follow the footsteps of Professor Turing, and compose music using Deep Neural Networks that cannot be distinguished from human composition. However, it is not our goal to rob the artist's job of composing music. It is to push the boundaries of Artificial Intelligence where pattern does not exist as clearly as others, and possibly aid the artists with composing music.

Although the history of automatic music composition using Deep Recurrent Neural Networks [1] [3] [4] does not stretch far, there have been other attempts, such as Google's Magenta [6] and a master's dissertation, Bachbot [7]. The basic framework behind these two works and this paper is similar in that it quantizes the musical data into time series form. The differences lie within the processing methods.

Music has a way of moving people's emotions. It sets the mood for one to be immersed in. However, With this great power comes such complexity for one to work with that the genre must be narrowed down.

In this thesis, we aim to generate hymn music melody. The music data is in the form of Musical Instrument Digital Interface, hereafter called "MIDI". It contains important information about a music piece such as key signature, time signature, note pitches and BPM. The MIDI data is converted into a quantized sequence and trained based on a carefully structured model.

A common problem with mixing artificial intelligence and art is that it can infringe on copyright. Since the model learns a specific style of music through the training set, one can wonder if it will create music that is too similar to the original songs. However, in this paper, a variety of scoring functions will be used to ensure that the compositions display a similar compositional distribution while being different from the original songs.

One of the main difficulties of evaluating music is that it cannot be numerically scored. Judging whether a song is good or bad is very subjective. Although one can score it by using the chain of conditional probability vectors feeding out of the softmax layer, it does not necessarily correlate well with human evaluation. Therefore, human survey will be conducted to imitate the Turing test to judge if the neural network can compose music indistinguishable from a human's composition.

Chapter 2

Data Processing

2.1 Transformation

Key signature is a set of sharps and flats that designates notes to be played higher or lower than the corresponding natural notes until the end of the piece or until another key signature appears. A song can be played in different keys while sounding similar because the interval between each note in a musical scale is identical regardless of the starting note. Therefore, to simplify the encoding process, the notes and chord progressions of each song must be transformed to the same key. Amongst the two, note sequence transformations are relatively straightforward. It can be defined as follows:

Definition 2.1. *Let* $\mathbf{K}$ *be a numeric representation of the current key signature so that* $\mathbf{K} \in \{0, 1, ..., 11\}$*, where C major is 0. Then the relative key difference between key X, and root key R can be defined as*

$$\triangle \mathbf{K}_{X,R} = \mathbf{K}_R - \mathbf{K}_X. \tag{2.1}$$

Definition 2.2. *Let* $\mathbf{N}_i$ *be a MIDI note pitch, where* $\mathbf{N}_i \in \{0, 1, ..., 127\}$*. Then, for some note sequence length l, the transformed note sequence from key X to root key R becomes*

$$\mathbf{NS}_{trans} = \{\mathbf{N}_1 + \triangle\mathbf{K}_{X,R}, \mathbf{N}_2 + \triangle\mathbf{K}_{X,R}, ..., \mathbf{N}_l + \triangle\mathbf{K}_{X,R}\}. \tag{2.2}$$

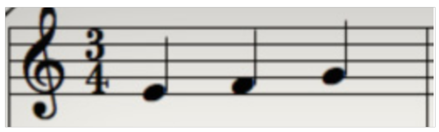
To transform a certain chord progression, one must understand the role of chord progressions in a music piece. Chord progressions are the base feature on which melody and rhythm are built. It restrains the melody from wandering around and encourages the melody to follow a set of given notes.

Definition 2.3. *Let $\mathbf{C}_i$ be a chord information expressed as a string data, and let $\mathbf{S} : \mathbf{C}_i \rightarrow \mathbf{C}_{num,i}$ be a bijective string decoder function. Then, for some note sequence length l, the transformed chord progression from key X to root key R can be defined as*

$$\mathbf{CP}_{trans} = \{\mathbf{S}^{-1}(\mathbf{S}(\mathbf{C}_1)+\triangle\mathbf{K}_{X,R}), \mathbf{S}^{-1}(\mathbf{S}(\mathbf{C}_2)+\triangle\mathbf{K}_{X,R}), ..., \mathbf{S}^{-1}(\mathbf{S}(\mathbf{C}_l)+\triangle\mathbf{K}_{X,R})\}. \tag{2.3}$$

2.2 Quantization

A melody track contains events with time information. It proceeds in a two step process repeating note on and note off. The time stamp on the "note on" events represent a jump. It indicates when to play the next note relative to the previous event. Similarly, note off events instruct when to stop playing the previous playing note. To convert these events into a time series sequence, the cumulative time stamps are quantized into desired steps.

Music Score	

MIDI	Message('note_on', note=64, velocity=80, time=0)
	Message('note_on', note=64, velocity=0, time=119)
	Message('note_on', note=65, velocity=80, time=1)
	Message('note_on', note=65, velocity=0, time=119)
	Message('note_on', note=67, velocity=80, time=1)
	Message('note_on', note=67, velocity=0, time=119)

Quantized (4 steps per quarter)	[64, -2, -2, -2, 65, -2, -2, -2, 67, -2, -2, -2]

Figure 2.1: Sample Quantization.

Unless the quantization steps are very small, it poses a problem. For example, if the time stamps are quantized into 1/16 notes, some 1/32 notes and triplet notes are bound to be missed. However, quantizing the time stamps into a very fine interval brings up a more severe problem. The data becomes so sparse, mostly with no events, that it becomes very difficult for the network to train.

2.3 Refinement

A standard MIDI note pitch ranges from 0 to 127, inclusive, with 60 representing the middle 'C' note on the keyboard. If the note range is too wide, it becomes too complex and sparse for the neural network to learn patterns. As depicted in figure 2.2, the note pitches of the dataset ranges about two octaves. Naturally, pitches 48 to 84, inclusive, were set to be the minimum and maximum range for melodic note generation in this paper.

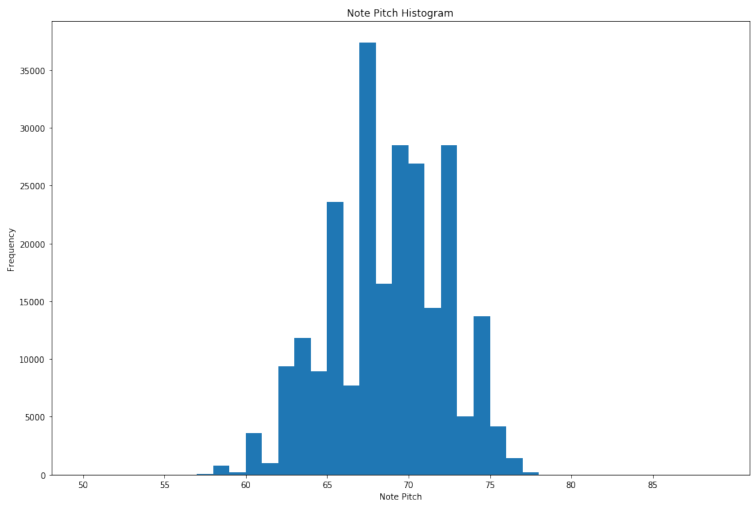


Figure 2.2: Note Pitch Histogram of Hymn Songs.

Long-short term memory networks, commonly known as "LSTM", are an important tool when modeling an architecture designed to train on time series data. They feed on the information from the past to predict future steps. However, it shows weakness when the time steps exceed a certain point.

Therefore, if the quantized note sequence length reaches beyond a certain threshold, it needs to be cropped into smaller sequences. On the other hand, if the melody is too short, it is considered to be insignificant compared to a musical phrase. Finally, a melody is considered to be whole until there is at least a bar of rest with a new melody starting.

2.4 Encoding

Transforming musical data into encoded information to be trained through a neural network can become a creative process. To put it simply, there is no right or wrong way to do it. It goes without saying that what effects a musical composition varies from person to person, and genre to genre. In order to understand the nature of hymn music, consulting with music professionals is utterly necessary.

Although the encoding process can be creative, some aspects are fundamental. Previous note information along with chord information determines the octave and mood of the piece. It restrains the predictions from being too random. Other minor details such as an indicator for the duration of hold or an indicator for a new bar are used for encoding.

Definition 2.4. *Let $\mathbf{E}_i$ be an encoded note. It is defined as*

$$\mathbf{E}_i = append(\mathbf{N}_{i,enc}, \mathbf{C}_{i,enc}, \mathbf{I}_{i,bar}, \mathbf{I}_{i,hold}), \tag{2.4}$$

where $\mathbf{N}_{i,enc}$, $\mathbf{C}_{i,enc}$, $\mathbf{I}_{i,bar}$, $\mathbf{I}_{i,hold}$ are one-hot note pitch vector, one-hot chord vector, binary new bar indicator and one-hot hold duration vector respectively.

Chapter 3

Model Descriptions

3.1 Melody Sequential Model

3.1.1 Model Architecture

The final model architecture incorporated into HymnTron included 2-layer LSTM networks, attention wrappers, dropout wrappers, and a softmax layer. Each layer has a purpose suited to its strengths.

LSTM networks were first introduced in [2] by Hochreiter and Schmidhuber. They work superbly well on a variety of problems and are now included in most state-of-the-art models. Furthermore, they are explicitly designed to solve the long-term dependency problem erected by simple recurrent neural networks (RNNs).

The attention wrapper is a trainable weight vector that signals which time step to focus on based on the input. It saves computational cost by learning which past time step is important to determine the outcome of the future step. Since note sequences are mostly comprised of no events, if given the signal to create a new event, it makes sense intuitively to not look at all the time steps in the past.

To recall, the goal of this paper is to compose automatic music that is indistinguishable from a human's composition. However, the machine made music loses credibility if it starts composing music too similar to the original training set. In the field of deep learning, this is commonly known as overfitting. If the model is overfitted on the training set, creating new music will become immensely difficult. Dropout wrapper from [5] is a fitting way to generalize the learning process. Depending on the hyperparameter of the dropout layer, the wrapper keeps only a certain percentage of the neurons and the connecting weights during the training phase, making it harder to learn. With this wrapper, the model is generalized well and can branch into new musical compositions.

The final layer of the model, softmax, converts an m dimensional time step into a probability vector. It is defined as follows:

Definition 3.1. *Let* $\mathbf{x}$ *be a time series encoded note sequence with length* n*. Then,*

$$\mathbf{x} = \{E_1, E_2, ..., E_{n-1}\}. \tag{3.1}$$

In addition, let the final logits of the model with m *features at time step* i *be*

$$f_{logits,i}(\mathbf{x}) = (f_{1,i}, f_{2,i}, ..., f_{m,i}). \tag{3.2}$$

Then, the softmax layer corresponding to the final logits layer is

$$\sigma(f_{logits,i}(\mathbf{x}))_j = \frac{e^{f_{j,i}}}{\sum_{k=1}^{K} e^{f_{k,i}}}, \text{ where } j \text{ is the index of the labels.} \tag{3.3}$$

At time step i, the softmax layer returns a probability vector displaying the likelihood of a specific label for the next time step. The outputs of this layer can be capitalized during the generation phase for statistical analysis and constraints.

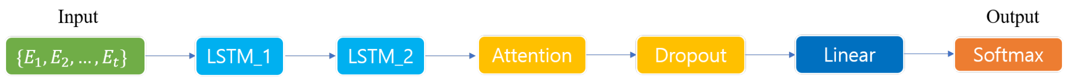


Figure 3.1: Model Structure.

3.1.2 Loss

Let $f : \mathbb{X} \to \mathbb{Y}$ be a predictor. Our goal is to obtain a predictor which satisfies $f(\mathbf{x})=y$ by the learning algorithm. In order to monitor the learning process, loss between the ground-truth label and the predicted label, $\mathfrak{L}(f(\mathbf{x}), y)$, are measured in between epochs. Recall that $\mathbf{x}$ is a time series encoded note sequence from 3.1 and y is the corresponding label. The corresponding label and loss function can be defined as follows:

Definition 3.2. *Let y be the corresponding label to (3.1), such that*

$$y = \{lab_2, lab_3, ..., lab_n\} \textit{ where } lab_i = \mathbf{N}_{i,enc}. \tag{3.4}$$

Finally, the empirical loss $\mathfrak{L}_{emp}$, over M samples, is defined as

$$\begin{aligned} &\mathfrak{L}_{emp}(f) = \tfrac{1}{M}\textstyle\sum_{i=1}^{M} \mathfrak{L}(f(\mathbf{x}_i), y_i), \ \textit{where} \\ &\mathfrak{L}(f(\mathbf{x}), y) = -\textstyle\sum_{i=1}^{M} y \log(f(x)). \end{aligned} \tag{3.5}$$

Since the main concern of this work is not the choice of the loss function, nor the optimization of the loss function, it will be left as a statement.

Chapter 4

Generation

4.1 Beam Search

The motivation behind the application of beam search is to create diverse, yet musically astounding, compositions. Beam search is a greedy search algorithm that does not consider all the possible branches. It only extends its branches from promising nodes. Although the number of search branches are limited, it can save a lot of memory. As illustrated in figure 4.1, only the

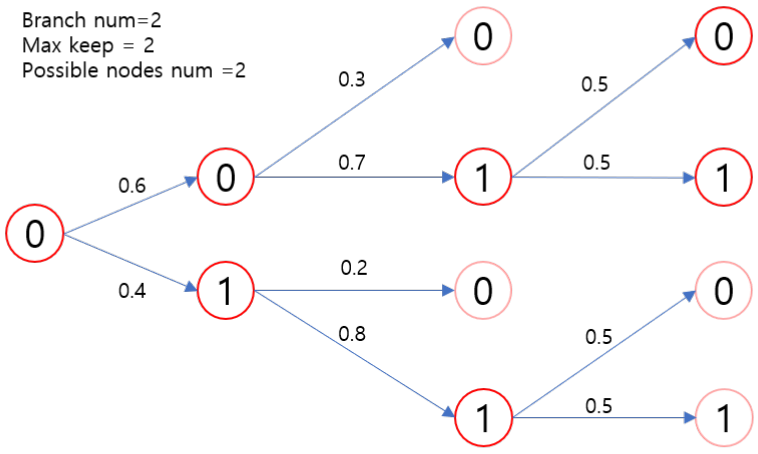


Figure 4.1: A Simple Beam Search Example.

nodes with the highest conditional probability are kept. By applying this concept to music composition, the search algorithm must keep a limited number of search routes at each time step and branch off from the final nodes in the last step. Recall that the softmax layer represents the likelihood of each notes at time $t+1$ given the encoded note sequences up to time t.

The question still remains on how one can effectively sample the branches to ensure a diverse set of branches. To effectively sample the branches, K-Means Clustering is utilized to group similar compositions. If the number of total branches exceed the maximum keep parameter, the branches are clustered into maximum keep parameter groups. The scoring functions in the following subsections work with the clustering algorithm for the selection process.

4.2 Temperature

Temperature is a variable that adjusts the softmax probability vector after the training is completed from [6]. It divides the final logits vector element-wise to increase or decrease the entropy. In other words, it either ensures or undermines the prediction. Since the accuracy during training phase is determined by the argmax function of the softmax vector, the temperature parameter does not affect it. The relevance of the temperature setting lies heavily with the beam search tree. If the probability vector is skewed towards one label, being sure of the prediction, the diversity of routes that the beam search tree can take becomes quite narrow. Therefore, we lower the entropy level in order to sample diverse routes from the seed note. Temperature infused softmax layer is defined as follows:

Definition 4.1. *The temperature parameter p, infused into the softmax layer from 3.3 is*

$$\mathbf{T}_{i,j}(p) = \sigma(\frac{f_{logits,i}(\mathbf{x})}{p})_j, \text{ where } p > 0. \tag{4.1}$$

4.3 Scoring

The creators of Magenta [6], considering the difficulty of evaluating music, decided to make use of the logarithm score to measure the quality of music. They agreed that it is not a reliable source of music quality measurement as music quality is measured subjectively. Furthermore, the branch from the beam search tree with the best logarithm score does not indicate the "best" music. It represents a single composition closest to the original dataset. However, recall that the goal is to create new music pieces **different** from the training dataset. To achieve that goal, there must be additional metrics to monitor the pattern. Specifically, perplexity scoring and difference scoring algorithms are applied for monitoring.

Definition 4.2. *Let* $\mathbf{B}_k$ *be a single branch from the beam search tree, and let* $\mathbf{O}$ *be the set of original compositions from the training set. First, the* **Logarithm Score** *of* $\mathbf{B}_k$ *in [6] is defined as*

$$\mathbf{LS}(\mathbf{B}_k) = \sum_{i=2}^{n} \log(\sigma(f_{logits,i}(\mathbf{x}))_j). \tag{4.2}$$

Second, the **Perplexity Score** *of* $\mathbf{B}_k$ *is defined as*

$$\mathbf{P}(\mathbf{B}_k) = \frac{\sum_{i=1}^{n} \mathbf{I}_i}{n}, \text{ where } \mathbf{I}_i = \begin{cases} 0 & \text{if } \mathbf{B}_{i,k} = \text{No event}, \\ 1 & \text{otherwise.} \end{cases} \tag{4.3}$$

Third, the **Difference Score** *of* $\mathbf{B}_k$ *is defined as*

$$DS(\mathbf{B}_k) = \frac{\sum_{i=1}^{n} \mathbf{IN}_i}{\sum_{i=1}^{n} \mathbf{IU}_i}, \text{ where}$$
$$\mathbf{IN}_i = \begin{cases} 1 & \text{if } \mathbf{B}_{i,k} = \mathbf{O}_i \& \text{ both events}, \\ 0 & \text{otherwise.} \end{cases} \text{ and } \mathbf{IU}_i = \begin{cases} 1 & \text{if } \mathbf{B}_{i,k} \text{ or } \mathbf{O}_i = \text{Event}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

The main purpose of perplexity scoring is to prevent the branches from choosing the "no event" node consecutively. It simply measures the ratio of note events in the full sequence. Branches with mainly "no event" labels will return low perplexity scores, and branches mainly with "event" labels will display high perplexity scores. Discarding all branches with low perplexity scores will guarantee that the final branches will not contain an overabundance of "no event" labels. That being said, it all comes down to the user's choice of parameters. One chooses the minimum and the maximum perplexity scores and only the branches within that score will be kept during the beam search process. If one desires a composition with frequent event changes, a higher minimum perplexity score is set. On the other hand, desiring a longer note duration will mean setting the maximum perplexity score low.

Difference scoring compares the note signals between the chosen branch $\mathbf{B}_k$ and the original compositions $\mathbf{O}$. However, it does not consider the pitch of the notes, since two compositions can sound very different with the same sequence of notes but using different rhythms. Again, the user decides the range of **DS** to keep in between searches. Choosing a higher score range will keep the branches that are similar to the original rhythms and a lower range will keep the branches that are different to the original rhythm.

4.4 Constraints

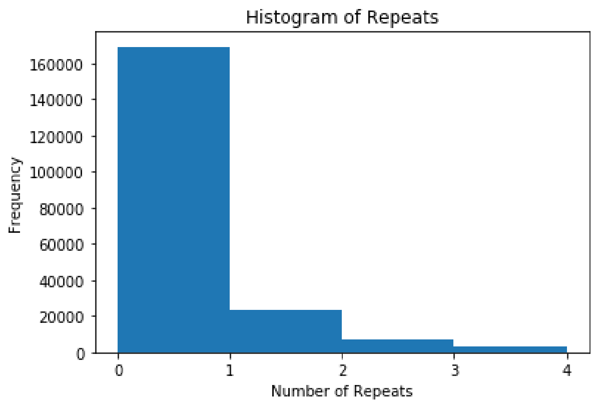


Figure 4.2: Repeat Histogram.

During the generation phase, some rudimentary music theory and prior statistical analysis are applied to prevent the model from choosing the wrong branch. Again, it will depend on the genre of music to determine the music theory that should be applied for the constraints. Therefore, basic hymn theories will be reflected on the constraints. Since constructing complicated music rules defeats the whole purpose of this work, only basic rules have been given. For example, it is quite uncommon for a note to jump an interval greater than a perfect fifth.

After conversing with professional composers, it became clear that repetition was a problem with the earlier models of HymnTron. The final model is given a “dice roll” which mimics the statistical behaviour of a repetition. The number of repetition determines the success rate of the roll. If the roll is successful, the model is given a token that allows repetition.

Chapter 5

Experiments

5.1 Experiment Premises

The survey is comprised of seven questions. To distinguish the candidates that are either professionals in the field of music or experts on hymn music, first two questions are addressed to indicate the level of musical background, and familiarity with hymn music. The following five questions are set up so that each question is given two options. One is a human made hymn music melody, and the other is a machine generated hymn music melody. The participants are asked to identify which one sounds most like hymn music. The survey will remain open at https://www.surveymonkey.com/r/YLNVJWZ and is open to anyone who is interested in taking the survey.

5.2 Results

Just over a month of collecting participants for the survey, 49 people have participated. The results that we expect to see, if we assume that we have created hymn music that are indistinguishable from human composition, is around 50%. Statistically speaking, it would mean that the participants have guessed which one seems more suitable since they are given a choice to pick between a machine generated piece and a human composed piece. Despite the small number of participants, the final result of overall 53% accuracy is quite notable.

Hymn Familiarity	Level of Musical Background			
	Beginner	Intermediate	Expert	Total
Beginner	11 43.6%	5 44%	0 NA	16 43.7%
Intermediate	10 58%	17 57.6%	3 46.7%	30 56.6%
Expert	0 NA	0 NA	3 66.7%	3 66.7%
Total	21 50.5%	22 54.5%	6 56.7%	49 53.0%

Table 5.1: Results of the HymnTron survey. The two rows in each cell represent the number of participants and the mean accuracy respectively.

Chapter 6

Conclusion

In this thesis, we created a constrained DNN based beam search generation model. Raw MIDI data was processed through transformation, quantization, refinement and encoding. A two layer LSTM network with attention wrapper was used as the foundation of the generative model. This generative model returns a vector of probability distribution, indicating which event is most likely to occur in the next time step. Beam search is used hand-in-hand with the generative model to create numerous branches. Each of these branches represent a single musical composition. Constraints are placed on the label vector so that musically indecent events are not chosen during the beam search phase. Additionally, scoring algorithms are utilized during branch filtering. Generated hymn music pieces are compared to the human composed hymn music through human survey. Despite the low number of participants, the result was satisfactory with overall accuracy of 53% .

Acknowledgement

The HymnTron project was only possible through help from numerous people. First and foremost, I would like to thank Professor Myungjoo Kang. Before he took me in his care, I was without goal and motivation. He introduced me to Deep Learning which now I have strong passion for. Second, a big thank you to TJ Media for providing me with the hymn music data. Third, some crucial encoding process was possible through the help of our composer friends at BRT Entertainment. Fourth, I'm grateful to doctor Hyeonuk Kim for mentoring and guiding me through studying deep learning algorithms. Lastly, I would like to thank Margaret Beinarovics for editing this dissertation.

Bibliography

[1] F. A. Gers, J. Schmidhuber, and F. Cummins, *Learning to forget: Continual prediction with lstm*, (1999).

[2] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural Computation, 9 (1997), pp. 1735–1780.

[3] Z. C. Lipton, J. Berkowitz, and C. Elkan, *A critical review of recurrent neural networks for sequence learning*, arXiv preprint arXiv:1506.00019, (2015).

[4] R. Pascanu, T. Mikolov, and Y. Bengio, *On the difficulty of training recurrent neural networks*, in International Conference on Machine Learning, 2013, pp. 1310–1318.

[5] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting.*, Journal of machine learning research, 15 (2014), pp. 1929–1958.

[6] The Magenta Team, *Magenta*. `https://github.com/tensorflow/magenta`, 2016.

[7] M. Tomczak, *Bachbot*, Master's thesis, University of Cambridge, England, 2016.

국문초록

최근의 개발로 인해, 딥 뉴럴 네트워크 (Deep Neural Networks)를 사용하기위한 많은 어플리케이션이 존재한다. 그 중 하나는 음악 작곡이다. 이 논문에서 우리는 인간 작곡과 구별 할 수없는 음악을 작곡하려고 시도한다. 시드 노트로 작곡을 시작한다. 다음으로 LSTM을 사용하여 주어진 시드 노트와 코드 진행에서 수많은 멜로디를 작곡한다. 마지막으로, 우리는 빔 검색 알고리즘을 통해 원하는 수의 컴포지션 만 유지하기 위해 다양한 스코어링 알고리즘 함수를 사용한다. 인간 조사는 품질 테스트를 위해 수행된다.

주요어휘: 음악 작곡, 빔 검색, 깊은 신경망
학번: 2016-22071