Music Composition with Deep Learning

by

Shao-Fan Wang
Tzu-Ping Chen
Yuan-Lin Hsu
Chee Yang Lo

COEN296: Natural Language Processing
Prof. Ming-Hwa Wang
School of Engineering
Department of Computer Engineering
Santa Clara University

Santa Clara, California
March 20, 2018

Music Composition with Deep Learning

Shao-Fan Wang
Tzu-Ping Chen
Yuan-Lin Hsu
Chee Yang Lo

Santa Clara University
March 20, 2018

ABSTRACT

Artificial intelligence is becoming more advanced than ever, and along with the increase of processing power, can be applied to a variety of elements of our daily lives. We propose a model using recent developments in deep learning to explore the potential of computer-generated music.

Contents

List of Figures

Chapter 1

Introduction

1.1 Objective

The core objective of this project is to develop a system that is able to create music that is aesthetically reasonable using deep learning methods.

1.2 Problem Description

The field of deep learning is a flourishing one. Advancements in methods appear frequently, and many researchers are exploring its potential applications. Music is one such example, and there have already been numerous endeavors towards creating a system to generate musical sequences that humans can enjoy listening to. However, many of the systems proposed so far have not been able to close the distance between computer-generated music and music written by human composers. If this challenge is accomplished – that is, if a human listener genuinely enjoys an artificially composed song – the implications of this success are vast and significant.

1.3 How the Problem Relates to Class

Music is represented in sequential form, as a series of musical notes. Many deep learning techniques have been used in the field of natural language processing, which require text – typically a sentence or a document – as input. These inputs are also sequences, albeit composed with different symbols. We are interested in seeing whether the techniques used in the field of NLP can be applied to music as well. In addition, both music and language use sets of structuring rules to govern composition. Just like how we have developed systems that can parse, understand, and produce text, it should be possible to do the same for music.

1.4 What Other Solutions Lack

Previous attempts at generating music involve methods like recurrent neural networks (RNN), genetic algorithms (GA), and others. While these machine learning techniques are capable of learning music structure, they haven't been able to carry this knowledge over to composing music that seems plausibly written by a human composer. This implies that previous solutions lack the creativity needed to successfully introduce variations on a ”main” melody that they may detect, and convince listeners that their compositions are not artificially generated.

1.5 Why Our Solution is Better

We believe that the design of generative adversarial networks (GAN) provide an interesting and possibly better way to incentivize the system to learn how to better compose music. By joining a RNN model with a GAN, the combined system should be able to produce a novel output sequence.

1.6 Scope of Investigation

The project will focus on the implementation and the performance of the designed system.

Chapter 2

Background

2.1 Definition of the Problem

A big challenge in music composition is to create a model that is easily trainable and capable of reproducing the long-range temporal dependencies of music. On top of this, we seek to enable the model to introduce creativity into the composition process.

2.2 Theoretical Background

Recurrent neural networks (RNN) and long short term memory (LSTM) cells are popular deep learning methods that have come about recently, due to the fact that they are designed to process sequential data and capture the temporal structure in their input.

The idea of RNN is to use sequential information. In a traditional neural network the assumption is made that all inputs are independent of each other. However, that is not the case every time. For example, we have to know the previous words to predict the next word in a sentence. This is a typical application of an RNN because it is able to remember information about the preceding sequence of inputs. That is, the RNN network contains a loop structure, allowing information to persist.

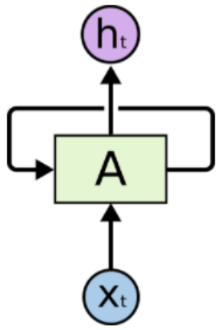


Figure 2.1: A single-layer recurrent neural network

Figure 2.1 depicts a visual representation of a simple RNN. The memory cell A looks at some input x and outputs a value h. Within A lies some set of values to determine the state of the cell. At each time step, the state of the cell is looped back into the cell through a set of weights. This mechanism allows the network to ”remember” over time.

It can be difficult to train standard RNNs to solve problems that require learning long-term temporal dependencies. This is because the gradient of the loss function decays exponentially with time, which is commonly known as the ”vanishing gradient” problem. LSTM networks are a type of RNN that uses special hidden units in addition to standard units. A LSTM unit is a ”memory cell” that can maintain information in memory for long periods of time. A set of gates is used to control when information enters the memory, when it’s output, and when it’s forgotten. This architecture lets them learn longer-term dependencies.

2.3 Related Research

There have been multiple efforts working on the topic of music composition. Two main different approaches are adopted: utilizing novel features extracted from music data to train the models, and applying different algorithms to generate new music. Colombo et al. proposed a novel music representation which contains information of pitch and note duration, and designed a new neural network structure which can accept these features to make their music generator [1]. Nomura and Fukumoto proposed using a distributed interactive genetic algorithm (DIGA) as a composition model [2]. Huang and Wu, as well as Sturm et al., apply deep learning methods, specifically LSTM networks, to music transcription modeling and composition [3, 4].

2.4 Advantages/Disadvantages of Research

New features can be useful, but some information may be lost during feature extraction. Moreover, extra work is needed to extract the new features and design a new neural structure to fit this input feature. DIGA has a mutation process, which can encourage and boost creative composition, but requires human interaction for evaluation, a costly process. While LSTM can retain knowledge well, its training process is more costly than a similarly structured RNN using a standard activation function such as ReLU.

2.5 Our Solution

We propose an RNN and generative adversarial network (GAN) model to build our music generator. Based on the survey by Liu and Ting [5], there has been no recent effort to combine these two models to achieve a music generator. That is, we are the first team to combine these two model to build our music generator.

We believe our solution is better because our model theoretically is more powerful. RNN is suited for handling discrete sequential inputs like music, and GAN is one of the most powerful neural network based models to date.

Furthermore, our proposal eliminates the downsides of the DIGA model: it does not require human involvement for selection.

Chapter 3

Hypothesis

3.1 GAN is suitable for music composition

Generative adversarial networks have been proposed as a way of efficiently training deep generative neural networks. Their ability to generate samples is key to creative music generation. With sufficient training, we believe that a model using GAN will be able to convince a human listener that its music is comparable to that of a novice composer.

Chapter 4

Methodology

4.1 How to Generate Input Data

4.1.1 ABC Music Notation and Encoding Scheme

The ABC notation is a type of musical representation we will use for our model. ABC notation consists of two parts: the header which provides the metadata of the music such as key signature and title, and the body which details the notes in the song.

Figure 4.1 shows an example ABC formatted song. Header tags, such as T for title and X for song number, do not affect music synthesis. The spaces and the newlines in the body are also extraneous characters, and are unneeded for music generation. Therefore, the scraped songs are then processed to only contain 5 important pieces of header information: song type, time signature, unit note length, number of flats, and song mode. Additionally, 2 more metadata headers are generated: number of measures and song complexity, which is the average number of notes played every beat.

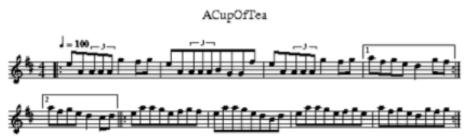
We have two approaches to the input method. Our first option is to feed the ABC file into the model character by character, and our second option is to tokenize the file then using the tokens as a sequence.

After formatting the ABC files, we map the characters to integer encodings by assigning a unique number to each character in the music body, and also to each keyword in each category of the metadata.

4.2 How to Solve the Problem

4.2.1 Designing GAN for sequence model

A conventional GAN has two parts: a generator that applies a deterministic transform to a random sample from input space to target space, and a discriminator that bipartites the target space. The discriminators gradient on target space is used to guide the parameter update in the generator model such that the slightly changed model would produce a slightly more realistic result. The key idea is the target space has to be continuously differentiable for this type of improvement to take place. However, considering the sequence model for most RNNs, the target space usually is



```
X: 1
T: A Cup Of Tea
Z: dafydd
S: https://thesession.org/tunes/3038#setting3038
R: reel
M: 4/4
L: 1/8
K: Amix
|:eA (3AAA g2 fg|eA (3AAA BGGf|
eA (3AAA g2 fg|1afge d2 gf:|2afge d2 cd||
|:eaag efgf|eaag edBd|eaag efge|afge dgfg:|
```

```
X:1
T:ACupOfTea
R:reel
M:4/4
L:1/8
K:Amix
Q:1/4=100
|:eA(3AAAg2fg|eA(3AAABG
Gf|eA(3AAAg2fg|1afged2gf:
|2afged2cd|||:eaagefgf|eaa
gedBd|eaagefge|afgedgfg:|
```

Figure 4.1: Top: Sheet Music, Left: Original ABC File, Right: Formatted ABC File

discrete and the slightly updated generator model would still create a similar output because there is no such output in the target space to reflect the gradual change. Another difficulty for the discriminator guidance for generator is that a score or loss is produced at the end of the sequence generation. The challenge lies in evaluating the partial result of current and future generated tokens.

4.2.2 Theory

To deal with the problem of ineffective gradients over discrete space, we adopt a reinforcement learning framework. Well treat the process of generating token y_t based on the input sequence $Y_{1:t-1}$ as taking an action y_t from state $Y_{1:t-1}$. Consider generating sequence $Y_{1:T}$ as a traversal starting from the initial state s_0, taking a sequence of actions y_1, y_2, ..., y_T. The generator is the stochastic policy model $G_\theta(y_t \mid Y_{1:t-1})$ with deterministic transition. The reward function is the output of the discriminator $D_\phi(Y_{1:T})$. Since the generator can generate tokens at each time step from 1 to $T-1$, but the discriminator can only evaluate a full sequence, we introduce a Monte Carlo Rollout process to estimate the reward for intermediate states. We represent an N-time Monte Carlo search as $Y_1, Y_2, , Y_N = MC^{G_\beta}(Y_{1:t}; N)$. Thus, we reach the following reward function:

$$Q_{D_\phi}^{G_\theta}(s = Y_{1:t-1}, a = y_t) = \begin{cases} \frac{1}{N} \sum_{n-1}^{N} D_\phi(Y_{1:T}^n), Y_{1:T}^n \in MC^{G_\beta}(Y_{1:t}; N) & t \leq T \\ D_\phi(Y_{1:T}) & t = T \end{cases}$$

Based on the policy gradient ascent algorithm, the following equation is derived:

$$
\begin{aligned}
\nabla_\theta J(\theta) &\simeq \sum_{t=1}^{T} \sum_{y_t \in \mathcal{Y}} \nabla_\theta G_\theta(y_t|Y_{1:t-1}) \cdot Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t) \\
&= \sum_{t=1}^{T} \sum_{y_t \in \mathcal{Y}} G_\theta(y_t|Y_{1:t-1}) \nabla_\theta \log G_\theta(y_t|Y_{1:t-1}) \cdot Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t) \\
&= \sum_{t=1}^{T} \mathbb{E}_{y_t \sim G_\theta(y_t|Y_{1:t-1})}[\nabla_\theta \log G_\theta(y_t|Y_{1:t-1}) \cdot Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t)]
\end{aligned}
$$

Since the expectation can be approximated by sampling method. We can update generator parameters with the following equation:

$$\theta \leftarrow \theta + \alpha_h \nabla_\theta J(\theta)$$

For our discriminator, the target is:

$$\min_{\phi} - \mathbb{E}_{Y \sim p_{data}}[\log D_\phi(Y)] - \mathbb{E}_{Y \sim G_\theta}[1 - \log D_\phi(Y)]$$

4.2.3 Model Building and Training

Figure 4.2 shows the overall structure of our project. An RNN serves as the generator, and a 1-D convolutional neural network is the discriminator.

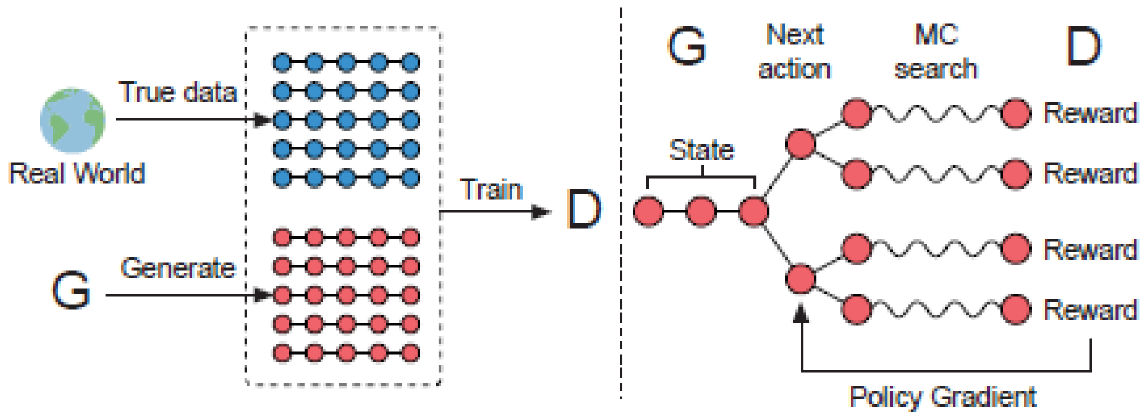


Figure 4.2: A generative adversarial network example

We will also need a copy of generator model as the rollout model used in Monte Carlo search. The generator and rollout models will be pre-trained with training data. Once this process is complete, we prepare negative samples from the trained generator and positive samples from our dataset in order to pre-train the discriminator. After all of our models have been pre-trained, we will follow the typical GAN training process of alternating between the G-step and the D-step. However, during the G-steps the Monte Carlo Policy Gradient based on discriminator reward will be used. In the D-steps, samples from both the generator and the dataset will be used to train the discriminator. Finally, the rollout model would adopt the updated generator parameters before entering the next iteration, and the entire system will be trained until convergence.

4.2.4 Music Generation and Evaluation

Music sequences will be generated by feeding in a seed sequence and allowing the RNN model to run until it reaches the end state. To evaluate our project, we will present some samples created by our model and samples by human composers, then ask class members to distinguish which ones are not artificially generated, and collect the mean opinion scores.

4.2.5 Language and Tools Used

This project will be written in Python, the *lingua franca* of the machine learning community. We will use TensorFlow as our machine learning library to develop our model.

Chapter 5

Implementation

5.1 Design

5.1.1 Data Design

Before we can start training, we have to preprocess our data in order to feed them into our training model. Initially, we transformed a song from ABC notation to a sequence of characters; each character in the former format becomes a different representation in the new input format. Next, we append a special token indicating the end of the file. We separated our training data into 20 notes per line. For each line, we started with 12 notes from the previous line. We enhanced our original input by adding more data, such as the genre of the song, in order to train our models on each genre separately. While doing so, we grouped similar genres together. In addition, for another experiment, we normalized all our songs to the same scale, so all songs are transposed to C major.

5.1.2 Procedural Design

We have previously described our model structure in figure 4.2. To explore different models and their potential effectiveness, we have branched our project into three different experiments.

As figure 5.1 depicts, in Experiment 1, we build two GANs that differ in the discriminator. They both use a character-based RNN to learn the musical sequential structure then generate music. In this experiment, we use two types of discriminators: an RNN and a CNN. This was to see which neural network model performs better on the task of differentiating between true data and the generated data.

In Experiment 2, we built three models: the first and second model are the models from Experiment 1 but with an upgraded generator, and the third model is simply the upgraded generator itself. We added an additional layer in our RNN, resulting in a two-layer network and increasing the number of nodes per layer from 32 to 64. In this experiment we attempt to answer three questions: 1. Does the GAN model perform better when we upgrade the generator only? 2. Do the two GAN models in this experiment outperform their respective counterparts in Experiment 1? 3. Does the GAN outperform the simpler RNN model?

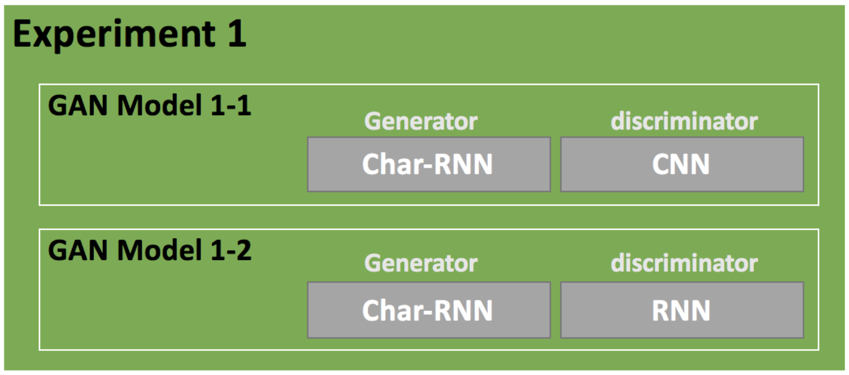


Figure 5.1: Two GAN models in Experiment 1

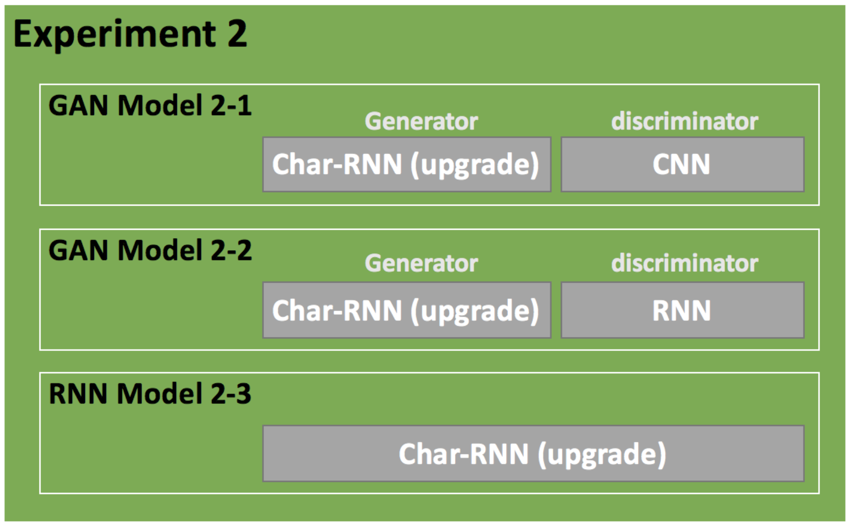


Figure 5.2: Two GAN models and one RNN model in Experiment 2

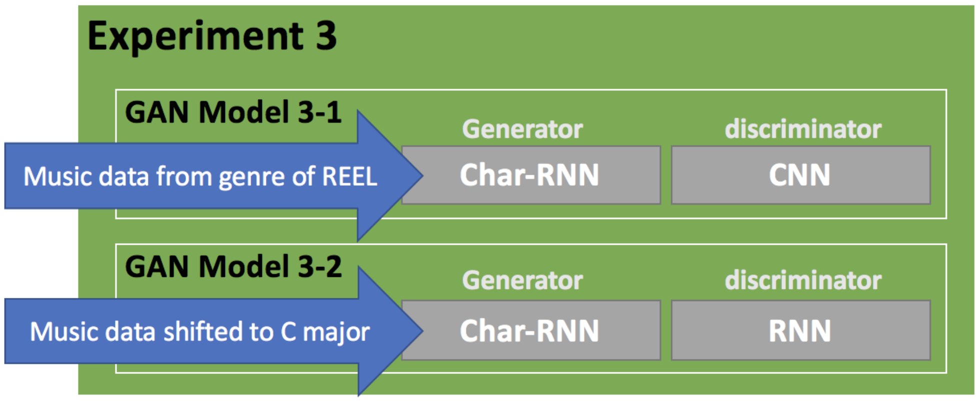


Figure 5.3: Two GAN models in Experiment 3

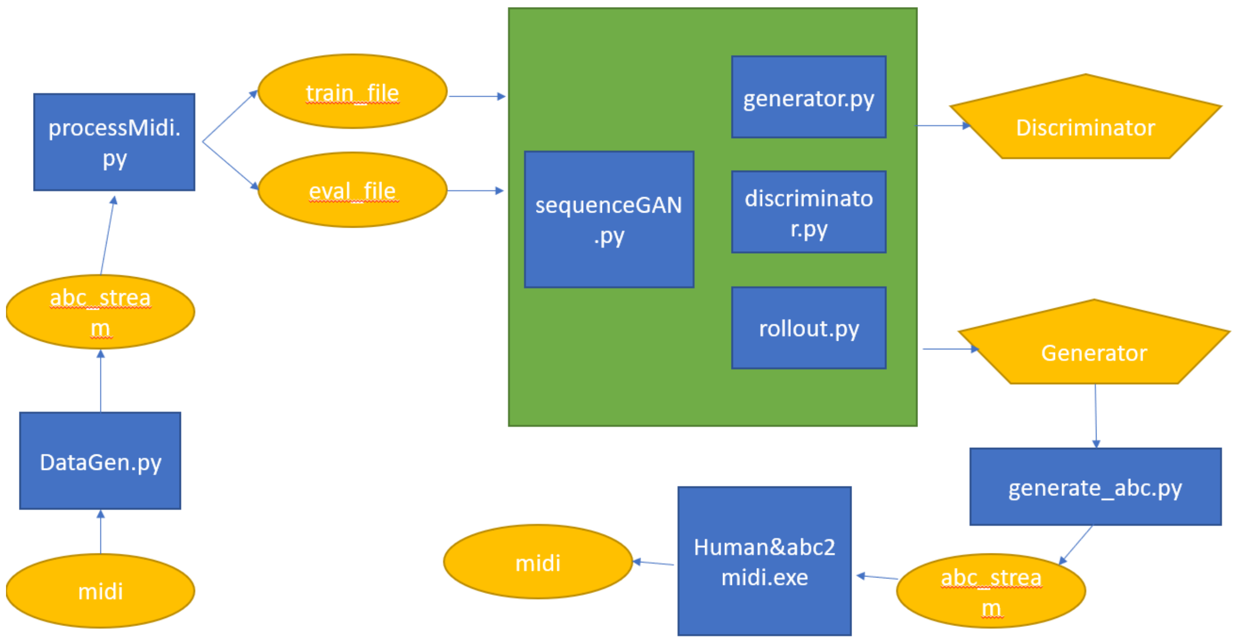


Figure 5.4: Process flow

In Experiment 3, we reused the two models from Experiment 1, but trained those models on altered training data. Our first type of training data consists solely of music from the reel genre. By reducing the number of genres appearing in the training data to one, we hypothesize that the output would be more consistent. We also used the key-normalized data described above, to avoid the cacophony that could arise from using songs with different key signatures in training.

5.2 Code

Figure 5.4 shows the overall process flow of our project. Our data first goes through *DataGen.py*, where we handle musical issues like normalization and ABC conversion. *processMidi.py* further processes the data to produce fixed-length training and test sequences. Next, *sequenceGAN.py* creates the generator, discriminator and rollout model, and is responsible for the two training stages. After training, *generate_abc.py* is called to generate song samples by feeding a start token to the generator and letting it run until the EOF token is generated. In the "Human&abc2midi.exe" block, ABC header data is manually added, then *abc2midi.exe* is used to convert the file to a music file for listening.

Chapter 6

Data Analysis

6.1 Output Generation

After our models complete the pre-training and training processes, we use the generator part of the model to create ABC music sequences. We then transform them back to MIDI format for listening. The generation of ABC notation by the generator model is stochastic; we pass in a start token that is excluded from other vocabulary in data to initialize the model. After that, we let the model ”free run”, using the current timestep’s output as input during the next timestep, until it reaches an end token and stops, as shown in figure 6.1.

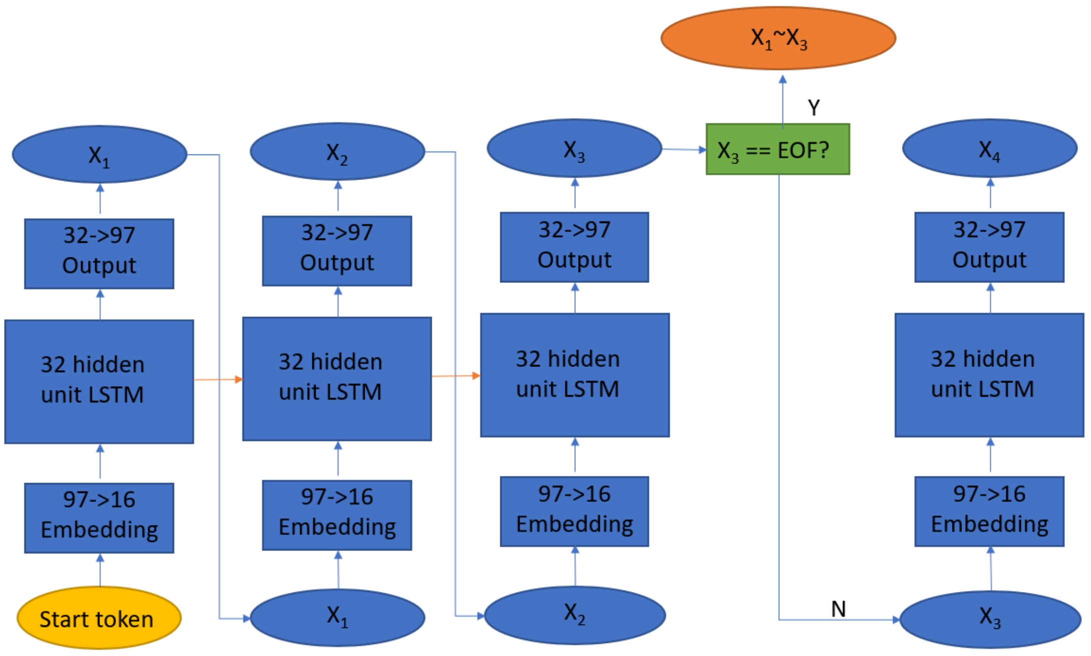


Figure 6.1: Music generation process

The start and end tokens are then removed from the whole sequence, and a token to ABC mapping is used to complete the translation. The ’body’ of music is thus generated.

To complete the generation of ABC notation, we also prepend a header to specify musical metadata such as tune and rhythm. The header we use are as follows:

```
X: 1
R: reel
M: 4/4
L: 1/8
K: Cmaj
```

According to the class of music we train, we changed the R value to other corresponding values. An example final result before converting to ABC is as follows:

```
X: 1
T: Difyrrwch Y Brenin
R: reel
M: 4/4
L: 1/8
K: Cmaj
BdBAGF E2E>FA>c (3efe {e}(fe/d/ de  f2 d ee 1 "Seen "d2 (de)
```

6.2 Output Analysis

We set aside 20% of our data for evaluation, computing the cross-entropy loss between that and the generator prediction as test loss. During adversarial training, test loss will usually increase after 10 epochs, so early stopping is employed. We observed that the loss does not decrease by much during adversarial training, compared to the amount in pretraining. This may be due to a strong discriminator memorizing each example in the training set, causing the generator to overfit. We believe better results can be generated with more training data and the use of regularization.

6.3 Comparison Against Hypothesis

We were successful in creating sequences of musical notes using GANs, but to determine whether the output affirms our hypothesis, we must enlist the help of volunteers to assess and compare our songs against songs from other works. We discuss our results below.

6.4 Discussion

To evaluate our models further, we took samples generated from each model, as well as songs from our true data and samples from Agarwala et al., for a group of 20 volunteers to listen to and evaluate. Each volunteer was given a random sample of the song collection, but will always listen to at least one song from each source. They will assign scores to six songs they listen to on a scale from 1 to 9, based on Huang et al.'s evaluation metric. On this scale, a score of 1 means "completely random noise", a score of 5 means "musically plausible", and a score of 9 means "composed by a novice composer."

Our survey results are shown in two tables. Table 6.1 contains the average scores given to songs generated in our experiments, and table 6.2 shows the average scores given to songs from Agarwala et al. and samples from our true data, which are composed by human composers.

Ex1.1	Ex1.2	Ex2.1	Ex2.2	Ex2.3	Ex3.1	Ex3.2
5.9	4.3	4.0	4.2	4.8	3.7	4.7

Table 6.1: Average scores for all three experiments

Stanford1	Stanford2	Human1	Human2	Human3
6.5	5.4	8.1	7.1	7.5

Table 6.2: Average scores for external samples

Some interesting observations can be found from these results. As we increase the complexities and theoretical power of our models, the average rating actually decreases. Our original model appears to perform the best, and the next best model is the simple RNN model. The Char-RNN and CNN configuration performs comparably to the results of Agarwala et al. Also, training a model on a single genre of music does not seem to improve the aural pleasantness of that model's output, but normalizing the key helps.

Visualizing some of our generated songs reveals that some temporal structure was learned in the training data. Figures 6.2 and 6.3 show that the models are able to learn, use and reuse musical patterns.

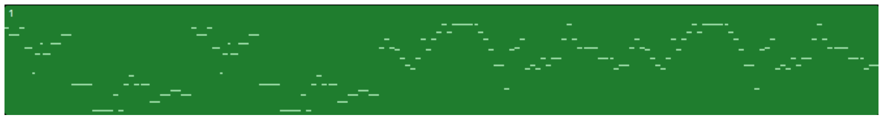


Figure 6.2: A generated song from Experiment 1.1

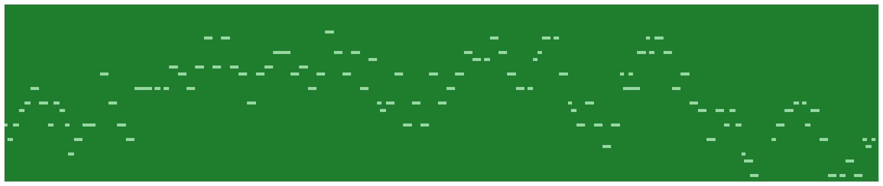


Figure 6.3: A generated song using key-normalized training data

Our evaluations could indicate that GANs may not be the best approach to computer-generated music, perhaps due to differences between the discriminator and a human listener. It is also possible that training on music that is closer to what we listen to could provide better results, but this could lead to vanishing gradients due to saturated activations. In other words, the differences between the training data and the generator's outputs could be so large that the generator is not incentivized to beat the discriminator.

Chapter 7

Conclusion

7.1 Summary and Conclusion

In this project, we attempted to design a system for learning temporal structure in a musical sequence, and using that knowledge to compose original melodies. We did this using the concept of the generative adversarial network as a base, with our objective being to train a model to compose music at the level of a novice human composer. We have shown that the GAN can compose music that is a step above completely random noise. In the process, we designed models with slight variations in an attempt to discover the most effective hyperparameters and structure. Upon doing so, we found an interesting observation: just because a model can handle more complexity does not mean it can create more musically plausible sequences.

7.2 Future Work

Our current work is just the beginning of the development of computer creativity. Needless to say, there is much room for improvement, and as new deep learning techniques and methods become feasible, we are certain that they will bring about performances that start to resemble plausible human compositions.

Chapter 8

Appendix

8.1 Appendix A: Source Code

For our source code, please refer to `https://github.com/lo16/coen296-group-project-final`.

Bibliography

[1] F. Colombo, S. P. Muscinelli, A. Seeholzer, J. Brea, and W. Gerstner. Algorithmic Composition of Melodies with Deep Recurrent Neural Networks. *ArXiv e-prints*, June 2016.

[2] Kota Nomura and Makoto Fukumoto. Distance analysis of music melodies created by distributed interactive ga. pages 80–83, 09 2017.

[3] Allen Huang and Raymond Wu. Deep learning for music. *CoRR*, abs/1606.04930, 2016.

[4] Bob L. Sturm, João Felipe Santos, Oded Ben-Tal, and Iryna Korshunova. Music transcription modelling and composition using deep learning. *CoRR*, abs/1604.08723, 2016.

[5] Chien-Hung Liu and Chuan-Kang Ting. Computational intelligence in music composition: A survey. *IEEE Trans. Emerging Topics in Comput. Intellig.*, 1(1):2–15, 2017.

[6] Nipun Agarwala, Yuki Inoue, and Axel Sly. Music composition using recurrent neural networks.

[7] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. 2017.