Music Composition using Recurrent Neural Networks

Nipun Agarwala[1], Yuki Inoue[1], and Axel Sly[2]

[1]Department of Electrical Engineering, Stanford University
[2]Department of Computer Science, Stanford University

{nipuna1,yinoue93,axelsly}@stanford.edu

Abstract

Music artists have composed pieces that are both creative and precise. For example, classical music is well-known for its meticulous structure and emotional effect. Recurrent Neural Networks (RNNs) are powerful models that have achieved excellent performance on difficult learning tasks having temporal dependencies. We propose generative RNN models that create sheet music with well-formed structure and stylistic conventions without predefining music composition rules to the models. We see that Character RNNs are able to learn some patterns but not create structurally accurate music, with a test accuracy of 60% and fooling only upto 35% of the human listeners to believe that the music was created by a human. Generative Adversarial Networks (GANs) were tried, with various training techniques, but produced no meaningful results due to instable training. On the other hand, Seq2Seq models do very well in producing both structurally correct and human-pleasing music, with a test accuracy of 65% and some of its generated music fooling $\sim 70\%$ of the human listeners.

1 Introduction

Similar to language, music acts as a form of expression where combinations of notes can communicate a range of emotions. A key note here is that music has to be both expressive and precise. For example, classical music is well-known for its meticulous structure and emotional effect. Composing music is difficult for beginners because they need to first understand the "language" of music, such as time signatures, maintaining unit beats consistently in a song, all while incorporating artistic novelty. We explore the question: Is it possible to model this dynamic musical structure and effects using deep learning?

Recurrent Neural Networks have shown great promise and results in modeling temporal dependencies and structure. They have been used in text generation, question-answering, building chat bots, sentiment analysis etc. with great success. We extend the application of RNNs to building a music generator using character-based neural language models such as Character RNN [1] and Seq2Seq [2], with GRU and LSTM cells [3] including bi-directionality and attention [4]. We represent music in text format using the ABC notation [5]. This data suits us well because it is readily available, character-based so standard NLP tactics can better be utilized, and simulates an environment that is closer to how human composers create music. Although other more expressive audio formats exist, such as MIDI and Mel-Frequency Cepstral Coefficients (MFCCs), they are mainly sound-based and thus may not be transcribable to sheet music.

2 Related Work

Most previous work in music composition makes use of MIDI or raw audio format to learn complex polyphonic structure in music [6]. In the case of raw audio, it is common to map the piece onto the mel-frequency cepstrum coefficients (MFCCs) [7]. Specifically, MFCCs approximates the human auditory system's response more closely by adding more weights to pitches human ear captures more. However, both data formats are of continuous in nature, and thus may not be able to be converted to a sheet music, which is discrete. So in order to guarantee a valid conversion to sheet music, the sound files are discretized and assumptions are imposed. For example, most of the papers on this topic enforce 4/4 time signatures, key of C, and define the fastest moving notes as a sixteenth note, ignoring any notes that are faster. By imposing such a restriction, the continuous problem can be simplified to a discrete problem, where there are 4 beats to a measure, and 4 discrete time steps in a beat. However, this simplification also implies that the models do not need to learn how to *compose* music- no matter what kind of output a model produces, because 16 time steps always define a measure, it can always be mapped back to a valid

sheet music. Therefore, these models are only learning to *combine sounds* together, rather than learning musical syntax and *compose* music. On the other hand, the ABC notation simply just defines a mapping between each character or a group of characters to specific symbols on a sheet music. Therefore, this better simulates how human composers create music, and the model would have to learn the *grammar* of the music (ex. place a bar line after running out of beats in a measure), because a randomly generated text will not have a valid mapping to a sheet music.

The models used in previous work have varied, including usage of Recurrent Neural Networks combined with Restricted Boltzmann Machines (RNN-RBM) [8] and Character RNN [9]. The RNN-RBMs focused on creating polyphonic music, albeit using piano rolls while the Char-RNNs focused purely on Irish folk music re-creation and not arbitrary music generation. Our focus is on general monophonic music composition across different genres.

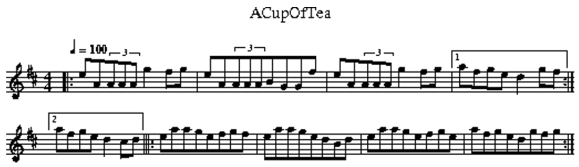
3 Dataset and Preprocessing

3.1 Data Acquisition

Unfortunately, there is no data repository available for ABC formatted music to our knowledge. Therefore, the first step of the project is to collect the data from popular ABC music websites such as *abcnotation.com* and *thesession.com*. In total, approximately 17,500 songs from *abcnotation.com*, and 16,500 songs are downloaded from *thesession.com*. These songs come from genres like Jig, Waltz, Polka, Hornpipe, Reel and Chinese, consisting of 19 genres in total.

3.2 ABC Music Notation and Encoding Scheme

ABC notation has 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. Fig. 1 shows an example ABC formatted song. There are header tags such as T (title) and X (song number) that do not affect music synthesis. The spaces and the newlines in the body are also extraneous characters, unneeded for music generation. Therefore, the scraped songs are then processed to only contain 5 important header information (song type, time signature, unit note length, number of flats, song mode). Additionally, 2 more meatadata headers are generated: *number of measures* and *song complexity*, which is the average number of notes played every beat.




```
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 1: Top: Sheet Music, Left: Original ABC File, Right: Formatted ABC File

We then augment the dataset by transposing them to 4 random keys, which involves chaning the pitch of every note in a song by a constant step. Transposing a song moves every note in a song only by a constant value, and hence transposed songs are valid songs because the note relationships are still intact. Despite the relationship being intact, the notes appear different from the original song, making transposing a good augmentation option.

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. Finally, the encoded songs are split into train/test/dev sets of 80/10/10.

4 Methodology

4.1 Continuous Bag-Of-Words

As a trivial baseline, we used the traditional baseline of Continuous Bag of Words (CBOW). CBOW uses the context window around a word to predict the context word. We used a simple fully-connected neural network having an input, output and a single hidden layer to find the probability of a word in the vocabulary being the context word given the context window. Since we wanted to predict the next music character given some history,

we chose the context window to be the previous n characters (n is a hyper-parameter), instead of the traditional balanced context window around the context word. The CBOW model averages the character embeddings of the characters (initially represented as one-hot vectors) in the context window within the hidden layer and then uses the output layer for predicting probabilities.

4.2 Character RNN

Traditional neural-language models which use word embeddings use word embeddings and hence have to predict probabilities over more than $10,000$ words at a time. This increases training time and the computation needed. As a result, using character embeddings and predicting characters in text is an easier model to train, though it might more likely result in words (in our case, notes) that do not make sense. Additionally, encoding the notes directly would defeat the purpose of our project, which is for our model to learn musical structure and syntax.

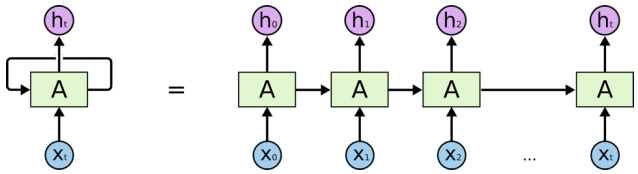


Figure 2: Character RNN Model

A Character RNN model was a simple RNN model: It has n layers of recurrent units like GRU or LSTM, which are unrolled in T time-steps, and the T time steps share the same weights and have hidden size h. One hot vectors of characters from formatted ABC text file are converted to character embeddings using a shred weight matrix W_e^{oxd} and then passed through the Char-RNN to get outputs. These outputs are passed through a shared weight matrix W_c^{dxo} and a softmax layer to get probabilities for the next character. Instead of using a zero vector as our initial hidden state, we created a shared weight matrix W_c^{mxk} for the metadata encodings and then concatenated all the embeddings into the initial hidden state.

4.3 Sequence-to-Sequence

Traditional RNN models use the current word (or character in our case) along with the context until that word to predict the next word or character. To make better prediction of words or characters based on long sequences, Sequence-to-Sequence models were created. They consist of an encode and decode stage. Our encode stage is a normal Character RNN without the weights W_c^{dxo} and softmax layer. We take a character sequence of length L_1 and use it to create a thought vector v, which is the final hidden state at the end of unrolling L_1 time-steps. This is passed into the decode stage which is another Char-RNN model that takes v as the initial state and a <go> token (having a separate encoding and embedding) as the first token to predict the next L_2 characters. We inputs to the encode is a set of L_1 characters. The inputs to the decode are the subsequent $L_2 - 1$ characters, with a <go> token appended at the start. The targets of the decoder are the L_2 characters following the L_1 character inputs to the encode. Since music is a sequence based art, where the next set of sequences to be played depends on the previous k set of sequences played, we hypothesized that this model would work well.

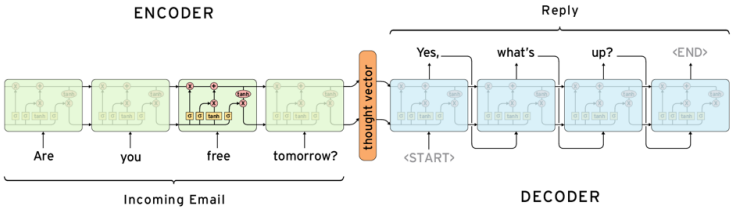


Figure 3: Seq2Seq Model

4.4 Generative Adversarial Networks

No RNN model is truly generative: We need to feed in some input, like a single character or a sequence of characters from real text or song to generate the next n characters. Generative Adversarial Networks (GANs) [10] get around this problem by making a Deep Neural Network (DNN) truly generative. We have 2 components: a Generator and Discriminator, both of which are DNNs. The Generator G takes noise as input and outputs predictions. These predictions are fed into the Discriminator D for a 2 class-classification of whether the input is fake or real. Real data is also fed in an alternating fashion with the output of G to train D on what is real. We now get a minimax game as a result of 2 competing objectives. We update the parameters of D by performing stochastic gradient descent on the loss: $\sum_{i=1}^{n}[\log D(x^i) + \log(1 - D(G(z^i)))]$, where x^i is the real data and z^i is the noise sampled from a prior distribution (Gaussian works best, but Uniform can be used too). The parameters of

G are updated using SGD on the loss: $\sum_{i=1}^{n} \log(1 - D(G(z^i)))$. Intuitively, G is trying to fool D into thinking that its outputs are real, where as D is trying to learn what outputs are real and trying to classify the ouputs of G as fake. Thus, as D gets better at discriminating, G gets better at fooling to maintain the Nash Equilibrium. We thought that such a model would be truly generative and produce human like outputs.

In our case, G was a Char-RNN model which gives a probability distribution for the predictions at each time-step T. We sample a character from the distribution at each time step to get T characters. We obtain a new set of embeddings for these characters using the weight matrix W_e^{vxd} where d is the size of embeddings. These embeddings are passed into a 5 layer Convolution Neural Network (CNN) for classification. The CNN had 2 convolution layers with Leaky ReLU layers in between (for better gradient propagation) and followed by a max-pool layer and 2 fully connected layers. . The initial filter size is dxk, where $1 \leq k \leq T$ with stride size s. The next convolution layer has only $1D$ convolutions. The output layer is a sigmoid layer to get probabilities of the fake and real class. Since we sample characters from the output distributions, we cannot simply backpropogate a loss function. Instead, drawing inspiration from reinforcement learning, we use the output of the CNN is used to backpropagate policy gradients, using the REINFORCE algorithm, of $p(y|x)$ to the samples that fooled the network and $-p(y|x)$ to the ones that did not, where $p(y|x)$ is the conditional probability of the sampled character.

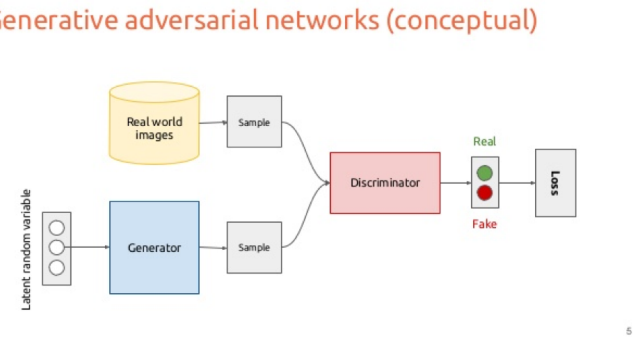


Figure 4: GAN Model

5 Results

5.1 CBOW

Since CBOW was implemented to gauge the baseline for other models, no hyperparameter tuning was conducted. The context size was set to be 15, which is roughly the number of characters for a measure of music. With these parameter settings, the model achieves around 20% accuracy for both the test and the train sets, which implies that the model did not have the expressive capacity to overfit to the data, as expected of such a simple design as CBOW.

5.2 Character RNN

5.2.1 Hyperparameter Tuning

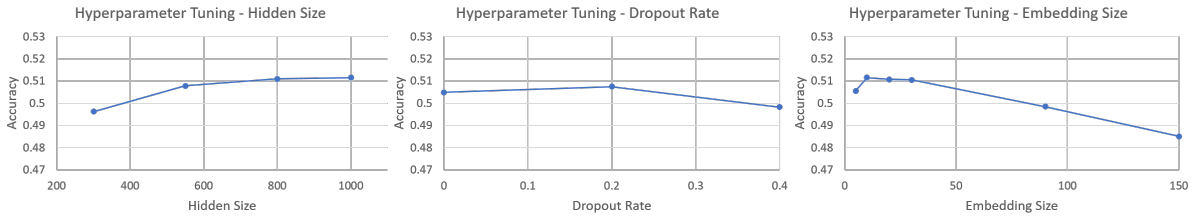


Figure 5: Results of Hyperparameter Tuning for Character RNN

Due to limited time and computing resources, we assumed that the input size and the rest of the hyperparameters are independent of each other. With this assumption, the hyper-parameter tuning can be decomposed into two stages:

1. A grid search on key parameters
2. Input size comparison on the optimized parameters

In the first stage of the hyperparameter tuning, a grid search with early stopping at 2 epochs was conducted on 3 parameters (hidden state size, music body embedding size, and dropout ratio). The grid search (18 configurations in total) was conducted for the 3 cell types (RNN, GRU, LSTM), and the Dev set accuracies of the best parameter settings for each cell type were compared. After 2 epochs, the best model for RNN achieved *39.5%*, GRU achieved *47.5%*, and **LSTM achieved *51.7%***. Therefore, for the rest of the project, only LSTM cells are considered.

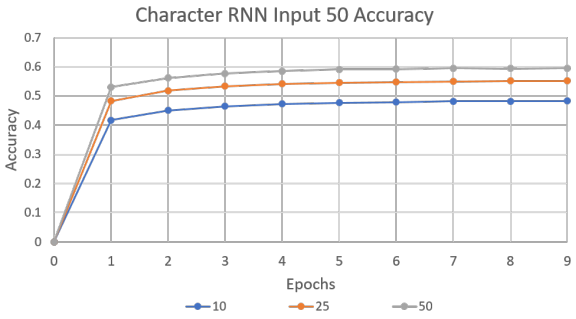


Figure 6: Input Size Comparison

After deciding on using LSTM cells, a more granular grid search of 72 configurations in total was conducted. Fig. 5 shows the effect of each parameter on the dev set accuracy, where the values are calculated using a median value of the grid search result. **In general, a larger hidden size and a smaller embedding size seem to improve the accuracy. We did not make the hidden size any larger, because the hidden state size above a 1000 significantly slowed the training speed. As a result, hidden state size of 800 (chosen over 1000 for a faster training), dropout rate of 0.2, and the embedding size of 20 were chosen for the model**.

The next step is to compare the effects of the window size, using the optimal parameter settings found previously. Fig. 6 shows the dev set accuracy for window sizes of 10, 25, and 50. **The plot clearly shows that the model improves with larger inputs, achieving 60% accuracy for the input size of 50**. Unfortunately, many of the songs collected contained less than 50 characters, so larger input sizes were not examined.

The final resulting model achieves **59.5% on the test set**.

5.2.2 Comparison with Kaparthy et. al.

Since Character RNN model is not typically used to synthesize music scores, and music generation is a hard task to have a standard performance metric on, it is difficult to judge if our model produces a reasonable result. Therefore, we compared the implemented model against the result obtained in the paper by Karpathy et. al [11], the original proposer of the Char RNN model, which claims to have achieved 58% prediction accuracy on Leo Tolstoy's *War and Peace*.

Using a window size of 25 and the best hyperparameter setting found before, our model was able to achieve **60%** on the War and Peace dataset, which shows that our implemented model has at least enough predictive power to predict English texts.

5.3 Seq2Seq

5.3.1 Hyperparameter Tuning

As Seq2Seq model is even slower of a model, the early stopping was set to 1 epoch, and the input sequence size was set at 25 in the interest of time and computing resources.

For this model, hidden state size, embedding size, attention mechanism type, as well as the directionality of cells (i.e. uni- or bi-directional) are tuned. Fig. 7 shows the result of each parameter against median dev set accuracy. **As in character RNN, larger hidden state size improves the accuracy. However, larger embedding size increases the accuracy as well, contrary to the result for character RNN**. So we decided to further investigated the effect of increasing embedding sizes. The bottom plot of 7 shows that the accuracy reaches the maximum around 100. The other two parameters, the directionality of the cells and the attention mechanism, did not seem to affect the accuracy as much, although we do note that Bahdanau attention model was slightly better than the Luong model. Also, the fact that the directionality of the cells does not affect the accuracy implies that music scores have weak to no backward dependencies.

Hidden state size of 800, embedding size of 100, unidirectional cells, and bahdanau attention model were chosen, achieving 65.5% on the test set.

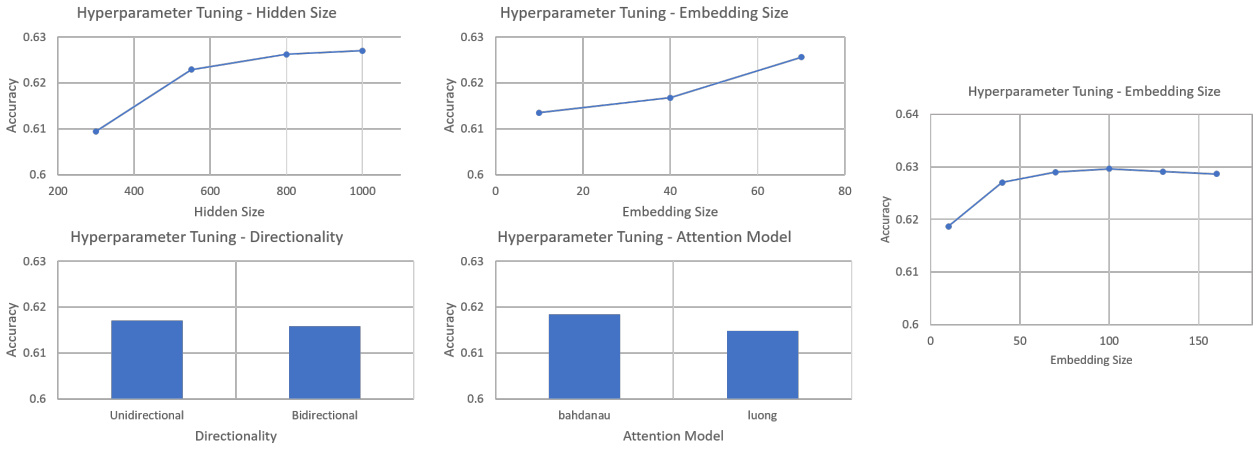


Figure 7: Results of Hyperparameter Tuning for Seq2seq Model

5.4 GAN

Our initial attempt at the GAN proved futile, since training the model was extremely unstable and the Char-RNN kept losing to the CNN and thus kept producing only noise. We used various methods suggested in [12], including label smoothing, where the targets are in the range $[0.8, 1.2]$ for positive predictions and $[0, 0.3]$ for negative predictions and also having a $KL(p||t)$ penalty on the loss function of the Discriminator, where p is the predicted distribution and t is the target distribution. We also tried doing multi-label classification of 20 classes: the 19 genres and the fake class.

(a) GAN accuracies for simple Policy Gradients

(b) GAN accuracies for TRPO Gradient updates

Figure 8: GAN accuracies for different optimizers

We then thought that the RNN might be losing the game since it is a slower and more unstable model to train than a CNN. Hence, we decided to pre-train the Char-RNN for 5 epochs on a new dataset to initialize it well. Subsequently, to balance the model, we trained the CNN on the subsequent outputs of the Char-RNN for 1 epoch. Yet again, the Char-RNN kept losing to the CNN and failed to generate meaningful outputs. Finally, we conjectured that since the REINFORCE algorithm is unstable and is propagating gradients of too large a step size (despite gradient clipping included), we turned to using **Trust Region Policy Optimization (TRPO)** [13]. TRPO uses the following loss function: $\sum_{i=1}^{n} \frac{\pi}{\pi_{old}} r_i + KL(\pi_{old}||\pi)$, where π is the newly predicted distribution, π_{old} is the old distribution and r_i is the reward for the i^{th} sample, which is $+1$ or -1 in our case . Intuitively, this loss function prevents major changes to the policy at each iteration, thus allowing for better learning and is proven to have monotonic improvement of policies. Despite some improvement in training, the Char-RNN still lost the game and was unable to generate meaningful output at test time. As we can see in Fig. 8, we see that TRPO method oscillates much less than the simple policy gradients one. We think that maybe using a Seq2Seq model instead would be learn better and not lose easily, but we would not try it in the time constraint.

6 Discussion

After the models are trained, they are used to generate music samples. This was done by warm starting the models using songs composed by humans. The warm starting songs are separate from the training songs, in order to discourage the models from regurgitating memorized song.

6.1 CBOW

As expected from the train and the test accuracies, CBOW is not able to output anything musical. An example output, after removing the metadata and the warm start characters, appears as seen in Listing 1. Because most of the output does not contain note characters ('a'-'g', 'A'-'G'), it cannot be converted to a sheet music. Therefore, we conclude that CBOW model does not learn anything about music structure.

```
|:FGABcdcf|Bagffedc|Bdgfd|2~eea1e|e=Gd^=~u^^P^t=tt*********
```

Listing 1: Example CBOW Output

6.2 Character RNN

Unlike CBOW, Char-RNN is able to generate samples that can be converted to sheet music. Fig. 9 shows an example of a music sampled from the Char-RNN model. An interesting note here is that the model was able to learn the character to note mapping on its own, the frequency of character outputting. That is why it is able to produce an output that is able to be transcribed into sheet music. This point is supported by the confusion matrix and the PCA projection of the embedding matrix as shown in Fig. 10. There are two squares in the confusion matrix, which correspond to the regions bounded by characters 'a'-'g' and 'A'-'G' ("note characters"). This implies that the largest ratio of mislabelings occur within the note characters. In other words, the model is understanding that those characters have similar functionalities. The faint regions on the off-diagonals of the confusion matrix are confusion between the upper/lower cases of the note characters, suggesting that the model also recognizes that the note characters have similar functions regardless of the cases.

This point is further strengthened by the PCA-projection of the embedding matrix in Fig. 10. This plot shows the lower dimensional representation (dictated by PCA) of the 20 dimensional character embedding for the note characters. Looking at the relationship between the upper and the lower case characters, we are able to see that there exists an 'octave vector', which maps upper and lower cases of the note characters. This implies that the model learned some music structure from the text.

However, Char-RNN model has a major drawback. Although the output may sound passable, it is not able to predict presence of bar lines ('|' in the ABC notation). As a result, the measures are not consistent and the sheet music generated is not syntactically correct.

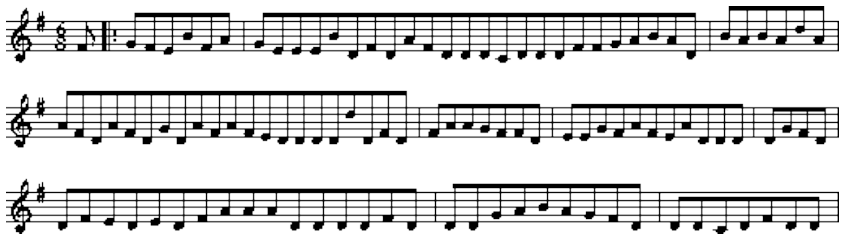


Figure 9: A Sample Music Generated by Char-RNN Model

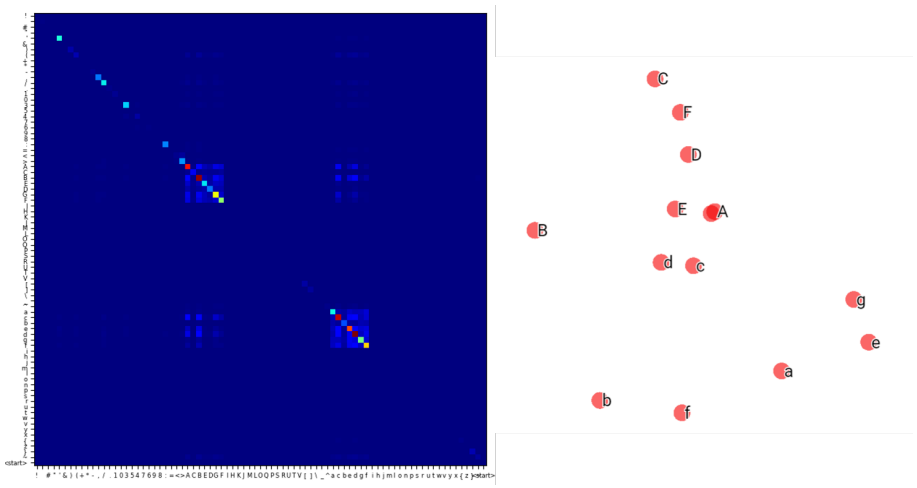


Figure 10: Confusion Matrix (Left) and the PCA Projection of the Embedding Matrix (Right)

6.3 Seq2seq

6.3.1 Generated Music

The sheet music on the bottom left of Fig. 12 shows an example of an output of the Seq2Seq model. Unlike Char-RNN, Seq2Seq is not only able to learn the functionality of the note characters, but also able to learn appropriate placement of bar lines. The resulting music is syntactically correct, and even possess some musical nuances. For example, the model repeats the first measure (warm start) in the third measure, and similar rhythmical figures show up on the tenth and the fifteenth measures as well. **In other words, the model is able to use the warm start notes as the motif of the music, which was something not observed in the Char-RNN model outputs**.

6.3.2 Exploring the Limits of the Model

Promising results from the Seq2Seq model made us try a few more experiments. We first investigated what will come out of the model if we input songs that sounded significantly different from those within the dataset, especially since many of the songs in the dataset are classical music. In order to test this, we converted songs from various genres to ABC format and used them as the warm start. The music on the top of Fig. 12 shows the result for warm starting with 25 characters of the music from the video game *The Legend of Zelda*. As expected, the middle of the song sounds very much like Irish folk song (one can observe the iconic quarter note/eighth note pattern), but it is interesting to note that that the model was able to replicate the warm start sequence in the eighth measure. This further confirms that the model is able to take the warm start characters as a motif to be used in its music, even if it is the first time the model has ever seen that note sequence.

We also applied the model to duet songs. An example ABC duet song is shown in Fig. 11. The ABC notation for multi-phonic songs is slightly tricky: the first two lines of the text represents the music for two voices to be played at once, and the pattern repeats until the end of the song. As one can imagine, adding another voice therefore adds a challenge: the model needs to be able to learn the corresponding beats in the two voices will be many characters away in the text. Since the original dataset did not include duet songs, the model was retrained with a new dataset of 1300 duet songs. The smaller dataset was a result of lack of duet songs written in ABC format. The music on the bottom right of Fig. 12 shows an example of a duet song generated. Despite the difficulty and smaller dataset, the model was able to generate mostly syntactically correct music. Despite few mistakes in the music (such as putting a repeat sign at different locations for the two different voices) and dissonant notes played from time to time, the overall syntax is correct and the two voices even play together at times (ex. second measure). It is interesting that the model is able to learn such a long dependencies, and it shows that the model has many potential applications.

```
[V:1] DE|F3E  F2A2|B2AF E2DE|F2F2 d2cB|c4  z2
[V:2] z2|d2A2 d2f2|g2d2 A2a2|d2d2 d2z2|A2a2 A2
[V:1] c2|d3c  B2B2|ABAF E2B2|ABAF E2FA|B4  z2|]
...
```

Figure 11: Example Duet Song

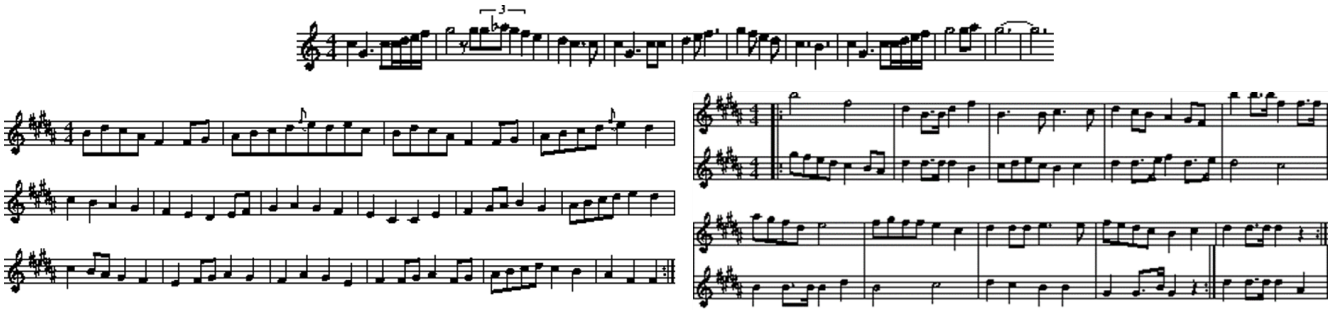


Figure 12: Songs Generated by Seq2seq Model - Warm Started with *The Legend of Zelda* (Top), Monophonic Output (Bottom Left), Duet Output (Bottom Right)

6.3.3 The Effect of Meta Data

In order to assess the effect of initializing the hidden states as the encoded concatenated meta data, the best seq2seq model was trained with a zero-initialized hidden vector. Fig. 13 shows the test accuracies for the models with/without meta data hidden state initialization. It appears that the model is able to learn a little faster due to the meta data, but the overall convergence is the same for both models. This further shows that the model is able to learn music structure without explicitly knowing the metadata, in fact ignoring it.

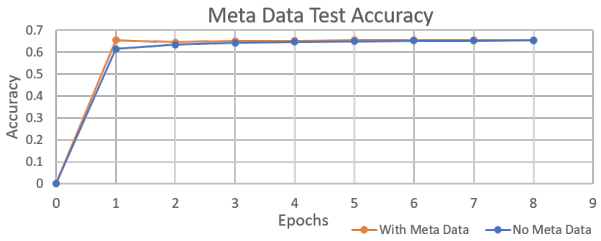


Figure 13: Test Curves for with/without Meta Data

6.4 Survey

Since there is no standard and concrete metric to measure the quality of music, we decided to conduct a survey to receive objective feedback on the quality of the songs generated by each model. 3 songs by Char-RNN and seq2seq models, as well as human composed music were chosen (total of 9 songs) for this survey, and in order to avoid warm starts affecting surveyee's decision, 15 seconds clips in the middle of each song is used. The surveyees are asked to label if they think the song they are listening to is computer generated or human composed. The result of 48 surveyees is summarized in Fig. 14. In the plot, each column represents a song, and higher the bar, more surveyees classified the song as human composed. In general, we see that the songs generated by Seq2Seq a comparable to the songs created by human. This can be seen from the fact that among the 3 songs that have more than 50% rating, 2 are generated by seq2seq. Of course, only 15 seconds of each song was revealed, and also the songs selected for Char RNN and Seq2Seq are among the best samples we came across, but it is still quite fascinating that the models were able to generate quite realistic music without prior musical knowledge at the start of training.

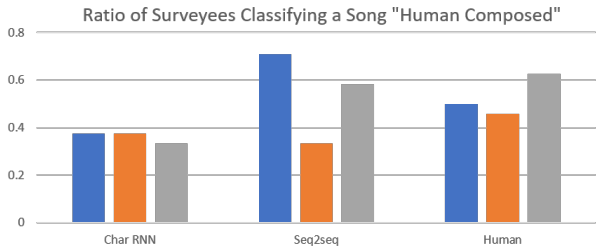


Figure 14: Survey Result

7 Conclusion and Future Work

With the advancements in artificial intelligence and machine learning, computers are now able to do tasks that were believed impossible before. And in many of these "impossible" fields, machines learn from scratch, without the knowledge of the governing rules. We hope that our project will be a small step in the right direction in the realm of machine music composition. Without having any prior concept of music, our models were able to learn to compose music that were both syntactically correct and also able to fool humans. We were also able to show that the flexibility of our model allows for exploring different parts of music composition, such as composing duet songs.

In the future, we would like to fix the GAN model so we can create even more musical and even more syntactically correct music. Also, we would like to extend the project by trying out different music notation formats, such as MusicXML, so we can have a more diverse dataset.

Our code can be found on our GitHub repository: `https://github.com/yinoue93/CS224N_proj`

References

[1] A. Karpathy, J. Johnson, and F. Li, "Visualizing and understanding recurrent networks," *CoRR*, vol. abs/1506.02078, 2015.

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[4] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014.

[5] B. L. Sturm, J. F. Santos, O. Ben-Tal, and I. Korshunova, "Music transcription modeling and composition using deep learning," *CoRR*, vol. abs/1604.08723, 2016.

[6] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016.

[7] C. Kereliuk, B. L. Sturm, and J. Larsen, "Deep learning and music adversaries," *IEEE Transactions on Multimedia*, vol. 17, no. 11, pp. 2059–2071, 2015.

[8] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," *arXiv preprint arXiv:1206.6392*, 2012.

[9] B. L. Sturm, ""lisl's stis": Recurrent neural networks for folk music generation," may 2015.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[11] A. Karpathy, J. Johnson, and F. Li, "Visualizing and understanding recurrent networks," *CoRR*, vol. abs/1506.02078, 2015.

[12] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, pp. 2226–2234, 2016.

[13] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization.," in *ICML*, pp. 1889–1897, 2015.