Harmony through Algorithms: Composing Music with Recurrent Neural Networks

Lian Gan
University of California San Diego
lgan@ucsd.edu

Jessica Song
University of California San Diego
jwsong@ucsd.edu

Zhirui (Raymond) Xia
University of California San Diego
z4xia@ucsd.edu

Boqi (Bobby) Zhu
University of California San Diego
b2zhu@ucsd.edu

Abstract

This report delves into the generation of music using a character-level Long Short-Term Memory (LSTM) network trained on a dataset of tunes in ABC notation. By employing deep learning techniques, specifically LSTM models, we aimed to capture the intricacies of musical structure and notation to autonomously generate new compositions. The model was trained with a focus on minimizing cross-entropy loss, aiming for a baseline performance, and tested under various conditions to explore the effects of temperature adjustments on the creativity and determinism of the output. Additionally, the study experimented with different network configurations, including variations in neuron counts and the implementation of dropout, to evaluate their impact on model efficacy and output diversity. The findings highlight the LSTM network's capability to learn and replicate musical patterns, offering insights into the potential of AI-assisted music composition and the critical role of hyperparameters in balancing innovation and coherence in generated music. Our key finding is that by tuning the hyper-parameter, we found that a number of 250 hidden neurons and a temperature of 1 in the softmax prediction lead to the best result. Also, the LSTM model tends to perform better and more robust compared to the RNN model in terms of test loss, scoring around a loss of 1.4, while RNN typically results in a test loss of 2.0.

1 Introduction

The task at hand, generating music using a character-level Long Short-Term Memory (LSTM) network trained on ABC notation, represents a cutting-edge intersection of technology and art. ABC notation, a simple text-based format for music notation, serves as the foundation for this exploration, allowing for the encoding of music in a manner that is both computationally accessible and human-readable. The importance of this task lies in its potential to democratize music composition, enabling both seasoned musicians and novices alike to explore musical creativity with the aid of AI. Music generation using deep learning techniques is not merely an academic curiosity but a significant step towards understanding how machines can learn, interpret, and recreate the complex patterns inherent in music.

The rationale behind employing LSTM networks for this task stems from their proven capability in handling sequence prediction problems. Unlike traditional neural networks, LSTMs possess the ability to remember long-term dependencies, making them particularly suited for the temporal and sequential nature of music. The character-level approach allows the model to learn the syntax and

Preprint. Under review.

structure of ABC notation from a granular perspective, enabling the generation of music one character at a time, with each character's prediction influenced by the context of all preceding characters.

2 Related Work

The foundational concept of Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997)6, introduced by Hochreiter and Schmidhuber in 1997, revolutionized the field of sequential data processing by addressing the vanishing gradient problem present in traditional recurrent neural networks (RNNs). Their innovation enabled the learning of long-term dependencies, a critical aspect for complex sequence generation tasks such as music composition. The introduction of gated units within LSTMs, including input, output, and forget gates, allows for the selective retention and discarding of information, making these networks particularly suited for capturing the nuanced dynamics of musical sequences.

Building upon Hochreiter and Schmidhuber's work, our research applies LSTM networks to music generation, extending their model with a novel approach to visualizing neuron activations. This technique not only enhances our understanding of how LSTMs process sequential data but also demonstrates the model's capability to generate coherent and melodically rich music sequences. By adjusting the temperature parameter in the LSTM's output layer, we further explore the model's flexibility, showcasing its potential for creative applications in music generation. Our work underscores the enduring relevance of Hochreiter and Schmidhuber's LSTM model as a foundation for ongoing advancements in deep learning and its applications to the arts.

3 Methods

3.1 Training network using Teacher forcing

3.1.1 LSTM Architecture

The LSTM model is structured with a single hidden layer the model is designed with 150 neurons in its hidden layer. This suggests a relatively simple yet capable architecture optimized for learning the sequential patterns found in the dataset.

3.1.2 Training Procedure

The training employs teacher forcing, where the ground-truth character at each time step is fed as the input to the next step, rather than using the model's prediction. This approach is evident in the loop where for each character in the sequence, the model predicts the next character, and the loss is accumulated over the sequence length.

3.1.3 Sequence Slicing

For each training iteration, a random slice of a sequence from the dataset is used, with a specified sequence length (SEQ-SIZE). This promotes variability and helps the model learn from different parts of the songs without relying on the sequence's start. At the beginning of processing each new song, the model's hidden state is re-initialized to zero, ensuring no carry-over state from the previous song, aligning with the stateless nature of each song's generation.

3.1.4 Loss Criterion

The model uses cross-entropy loss (nn.CrossEntropyLoss()) to measure the discrepancy between the predicted probability distribution over possible characters and the target distribution. This loss function is suitable for classification tasks like predicting the next character in a sequence, as it penalizes deviations from the actual character.

3.1.5 Optimizer

The Adam optimizer is utilized (optim.Adam(model.parameters(), lr=LR)) for adjusting the weights. Adam is chosen for its adaptive learning rate capabilities, which helps in converging faster by

automatically adjusting the learning rate based on the first and second moments of the gradients. The learning rate (LR) is a configurable parameter, allowing for flexibility in training speed and stability.

3.2 Song Generation

3.2.1 Priming the Network

To initiate the song generation process, our approach begins with "priming" the LSTM network with an initial sequence. This technique involves feeding a predefined string, <start>, to the model, effectively setting the initial context for the song generation. The choice of this priming string is crucial as it influences the direction and initial tone of the generated music. By starting with <start>, we signal the beginning of a new musical piece to the model, allowing it to predict subsequent characters based on the learned patterns from the training phase.

3.2.2 Incorporating Temperature in Generation

The temperature parameter (T) plays a pivotal role in controlling the diversity and creativity of the generated output. By adjusting the temperature, we can influence the randomness of character selection during the generation process. The function softmax-with-temperature is used to apply temperature scaling to the logits obtained from the model. A higher temperature results in a more uniform probability distribution, encouraging the model to make more varied and potentially more creative choices. Conversely, a lower temperature sharpens the probability distribution, making the model's output more deterministic and closely aligned with the most likely next characters as learned during training.

In our implementation, the temperature is set to 0.8 by default, offering a balance between variability and fidelity to the learned music patterns. This setting can be adjusted to explore the spectrum of outputs ranging from highly predictable to more experimental compositions.

3.2.3 Generating the Song

The song generation loop iterates up to a maximum length (max-len) or until an <end> token is encountered, signifying a natural conclusion to the music piece. At each step, the current character is fed into the model, and the output logits are processed through the temperature-scaled softmax to obtain a probability distribution over the next possible characters. A character is then sampled from this distribution using sample-char, which selects a character index based on the provided probabilities.

The generated character is appended to the growing song string, and this character is then used as the input for the next iteration, continuing the cycle. This process leverages the LSTM's ability to maintain a state across characters, allowing the generated music to exhibit coherent structural patterns that reflect the training data's underlying music theory.

3.2.4 Handling the End Token

The loop includes logic to detect the sequence <end>, allowing the generation process to conclude in a musically logical manner. Once the end token is detected or the maximum length is reached, the song is considered complete. This method ensures that generated pieces have a defined structure and length, mimicking the composition of real music pieces in ABC notation.

3.3 Hyper-parameter Tuning

By configuring the model-type parameter, both LSTM and RNN architectures were explored to determine their impact on the model's ability to capture long-term dependencies and generate music sequences. LSTMs are generally more capable of learning long-range dependencies due to their gating mechanisms but are more computationally intensive than basic RNNs.

The hidden-size parameter, which determines the dimensionality of the hidden state (and embeddings), was varied. Larger hidden sizes increase the model's capacity but also the risk of overfitting and computational cost. Experiments were conducted with different hidden sizes to find a balance between model complexity and performance. The num-layers parameter, specifying the number of recurrent

layers stacked in the model, was tuned to assess how increasing model depth affects learning and generalization. The dropout parameter was adjusted to identify an optimal dropout rate that effectively mitigates overfitting while allowing the model to retain its ability to learn from the training data.

The model's performance was evaluated after each round of hyperparameter adjustments, using metrics appropriate for sequence generation tasks (e.g., cross-entropy loss, qualitative assessment of generated sequences). This iterative process of evaluation and adjustment continued until the model achieved satisfactory performance.

3.4 Feature Evaluation

3.4.1 Importance of Feature Evaluation

Feature evaluation is an essential aspect of understanding and improving machine learning models, especially for complex tasks like sequence generation, where the model's decision-making process can be opaque. It provides deep insights into how the model processes its inputs and which features (e.g., characters in a sequence) are most influential in determining the model's outputs. This understanding is vital for diagnosing the model's strengths and weaknesses, ensuring that it correctly learns the underlying patterns rather than memorizing the training data or being misled by irrelevant features. Furthermore, many machine learning models, particularly deep learning models, are often criticized for being "black boxes" due to their complex internal mechanisms. Feature evaluation helps demystify these models by highlighting how different inputs influence the model's predictions, thereby improving transparency and trust among users and stakeholders.

3.4.2 Approach to Generating Heatmaps of Neuron Activations

The first step involved performing a forward pass of the generated music sequence through the model, specifically designed to capture the activations of neurons in the hidden layer(s) in response to each character. This was achieved by modifying the forward method to not only return the output predictions but also the raw activations (rnn-out) from the recurrent layer.

The extracted activations are appended with 0.0 at the end. This step might serve as a delimiter or simply ensure consistent array lengths for subsequent operations, although it's not a normalization step. The pad function is applied to both the generated song and the extracted neuron activations (data). This padding ensures that the activations and their corresponding characters are structured into a uniform grid for visualization. The pad factor determines the granularity of this grid, affecting the heatmap's layout. Both the padded song and padded activations are reshaped into a 2D matrix format, where each row corresponds to a segment of the song determined by the pad factor. This restructuring is crucial for aligning the activations with their corresponding characters in the heatmap.

For each neuron selected for analysis, a heatmap was generated using visualization libraries such as Matplotlib in Python. The heatmap depicted the sequence of characters along one axis and the activation level of the neuron in response to each character along the other axis. The color intensity within the heatmap indicated the strength of the neuron's activation, providing a visual representation of how the neuron responded to different parts of the music sequence. The heatmap is annotated with the corresponding song characters (if show-values is a function that does this), adding a layer of interpretability by directly correlating neuron activations with specific characters.

4 Results

4.1 Baseline LSTM

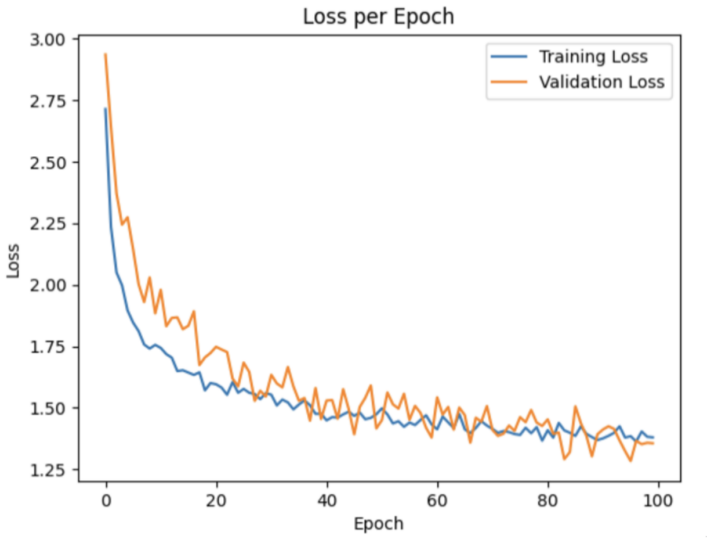

Figure 1: LSTM Baseline Loss Plot

This is our baseline model with 150 Neurons. Surprisingly enough, we were able to reach both a training and validation loss below 1.5, way below the report specified 1.9. Our loss plot also shows that our model is not overfitting, as our performance on validation set is similar to the training loss.

4.2 Temperature Experiments

Out of three experiments of temperature, we chose T=1 as our best result judged by all group members. It features very smooth connection between notes and a melody. We show the ABC notation and music representation in 2 and 3, respectively. Lastly, we also show the loss plot when T=1 in 4, showcasing that we reached a even smaller train and validation loss than before, showing that T=1 is a good parameter for our LSTM training.

<start>>

X:1

T:Sechin Seache

K:D

|: B>c de/e/ | c2 Be | ce d2 | Be fe/e/ | ed e>f|ge e2 | f2 dB | A/A/ G/A/B/c/ | df/e/ dc/d/ | eG/A/ BA B/ B/A/B/A/ | B2 ec | dc Be:|

|:Bc BB/B/|dB BA | fe c/e/ |1 d4 d2 g2 ||

d2 e/d/c/B/ |

AE Gd/d/ | ef ga|fd fd/e/f/e/ df/e/ | B>B B/d/e/f/ | g2 dB | B/A/B/f/ eg/e/ | d/B/A/B/ dB | BB B2 LBE EF |1 D2 D2 :|2 E>A GA | Bc d>f | ed c>c | B2 Bd | f/e/f/d/ ^fc | f>d eg/e/ ||

|: dA d>d | cd/e/ f/2g/2 f/g/a |

w:|

|:ef/e/ g/f/e/f/ | ed c>c | e/g/c/B/ AG :|

K:C

Ge e/f/e/d/|cA cA|cA AB|ef ed|B/f/e/f/ d>c|BA FA|A>B ce|e>f ed|e>d ef|ga bg|ge/e/ dB|cB BA|Be f>e|dB B/d/c|dB B/B/c/B/|1 Ac fe:|2 fe ed|Bd B/c/d|ed B/c/d|ef gc|a2 f/g/d/e/|a/g/e/f/ ge|fd/f/ ge|dd d/e/c/B/|Ad B | AB B/f/e/d/B|BA/B/ AG|FA B/e/c/A/|G>A BG|E>F/G/ AB|=cd c>B|AF EF|E>F G/A/B/c/|dB AF/G/|A>d d>e|e/f/e/f/ ec|1 dA FA:|2 AG G>A||

|:d2 dc|B/A/G/A/ BA|BA A2|]

<end>

Figure 2: ABC Annotation for $T = 1$

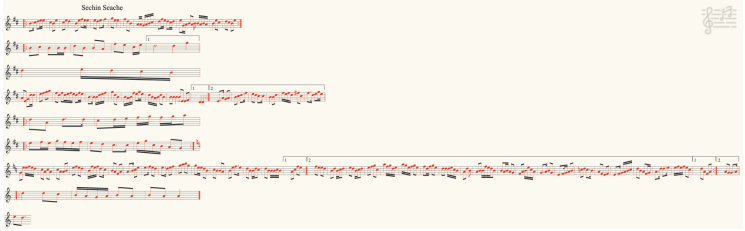

Figure 3: Music Representation for $T = 1$

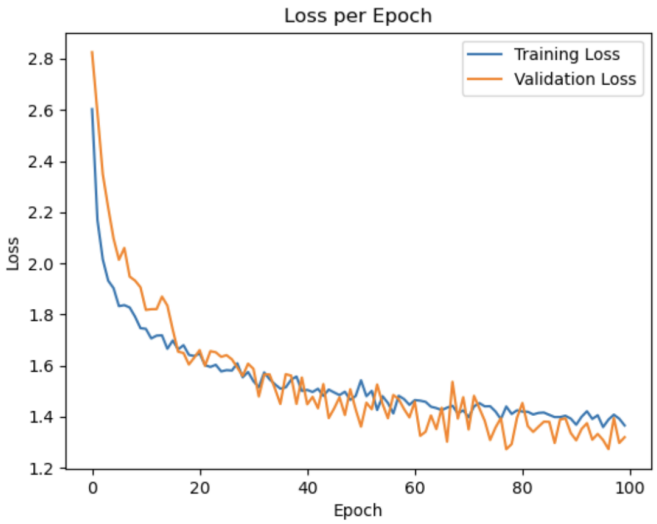

Figure 4: Loss Plot, T=1

Other than T=1, we also experimented with T=2 and T=0.5. 5, 6 and 7 are from the temperature=0.5 case.

<start>>

X:1

T:Le Reevy Boy a Magan Liver

R:polka

D:Michel BELLON - 2005-06-11-25

Z:Pour toute observation mailto:galouvielle@free.fr

M:2/4

L:1/8

K:A

A>B AG|FA A>B|AG FE|D2 DE|FD D>E|FD D2:|

|:d>e fe|f/e/d Bd|ef gf/e/|dc BA|d>e f/e/d|BA AB|ce fe|1 df d2:|2 fd ed||

|:ed BA|d2 f>e|dc BA|d2 e>d|cB AB|cB AB|cd ed|BA AB|cB AB|AB AB|cd ed|BA AB|cB BA|Bd BA|BA AB|cd ef|ed BA|1 BA G>A:|2 A2 A2||

|:a f/e/d|cA AB/c/|dc BA|FA AF|AB AF|A2 AB|ce g>f|ed B>A|Bd ed|ef g>e|dB BA|BA B/c/d|BA AG|B/c/d e>d|Bc BA|G>A BA|de f>e|dB AG|A>A AB|cB AB|c/B/A/B/ AF|AF AF|AF FA|cA AB|AB AB|cB AG|FA AB|cA BA|eA AB|cA eA|fA AB|cA AB|cd ed|cA AB|AB cB|AB/c/ BA|Bd BA|1 G2 G>A:|2 A2 AB||

|:ed d>e|ff f>f|ef gf|ec AG|AB c/d/c|BA AB|de f/e/d|cA BA|eA AB|ce e>c|BA FA|G>A BA|1 A2 A2:|2 ed A2||

|:ea e/f/e/d/|ef gf|ef g/f/e/d/|fd ef|ed BA|de/f/ gf|ec AB|cB AG|FA AB|cA AB|cB AB|ce e>f|ge dB|E2 E2||

|:ed B/c/d|ed de|fd ef|ec AF|A2 AB/A/|GE FE|D>E FD|A>B AB|cB cB|AF DF|AB/A/ BA|BA F2|cB AB|cA F>A|BA FA|1 BA G2:|2 G2 G>A||

|:Be e/f/g/e/|

Figure 5: ABC Annotation for $T = 0.5$

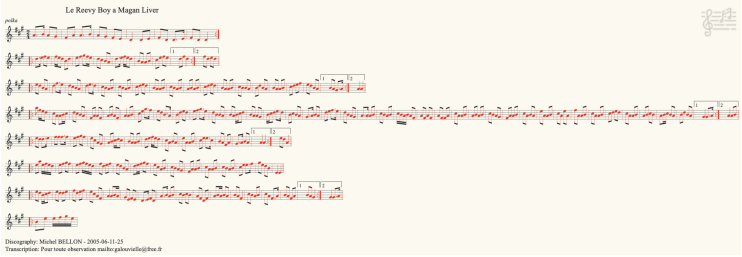

Figure 6: Music Representation for $T = 0.5$

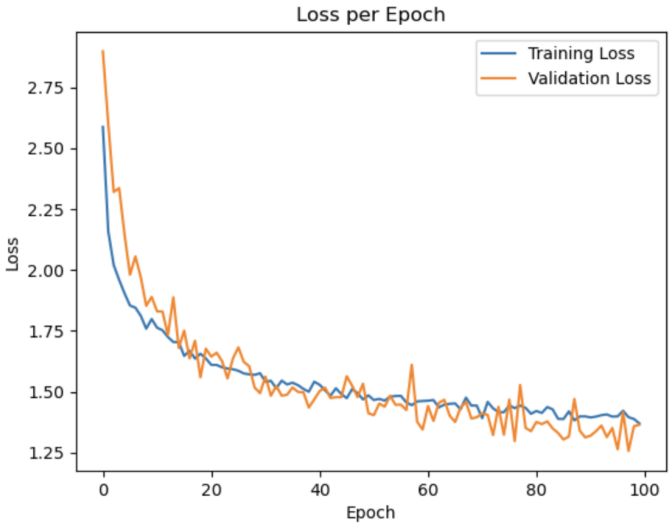

Figure 7: Loss Plot, T=0.5

Lastly, 8, 9 and 10 are from the temperature=2 case.

```
<start>etedanFit|e3 adfrinGy Cosee

Z:<adJe

~B3B|cBBfec|feAB dcBA|(3A=FAFEE),|C2EDdCBB^G4)F/B|d2A2de|1A2 -|[c]fdeg|asA)b

S:3ffb |1 bf/c/A aEDb (3fed/d^ ADnd} g(?abt Skpt~alt

AA\

P:ig

Z:id:hn-polkb5h>

S:CorDpy E'-1y, J B,BKalyl

hEB]B2|d^cde gBdG|f2gf g3e|(f4fbge

eC a>bwg"gfe|"B77"A1f2zd:

H):Ah M:'ivaritre

"K:2

Fd hD|d2^c, EGEE FGCF|E^cGE F>BBc|c^d^c BFAGM6 B/A | O5y, Pspa) Pa-Lmgape-3:e)

C:L'e8e>e ess, <ddbvays

a|de dc|^d>c _e/d\:~D|

(3C:

X:67

T: 6#7:EFRe

O:Frfefeothf"Gf(Vafbkst, pt)niea

R:BArntan

Z:S/9ivd-8

M:4/8|deg, Be=9zBm/Sl)\ipe

f:-11
```

Figure 8: ABC Annotation for $T = 2$

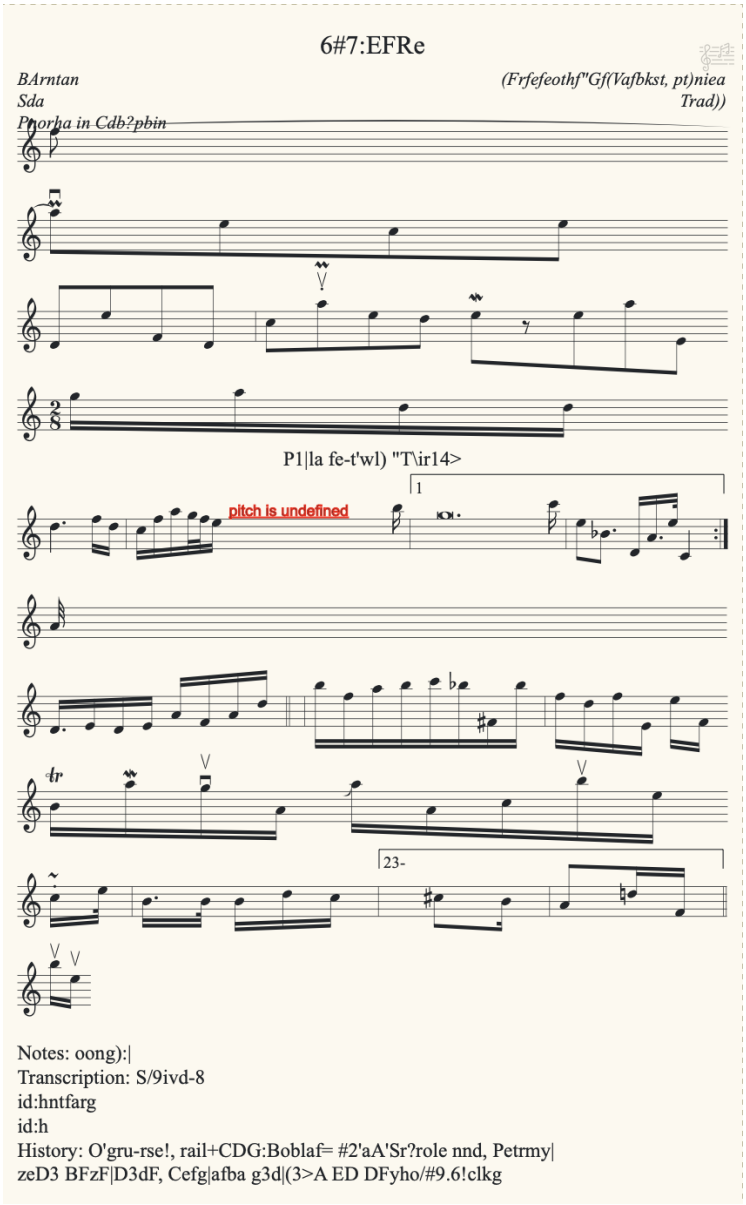

Figure 9: Music Representation for $T = 2$

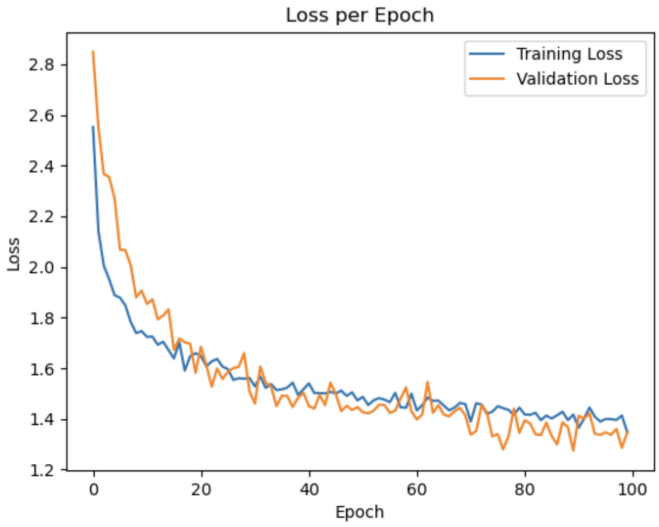

Figure 10: Loss Plot, T=2

For the tuning of the temperature parameter, we're mainly adjusting the strictness and the action of the softmax function in the generate function, where we calculate the probabilities of the output belonging to each character and sample the predicted character based on the character likelihoods.

When the temperature is set to 1, the softmax function behaves normally, without any scaling effect on the input vector. This setting creates standard softmax calculations where the raw inputs are directly used to compute the probabilities, which, in our case, created the best result.

For a high temperature ($T > 1$), increasing the temperature makes the softmax output more uniform (i.e., the probabilities become closer to each other). Higher temperature values increase the denominator in the softmax function, leading to smaller differences between the output probabilities. Therefore, we are more likely to get different notes and settings from our training set, and a more generic song. In our result, we got a slower tempo, similar notes, and a song that sounded like someone was randomly pressing on pianno keys. This showcases how this temperature parameter indeed made our song more different from our training set.

For a low temperature ($T < 1$), the softmax output more extreme or "sharper", with higher probabilities assigned to the larger inputs and smaller probabilities approaching zero. This effect increases the confidence in the predictions, making the output distribution more peaky. Therefore, we predict the more likely notes extremely confidently. Our song has thus sounds more similar to the training song.

4.3 Loss plots of hyperparameter tuning experiments

4.3.1 RNN vs LSTM

In our comparison between two groups of RNN and LSTM, the first group features a hidden size of 200 with a dropout probability of 0.1, and the second has a hidden size of 250 with a dropout probability of 0.2. We wanted to use two different set up so that we ensure that the difference we see between two models isn't due to other factors, but purely based on the different behaviors of both models.

For both settings, the LSTM's validation loss curve aligns more closely with its training loss curve. Conversely, the RNN's validation loss curve shows greater fluctuations and stands at a higher level than its training loss curve. This suggests that LSTMs offer more stability and are likely to yield more consistent results. Moreover, LSTM converges at the same speed on both training and validation sets. At last, LSTM produces a very low training and validation loss, around 1.3 for both hidden size of 200 with a dropout probability of 0.1, and the second has a hidden size of 250 with a dropout probability of 0.2.

On the other hand, RNN tends to have a lower starting loss for training set. Its validation loss often starts at a much higher starting point when compared to the training loss. Moreover, RNN converges really fast (within 10 epochs) to a relatively stable training loss, and then lowly fluctuates around the same loss. In comparison, its validation loss often fails to converge that fast or reach a similar number as the training loss. In the end, RNN displays a significantly higher training and validation loss, with training loss around 1.8 and validation loss fluctuating greatly, around 2.0. This shows us that RNN fails to guarantee a low validation loss, unlike LSTM.

Overall, we can see that LSTM performs much better with a similar training and validation loss, less fluctuations with epochs, and a significantly smaller loss (around 1.3).

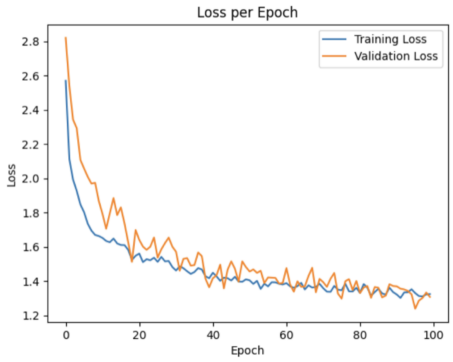

Figure 11: LSTM, hidden size 200, dropout 0.1

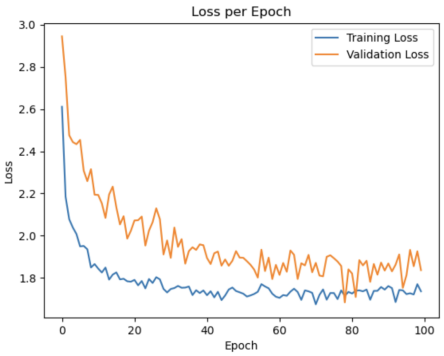

Figure 12: RNN, hidden size 200, dropout 0.1

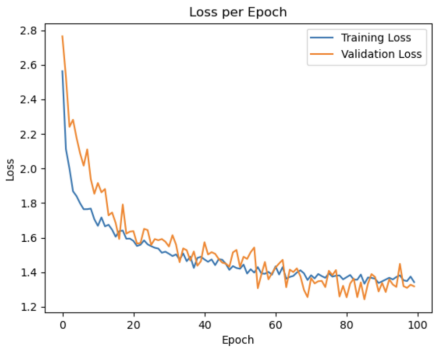

Figure 13: LSTM, hidden size 250, dropout 0.2

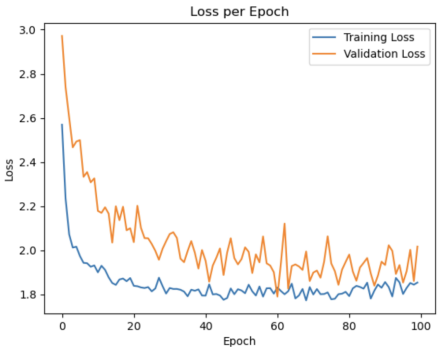

Figure 14: RNN, hidden size 250, dropout 0.2

4.3.2 Changing number of neurons in the hidden layer

When adding more neurons to the hidden layers of LSTM and RNN models, the improvement in model performance might be marginal, evidenced by only a slight reduction in loss. This situation can be attributed to overfitting and data constraints.

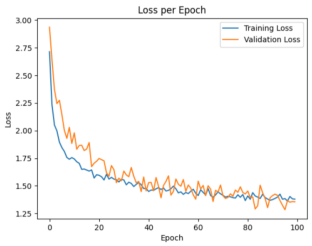

Figure 15: LSTM, hidden size 150

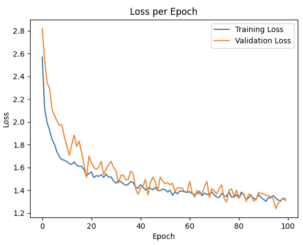

Figure 16: LSTM, hidden size 200

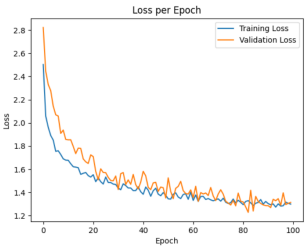

Figure 17: LSTM, hidden size 250

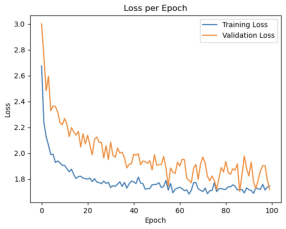

Figure 18: RNN, hidden size 150

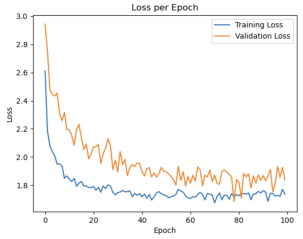

Figure 19: RNN, hidden size 200

Figure 20: RNN, hidden size 250

4.3.3 Varying dropouts

We performed experiments through varying dropout probability from 0.1 to 0.3 on both RNN and LSTM. For RNN, a dropout probability of 0.1 lead to the smallest training loss, while dropout probaility of 0.3 has led to the smallest validation loss. We can tell that a higher dropout probability slightly improves our models ability to generalize. However, the difference is not significant.

On the other hand, varying drop out probabilities for LSTM have to led to any changes on our model's performance in both training and validation data.

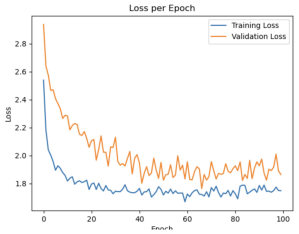

Figure 21: RNN, Dropout 0.1

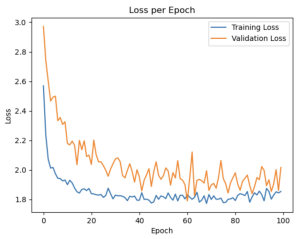

Figure 22: RNN, Dropout 0.2

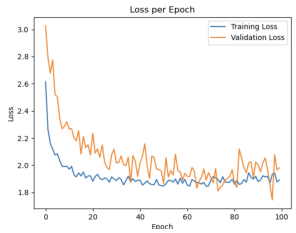

Figure 23: RNN, Dropout 0.3

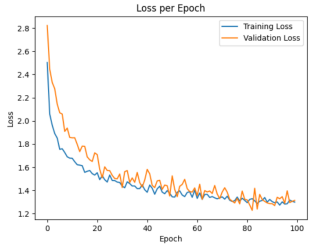

Figure 24: LSTM, Dropout 0.1

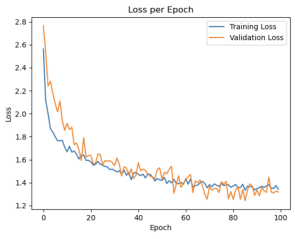

Figure 25: LSTM, Dropout 0.2

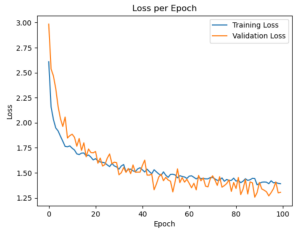

Figure 26: LSTM, Dropout 0.3

4.4 Heatmap

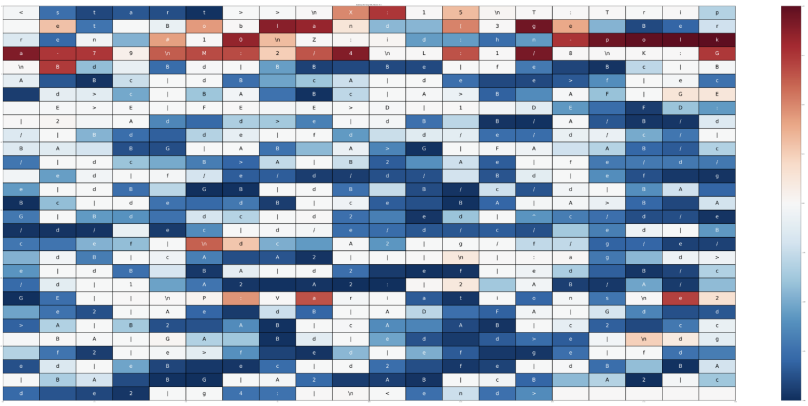

Figure 27: Heatmap for the first neuron

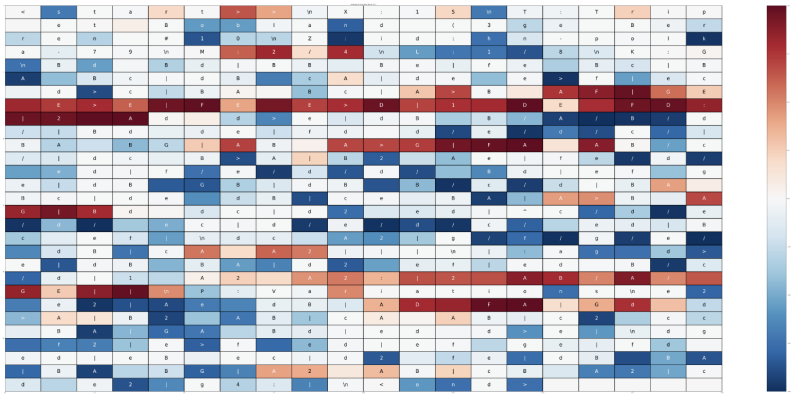

Figure 28: Heatmap for the second neuron

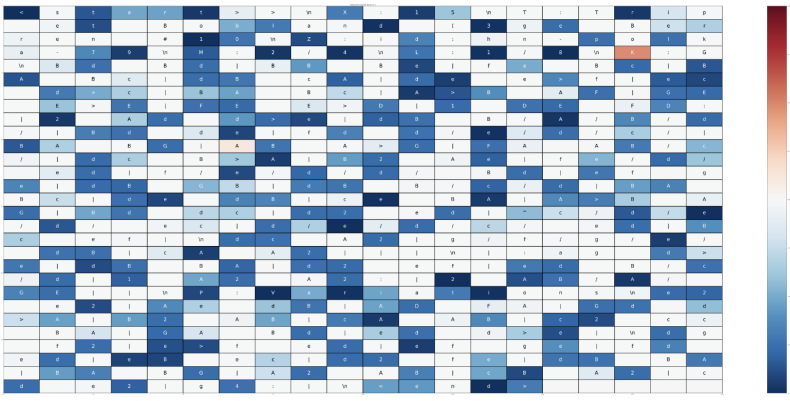

Figure 29: Heatmap for the third neuron

5 Discussion

Our findings underscore significant differences in performance between RNN and LSTM models, both qualitatively and quantitatively. The LSTM models consistently outperformed the RNNs in terms of loss metrics, achieving around 1.4 for both training and validation loss (while RNNs achieve 1.8 and 2.0 for training and validation loss, respectively). This indicates that our LSTM featured a more efficient learning process and better generalization to unseen data. Qualitatively, music generated by LSTM models exhibited a more coherent and melodically pleasing structure, likely due to LSTMs' superior ability to capture long-term dependencies in musical sequences. In contrast, RNN-generated music often lacked this coherence, with melodies appearing more fragmented and less structured. There would also be occasional pauses or sudden changes in tempo from RNN-generated songs, indicating a worse qualitative performance.

Increasing the number of neurons in the hidden layers of both RNN and LSTM models showed only marginal improvements in performance, evidenced by slight reductions in loss. This phenomenon suggests that for our particular small dataset and song generation task, our model already performs well enough with a hidden size around 150. Therefore, an increase in the number of neurons doesn't necessarily equate to significant gains in learning or generalization. If we were to have a lot more training data and/or were required to generate a much longer or sophisticated song, then this variation in number of neurons in the hidden layers can create a bigger difference.

Varying dropout rates had a nuanced effect on our models. While dropout is intended to mitigate overfitting by introducing regularization, our model with a dropout probability of 0.1 already featured a small difference between training and validation loss. It is evident that, on the RNN models, where the training loss and the validation loss had a relatively more significant difference (1.8 vs. 2.0), the higher dropout rate was effective in terms of reducing the validation loss. Yet, overall, experiments did not show a clear pattern of improvement in model performance with changes in dropout rates. We believe that this is because that our model was an over fitting originally, and that we have a relatively small training dataset and network, so the neural network wasn't complicated enough to require dropouts in the first place.

The first Neuron exhibits heightened activation in response to the music's header but shows reduced activity during the body of the piece. This pattern suggests that the neuron has learned to recognize and respond to structural markers that denote the beginning of a music piece, such as key signatures, tempo indications, or specific introductory notes. The second neuron demonstrates increased activation during the body of the music, where the melodic and rhythmic complexity unfolds, but less so for the header. This indicates its specialization in processing the musical content beyond the structural cues, focusing on elements that define the piece's musical narrative and expression. The last neuron displays low activation levels for both the header and the body of the music, this neuron suggests a more generalized or subdued response to the input sequence. This behavior might imply that the neuron serves a different role, possibly as part of the network's mechanism for integrating information over longer stretches of music or contributing to the overall robustness of the model by not overfitting specific features.

The differentiation in neuronal responses underscores the network's capacity to specialize, with some neurons becoming attuned to structural aspects of music, while others focus on the dynamic content within the music's body. Such specialization enhances the model's ability to grasp the multifaceted nature of music, from form and structure to melody and rhythm.

6 Authors' Contributions and References

Jessica Song: I wrote the `generate_song` function in `generate.py`. I also helped debugged the whole process when we were intially having trouble with reaching a loss of 1.9 on the LSTM baseline model through going to office hours and making a Piazza post. I ran a couple of experiments between different numbers of neurons, and I interpreted our experimentation results in Results and Discussion.

Bobby Zhu: My main contribution is tuning the model for RNN, and LSTM, I run different sets (15 different configs) of hyperparameters and find the best model, and save all loss plots, heatmaps, and generate songs for later reports. I also contributed to the loss function visualization, I fixed a bug of

mismatch between loss data and loss plot. Regarding the report, I did the following part alone: title, abstract, introduction, methods, 6d, and the discussion for three different heatmaps.

Lian Gan: I tuned the model for RNN, and LSTM, I run models of hyperparameters for report and find the best model to generate songsadn heatmaps. I fixed some bug when working on a GPU system. For the report, I did the following part alone: Related Works, result and discussion.

Zhirui Xia: I was in charge of the coding aspects of our project, developing and training the LSTM (RNN) architecture to understand ABC notation in music files. I evaluated the model's performance after each training epoch to ensure the baseline model achieved a loss of 1.9 on the validation set. On top of that, I ensure the code is bug-free, and can output correct loss curve and generate heat maps for later interpretation and fine-tuning.

Reference:

Sepp Hochreiter, Jurgen Schmidhuber (1997) *LONG SHORT-TERM MEMORY* https://www.bioinf.jku.at/publications/older/2604.pdf