Character Level Music Generator

Jack Kai Lim
Halıcıoğlu Data Science Institute
University of California, San Diego
jklim@ucsd.edu

Vivian Chen
Halıcıoğlu Data Science Institute
University of California, San Diego
vnchen@ucsd.edu

Hou Wan
Halıcıoğlu Data Science Institute
University of California, San Diego
hwan@ucsd.edu

Elsie Wang
Halıcıoğlu Data Science Institute
University of California, San Diego
e2wang@ucsd.edu

Abstract

This project explores the capabilities of Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks in generating music to deepen our understanding of these models' mechanics. Focusing on character-level generation of music in ABC notation, we compare the performance of RNN and LSTM models to identify their respective strengths and weaknesses in this context. We optimized our best-performing model with a Dropout probability of 0.2 and 150 hidden LSTM layers, achieving a validation loss of 1.469. This result demonstrates the potential of LSTM models in accurately generating complex music sequences, offering insights into the nuanced differences between RNN and LSTM architectures.

1 Introduction

The task we are trying to tackle is a Character Level Music Generator using Long-Short Term Memory (LSTMs). We will also be running an regular Reccurent Neural Network(RNN) to comparing the 2 different models in their performance. We are using ABC notation and we are using this website which will allow us to convert the ABC notation music generated by our Deep Learning model into a playable .midi format.

1.1 ABC Notation

ABC notation is a way to translate musical notes into text data which will allow us to build a model and train it to be able to generate and make music. ABC notation is written in the follow way,

1.2 LSTMs

LSTMs are a type of RNN which are usually used for time-series analysis which addresses the RNN's long-term dependency issues. Due to RNN's design, which has information flowing through the network in a loop-like fashion, they usually fail to maintain information over long sequences due to the vanishing gradient problem. LSTMs overcome this problem by incorporating a mechanism that controls the flow of information by having an internal memory, an gates to control the flow of the information.

The components of the LSTMs that do these are, the **cell state**, the gates: **Forget Gate**, the **Input Gate**, the **Cell State** and the **Output Gate**, and the **hidden state**. These unique to LSTMs components use the learned weights by the LSTM to regulate the flow of the information through the network, allowing it to remember and forget patterns over long sequences of data. Specifically, the cell state is a

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

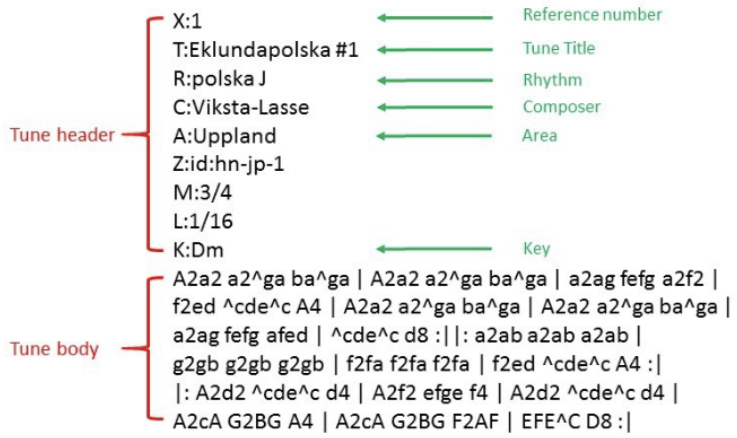

Figure 1: ABC Notation

long-term memory that carries information across time steps, the gates control the flow of information, and the hidden state carries relevant information to the next time step.

2 Related Work

We got inspired to structure our RNN and LSTM architecture by the book, Dive Into Deep Learning Zhang et al. [2023].

3 Methods

3.1 Teacher Forcing

For our baseline model, we used an LSTM architecture with one hidden layer for music generation. The model structure includes an embedding layer for character index conversion, a recurrent layer of either LSTM or RNN, a linear output layer, and a dropout layer. The training of the model employs teacher forcing as a strategy to accelerate convergence and improve stability. In teacher forcing, the true previous output token is used as the next input to the model during training, rather than the model's prediction from the previous time step. At the beginning of each sequence, the hidden states of the LSTM layers are initialized to zero to avoid interference from previously processed sequences. Training sequences are randomly sampled from the dataset, and each token in a sequence is fed into the model sequentially. The embedding layer processes each token, followed by the LSTM layers and the output layer to predict the next token. For our optimizer, the Adam optimizer was utilized for its adaptive learning rate capabilities, which help in converging faster and more effectively compared to standard SGD. The learning rate was set to 1e-3. For loss, we used cross entropy due to its effectiveness in classification tasks, including predicting the next token in a sequence, where the task can be considered classification over the vocabulary. This loss was accumulated over all tokens in the sequence to form the total loss.

3.2 Song Generation

First we primed the model by looping through each character in **<start>**, then we looped **max_len** number of times, which essentially gives us the length of the text to generate. During each loop we feed back the last output from the model, which then generates the next character as part of our generated song. For selected which character to append to the generated song, we first got all the probability for all the characters from the softmax output of the model, then we applied a temperature scaling. Pseudo Code for the generation is as follows:

```
generated_song = "<start>" # To store generated song
```

```
for c in "<start>" # Priming the model
    model.forward(c)

for _ in range(max_len)
    probs = model.forward(last_outputed_char)
    <temperature scale the probs>
    <get highest prob from tprobs>
    generated_song += highest_prob_char

    break when "<end>" is outputted or max len reached
```

3.2.1 Temperature Scaling

We applied a temperature scaling which is a way to add some randomness when generating sequence. We did this as if we always only chose the most probable output from the softmax outputs, this would make it so that when the model see for example the character "A" it will make it extremely likely to generate the same next character which would not be good for diversity and creativeness of the model. This is done by dividing all the logits from the softmax output by T which the value of the temperature set during generation. Then the logits are all exponentiated.

The effect of temperature is as follows, for a **High Temperature** $T > 1$ this increases the randomness of the output distribution and makes it more likely to sample less likely characters from the softmax output. Which would make the generation more diverse and 'creative'. But if it is too high it could also start outputting nonsensical outputs and random generations. On the flip side for a **Low Temperature** $T < 1$ this makes the outputs more deterministic, and makes it more likely to choose the probable characters instead. So when T tends to zero, it will always choose the most likely outcomes.

3.3 Hyper-parameter Tuning

The RNN architecture that was used in our model was an embedding layer for character index conversion, recurrent layer, linear output layer, and a dropout layer. The approaches we took in tuning our hyper parameters was changing the number of neurons in the hidden layer with 150 (baseline), 200, and 250 neurons, and we used dropout with rate = 0.1 (baseline), 0.2, and 0.3, and we did this by creating new config files.

3.4 Feature Evaluation

Furthermore, we used feature evaluation to evaluate our model. In the context of music generation, feature evaluation is important for understanding and interpreting the behaviour. Specifically, it gives insight into which features or patterns within the input data are being recognized by certain neurons in the network. This helps with improving the model, finding any biases, and interpretability. In feature evaluation, we performed forward propagation on generated music samples and examined the activation of each neuron for every character and plotted them on heatmaps, aiding in identifying neurons that specialize in recognizing a specific feature or musical element of the composition created.

4 Results

4.1 Baseline

For the baseline model we ran a LSTM with 150 hidden neurons for 100 epochs and obtain the loss plot seen here here.

The other hyper parameters that we used which are going to stay the same unless stated throughout are

```
Learning rate: 1e-3
Sequence Size: 30
No_Epochs: 100
```

```
No_layers: 1
Temperature: 0.8
max_generation_length: 1000
```

Training Epoch: 99, 100.00% iter: 801 Loss: 1.3351
Validation Epoch: 99, 100.00% iter: 151 Loss: 1.591

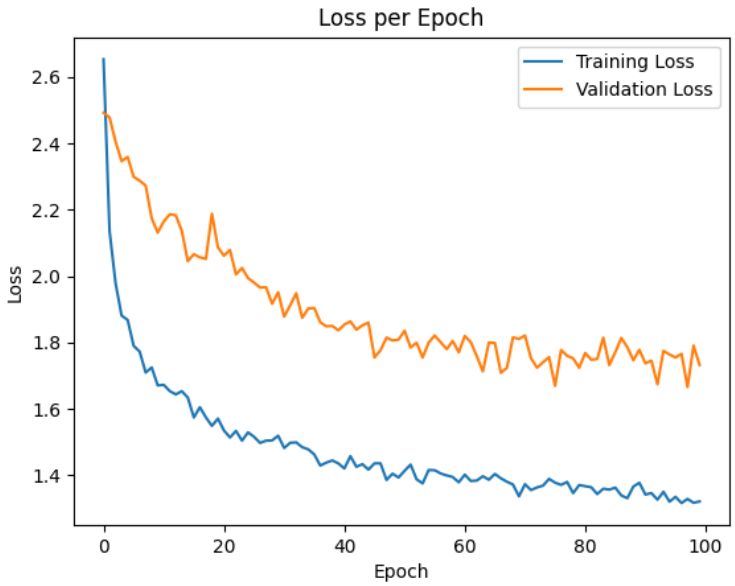

Figure 2: Baseline LSTM Loss Plot

4.2 Generated Music with 3 different T values

All music sheets are in the appendix of this document

4.2.1 T = 1

Here is the music sheet generated by the model for when temperature equal 1. Which implies that the logits were not tampered with and the generation made a random choice based of the softmax layers output probabilities.

4.2.2 T = 0.5

Here is the music sheet generated when temperature is 0.5. Which makes the generation more deterministic and chooses the outputs which the higher probabilities with a higher likelihood from the softmax output

4.2.3 T = 2

Here is the music sheet generated when temperature is 2. This makes the model generate more randomly, and allows it to choose outputs that would otherwise be more improbable, from the softmax output layer.

4.3 RNNs vs LSTM

Using the same hyper-parameters as the baseline LSTM model, we trained an RNN model for the same The plot for the loss of the RNN model can be found here.

Training Epoch: 99, 100.00% iter: 801 Loss: 1.481
Validation Epoch: 99, 100.00% iter: 151 Loss: 1.891

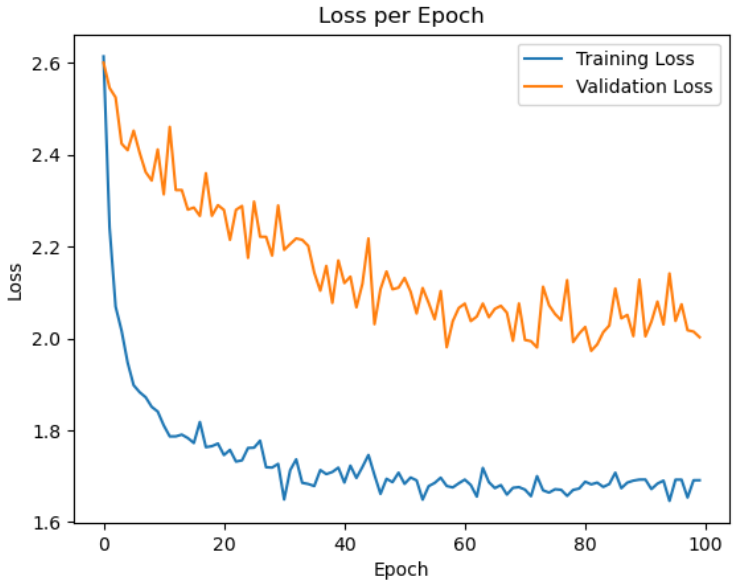

Figure 3: RNN Loss Plot

4.4 Different Number of Neurons

4.4.1 200 Neurons

To test how different number of hidden neurons would affect the model, we first tried a LSTM with 200 hidden neurons with the same hyper parameters as the baseline. The plot for the loss can be found here.

Training Epoch: 99, 100.00% iter: 801 Loss: 0.7513
Validation Epoch: 99, 100.00% iter: 151 Loss: 1.842

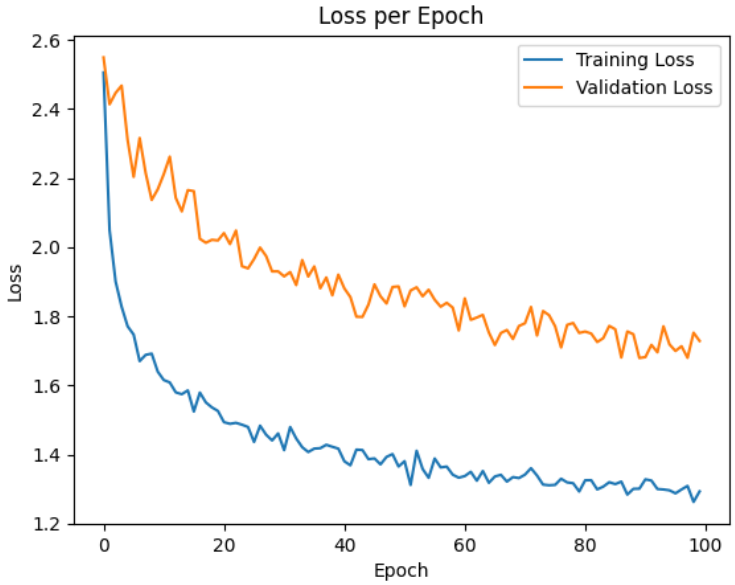

Figure 4: 200 Hidden Neurons Loss Plot

4.4.2 250 Neurons

In this training example, we trained with the same hyper parameters as the baseline but with 250 hidden neurons. The loss plot is here.

Training Epoch: 99, 100.00% iter: 801 Loss: 1.227
Validation Epoch: 99, 100.00% iter: 151 Loss: 1.485

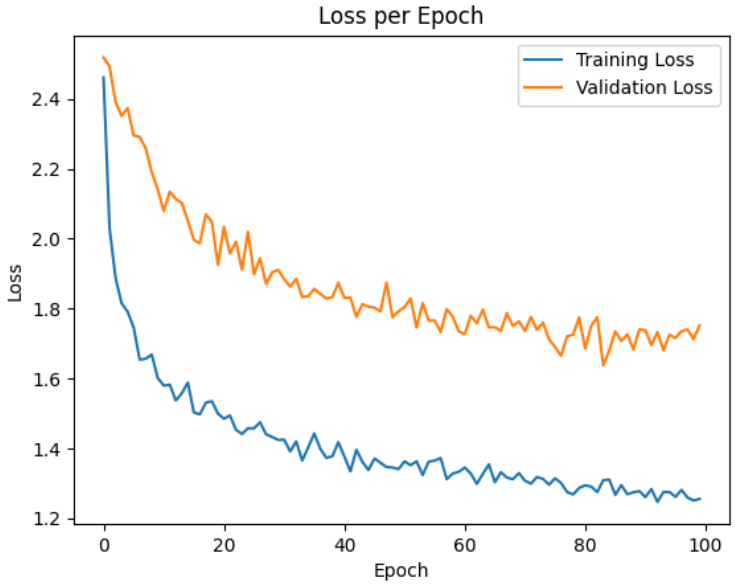

Figure 5: 250 Hidden Neurons Loss Plot

4.5 Varying Dropout

4.5.1 Dropout = 0.2

To test how varying dropout affects the model we also ran one with the same hyper parameters as the baseline but with a 0.2 dropout instead. The plot can be found here.

Training Epoch: 99, 100.00% iter: 801 Loss: 1.8276
Validation Epoch: 99, 100.00% iter: 151 Loss: 1.469

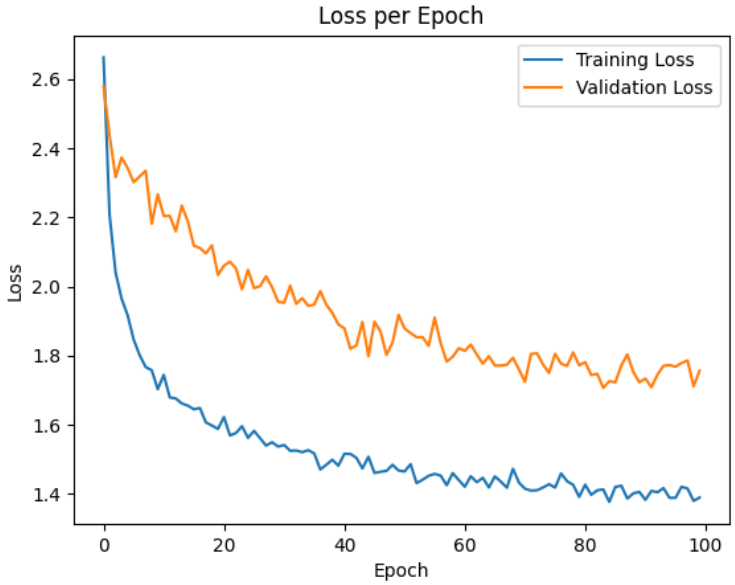

Figure 6: 0.2 Dropout Loss Plot

4.5.2 Dropout = 0.3

Additionally, we ran with the same hyper parameters as baseline but with a 0.3 dropout instead The plot can be found here.

Training Epoch: 99, 100.00% iter: 801 Loss: 1.226
Validation Epoch: 99, 100.00% iter: 151 Loss: 1.488

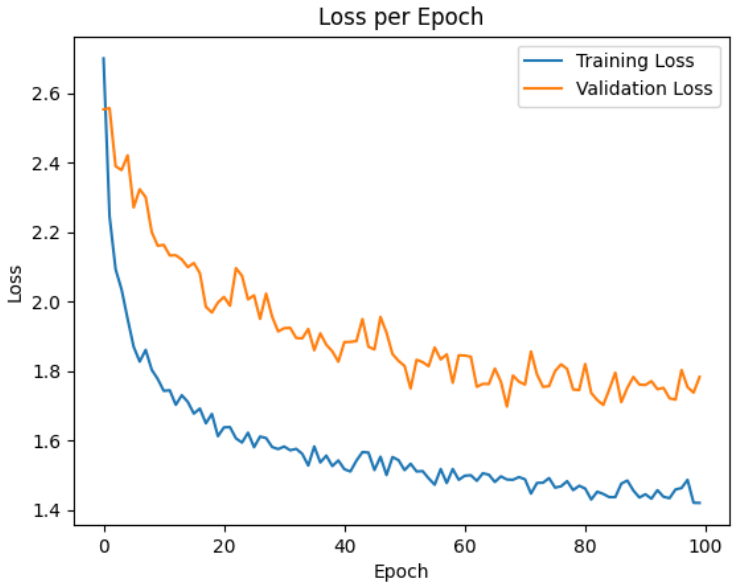

Figure 7: 0.3 Dropout Loss Plot

4.6 Heatmap For Best Model

The different heatmaps for the neurons are seen to be highly activating to different components of the generated music. For our best performing model, which is the LSTM network with a Dropout probability of 0.2 and 150 hidden layers, we get clear representations of the concepts that neurons 0, 10 and 50 activate highly on.

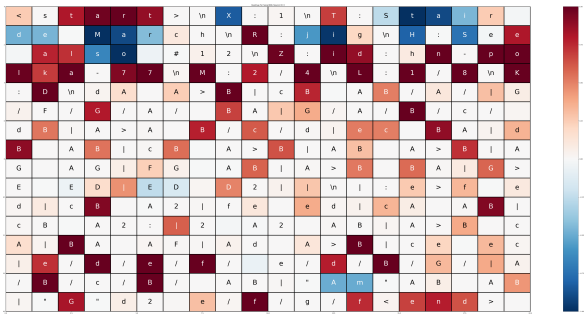

Figure 8: Neuron 0 heatmap

As seen in this heatmap, we can see that neuron 0 activates strongly at the beginning of the text sequence with the "<start>" header, as well as the end of the sequence with the "<end>" header so the neuron could possibly be interpreted as the headers of the sequence.

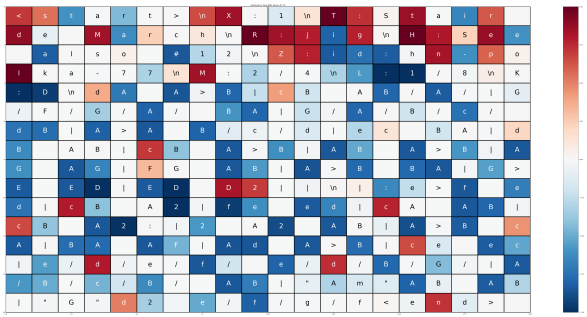

Figure 9: Neuron 10 heatmap

Neuron 10 has an interesting concept: It responds particularly highly to the title of the music, and more specifically the capital letters of each word within the titles.

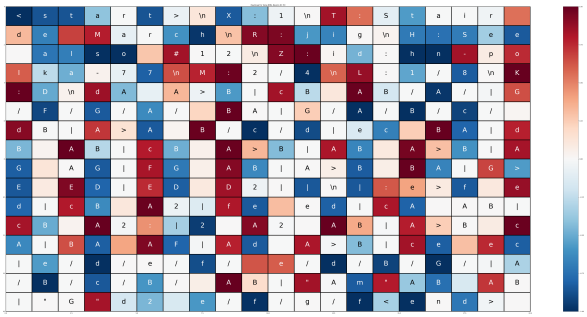

Figure 10: Neuron 50 heatmap

Finally Neuron 50 is seen to activate highly in the body of the music, which consist of many varying notes. This neuron activates all over the sequence, and likely represents the generating of notes throughout the sequence.

5 Discussion

The difference between RNN architecture and LSTM architecture is that LSTMs address the limitations of RNNs by incorporating specialized mechanisms that enable them to capture and use long-term dependencies in sequences more effectively. Since LSTMs are more complex than RNNs, the validation loss of the LSTM model (1.591) did end up being lower than the validation loss of the RNNs model (1.891). We also changed the number of neurons in the hidden layer. Specifically, besides the 150 used in the baseline, we also tried using 200 and 250 neurons in the hidden layer. Using 200 neurons had higher validation loss (1.842) than the baseline, but the 250 neurons had lower validation loss (1.485) than the baseline. A possible explanation why the lower amount of neurons did not perform as well as 250 neurons is that having too few neurons might limit the model's ability to capture intricate relationships in the data, which leads to under fitting. Another hyper parameter tuning that was attempted was adjusting the dropout rate. We already tried doing a dropout rate of 0.1 in the baseline, so we tried the dropout rates of 0.2 and 0.3. The validation loss for dropout rate = 0.2 was 1.469 and the validation loss for dropout rate = 0.3 was 1.488. A possible reason for this is that a dropout rate of 0.1 may be too low, which does not provide sufficient regularization to prevent over fitting while a dropout rate of 0.3 may be too high, leading to under fitting by dropping out too many neurons and hindering the learning process. From observing the heatmaps generated

by the music sequences, they provide valuable qualitative insights into what the model has learned. For instance, neuron 0's activation at the sequence headers suggests it has learned to recognize the structural components of the musical pieces, potentially aiding in understanding the sequence's start and end. Neuron 10's sensitivity to capital letters in music titles might indicate its role in identifying key features or thematic elements within the music titles, which could be crucial for interpreting the music's style or mood. These heatmaps not only serve as a tool for model interpretation but also reinforce the importance of architectural choices in model performance.

6 Authors' Contributions

Vivian pair-programmed with Jack in completing train.py and assited debugging with indentation errors. She also did hyper parameter testing with RNN and dropout rate = 0.2 and generated plots for both. In the report, She wrote about Hyper-parameter Tuning, part of LSTMs, and Discussion, and part of the README.md file.

Elsie pair-programmed with Hou in completing and debugging SongRNN.py. She also did hyper parameter testing with the baseline model, hidden neurons of 200, and dropout rate = 0.3 and generated plots for all of them. In the report, she wrote Teacher Forcing, Feature Evaluation, and some of Results.

Hou pair-programmed with Elsie in completing SongRNN.py, as well as debugging SongRNN and generate.py. He also worked on generating heatmaps. In the report, he wrote about the heatmaps in Discussion, Heatmap for Best Model section, and part of the README.md file.

Jack, I work on train.py with Vivian. Did the hyper parameters for 250 neurons and wrote the introduction for the writeup, the temperature scaling, and did all the plots and images on the report. I also did the song generation part of the report and part of the LSTM introduction.

References

Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. `https://D2L.ai`.

A Temperature Music Sheets

A.1 T = 1

Link to Image

A.2 T = 0.5

Link to Image

A.3 T = 2

Link to Image

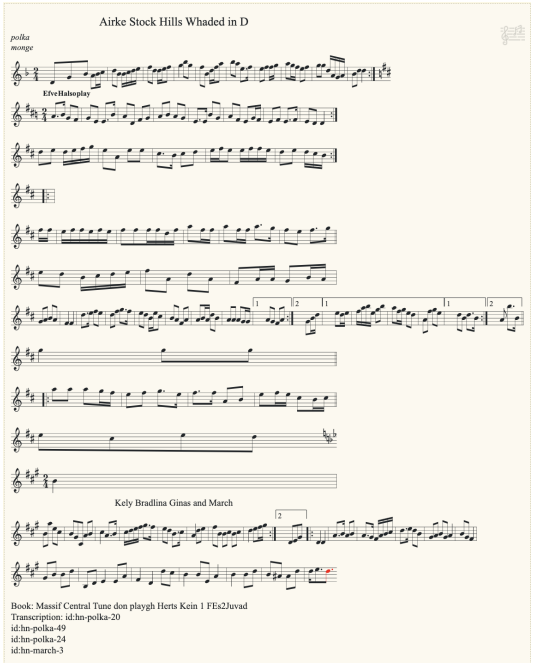

Figure 11: T1 Music Sheet

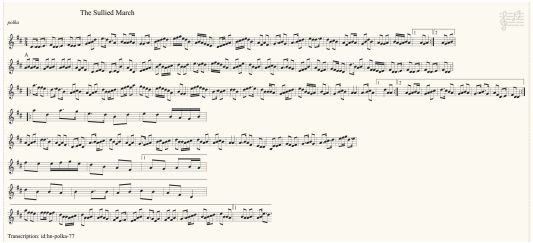

Figure 12: T0.5 Music Sheet

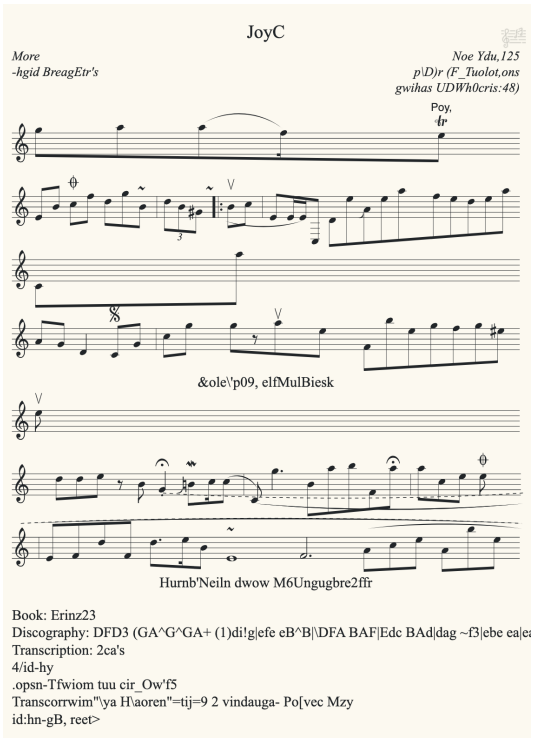

Figure 13: T2 Music Sheet