DIGS 30032 Final Paper

Bryant Cong

May 24, 2023

1 Introduction

In my final project, I attempt to generate music from ABC notation, which textually represents musical sounds in the form of an alphabet.

With its long human heritage, music is often labelled a "universal language" that transcends culture and can be understood by all people (See, for example, [Higgins 2012]). In doing so, scholars usually discuss the shared auditory dimension of language and music: they link the seemingly common human ability to understand musical sound with the ability to comprehend spoken language.

A less-discussed link between music and language is the shared tradition of representing their sounds with writing. This raises the natural question of whether it is possible to convert written language to music, and vice versa. The former is the question in a more limited form that motivates my final project. Instead of working with natural language, I choose to work with ABC notation, which is specifically designed to capture musical sound.

To generate music from ABC notation, I train a decoder-only model with architecture mimicking that of GPT-2. My results are acceptable but exhibit noticeably worse audio appeal and musical structure than a human-composed musical piece. Although I am able to produce some music samples, they are short, do not respect musical meter, and contain many mistakes in their ABC notation representation.

2 Ethical Considerations

Music is a core aspect of human culture, and there are many ethical questions to grapple with as I attempt to artificially generate some. Most importantly, should we as humans even want to live in a world where it is possible to create music from text?

My personal answer is **"yes, but ..."**. I am comfortable with the idea of generating music, but I believe that the purpose for which the music is generated matters. For me, an acceptable application is using generated music to inspire one or two motifs that may appear in a personal music composition. This seems to me like a way in which artificial intelligence assists and complements human creativity, but does not take it over. This is particularly the case with ABC notation, which was originally created to preserve folk tunes from Ireland and Scotland (Walshaw 2023). Accordingly, a dream application of the technology that I investigate in this paper would be to aid a modern-day artist in envisioning how new, modern folk tunes might be created to supplement the cultural heritage of tunes that already exist.

I am highly uncomfortable in applying text-to-music to generate free-standing pieces on their own, doubly so if the creator intends to commercialize those pieces. I think that deploying text-to-music in this case would insult and diminish human creativity, as running a script would replace the hard work of crafting a piece of music. Thankfully, ABC notation is fairly limited, so I am not concerned when it serves as the source "language." I am also more worried about machine learning technologies that seek to directly generate audio and sound forms, instead of relying on textual notation.

3 ABC Notation and Data

3.1 Overview

As briefly mentioned in the last two sections, ABC notation is a system to transliterate the Western musical scale.

It allows text to represent musical pitch (notes like C, D, E, ...), annotations (sharps, flats, etc.), as well as complex rhythmic patterns like triplets (Walshaw 2023). See Figure 1 for my transcription of the opening bars of "The Scientist" by Coldplay into ABC notation. As can be seen, each tune consists of a header (containing information on the artist, song title, time signature, etc.) followed by the tune body.

The dataset that I work with is the Nottingham Music Database. This database is a collection of over 1,200 British and American folk tunes that have been transcribed into ABC notation (Foxley, n.d.). I primarily choose this database because 1) the tunes are cleanly transcribed in accurate ABC notation, 2) it has previously been used in machine learning research, and 3) more contemporary musical pieces are inaccessible due to copyright.

The database contains raw text files with the `.abc` extension, and the files are organized thematically and alphabetically. For instance, jigs are sorted into their own file. Another file contains tunes whose titles start with the letters A-C. Each file contains folk tunes transcribed in ABC notation, with two new line characters separating each tune.

3.2 Text Processing

To begin working with the corpus, I first gather all of the tune fulltexts (including their headers and bodies) as into a Python list, with each tune corresponding to each list element. I then create an 80%-20% split of the tunes into training and test sets. Subsequently, I tokenize the tunes by combining character-level tokenization and a custom function to parse ABC notation. The custom function is necessary since it makes sense for some consecutive characters to go together instead of being separated (e.g., a triplet in ABC notation). Finally, I flatten the tokenized tunes to create one-dimensional training and test set PyTorch tensors consisting of a sequence of characters.

4 Model

4.1 Parameters

To generate the music, I train a GPT-like decoder-only architecture using PyTorch. My full model consists of an initial embedding layer with dimension $d_e = 10$, and a 6-layer decoder stack. Each stack has hidden dimension $d_h = 30$ and is allowed 2 self-attention heads. As is standard in a decoder-only framework, each decoder layer contains a self-attention layer and a feedforward neural network with a Gaussian Error Linear Unit activation function. The PyTorch implementation of a single layer also incorporates normalization and residual connection. Prior to passing the input to the first decoder stack, I incorporate positional encoding that is handled by a matrix $\mathbf{P}$ of size $5000 \times d$. I use a dropout of $p = 0.1$ everywhere and initialize weights across all layers as Gaussian random variables with mean 0 and standard deviation 0.02, similar to the GPT hyperparameters.

At training time, I train the model for 10,000 epochs with a learning rate of 0.0001. I'm able to train for a large amount of epochs due to the shallow model depth, which means that each epoch takes only seconds to complete. I allow for the model to back propagate in time for 35 iterations, and set the batch size to 32. To control the learning rate decay, I use the AdamW optimizer, which uses the running gradient and its empirical variance to update the learning rate (Loshchilov and Hutter 2017). As is the convention for a transformer-based model, I minimize the cross-entropy loss in predicting the next word. I use a GPU partition on the University of Chicago's Midway3 cluster to train my model.

I chose these hyperparameters after my own experiments. Notably, in my test runs, a small d_e and d_h produced the best accuracy, which is why they are tiny in my model compared to even the smallest GPT-x model. I think that this makes sense, given that there are only a few dimensions of the ABC notation vocabulary to learn (tokens corresponding to pitch, note length, etc.), and that the vocabulary itself is small (it consists of only 68 tokens).

4.2 Music Generation

Given a starting sequence, the model's final softmax layer produces a probability distribution over the vocabulary to decide on the next token. To predict the ABC notation token, I choose to use a top-k scheme instead of simply selecting the most likely token. The reason is that after training, the model decides that another bar line (|) is always the most likely token that follows a given bar line. This makes sense, since each piece ends with a double bar line (||). However, this means that once my model predicts a bar line to create a separate measure, it will abruptly end the piece by predicting another bar line after that. To avoid this issue, I use the softmax probabilities to select the next token randomly from the entire vocabulary. For instance, if the model decides that the note E has a 95% probability of following the notes C and D, the E token will be assigned a 95% probability of being selected as the predicted token.

5 Results

I unfortunately did not have the foresight to plot training and test error against number of epochs. However, I do remember that the test error continued to improve for almost the entire duration of the 10,000 epochs. It plateaued at around 1.6, which is serviceable but not outstanding.

To generate music, I fed the model two separate prompts. I first asked it to generate a free-form composition of its own, then asked it to generate a piece in $4/4$ time and in the key of D major.

Overall, the results were tolerable but not the best. Starting with the music itself, the notes selected are relatively reasonable, but there tends to be too much repetition. This is apparent on listening to the samples I kept from the first prompt (here) and the second one (here). Furthermore, there is no respect for tempo and meter at all, as seen by the rapid-fire notes from the second sample.

Even worse, the ABC notation corresponding to the samples had many mistakes. See Figure 2 for a visual representation. As the figure depicts, common errors included nonstandard text enclosed within quotation marks (which should denote chords), as well as meter symbols like the /2 annotation appearing at the start of a measure, not affixed to a note.

6 Discussion and Conclusion

Overall, I'd assess my model performance as less than fair. Neither the generated ABC notation nor the music corresponding to the notation were too reasonable. I have several theories on why my model may have performed poorly.

First, I might have just coded the model incorrectly in PyTorch. I did not have experience using PyTorch before starting this project, and had to learn the transformer architecture as I went along. Therefore, it's completely possible that errors in the architecture caused my model to not perform well.

Second, ABC notation might be too "structured" of a language to work with in music generation. This is counterintuitive, since one would think that a highly regimented language would make it easier to predict tokens. However, musical notation is a language where there *must* be certain elements in sequence–for instance, each measure should contain 4 beats in $4/4$ time signature. This makes it very different from natural language, where words are more free-flowing and sentence length may vary. Therefore, a model to predict the next token may generate interesting natural language, but may not work so well when there are strict "grammar" rules to musical notation.

Finally, building on the previous point, there are other embeddings in ABC notation besides word meanings that I didn't attempt to capture. Rhythm has been my running example, but there's also examples like the overall sentiment of a piece and when a piece should end. It's unclear whether my results' failure to capture these was due to my model, or due to the limitation of the corpus. The pieces in the corpus are surprisingly not annotated with information on the tempo of a piece, such as the 80 beats per minute quarter note tempo that I specified in Figure 1. If I were to improve my project, I would probably try to create dedicated tokens like BERT's `[CLS]` and `[SEP]` that represent the starts and ends of sentences. This, for example, has been done recently with the introduction of tokens like `[BARS x]` before an ABC notation text to specify that x many bars appears in the piece (Wu and Sun 2023).

References

Foxley, Eric. n.d. "Nottingham Music Database," https://ifdo.ca/~seymour/nottingham/nottingham.html.

Higgins, Kathleen Marie. 2012. *The Music between Us: Is Music a Universal Language?* University of Chicago Press.

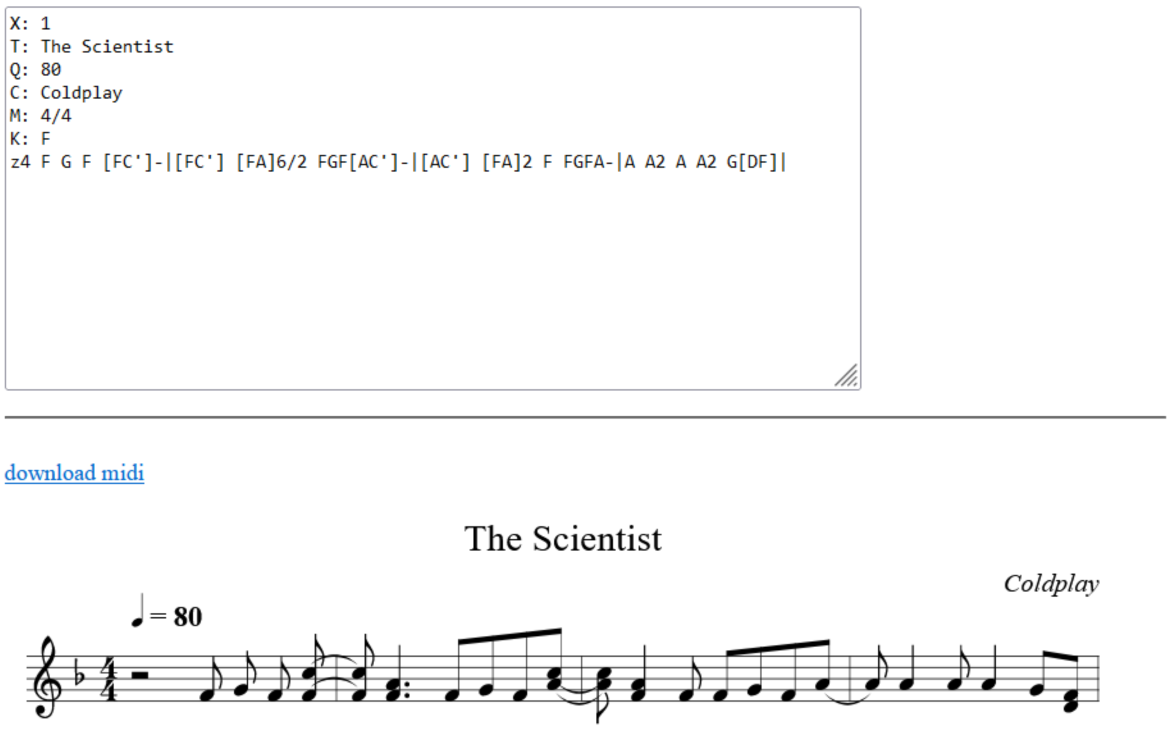
Loshchilov, Ilya, and Frank Hutter. 2017. "Decoupled weight decay regularization." *arXiv preprint.*

Walshaw, Chris. 2023. "About ABC Notation," https://abcnotation.com/.

Wu, Shangda, and Maosong Sun. 2023. "TunesFormer: Forming Tunes with Control Codes." *arXiv preprint.*

A Figures

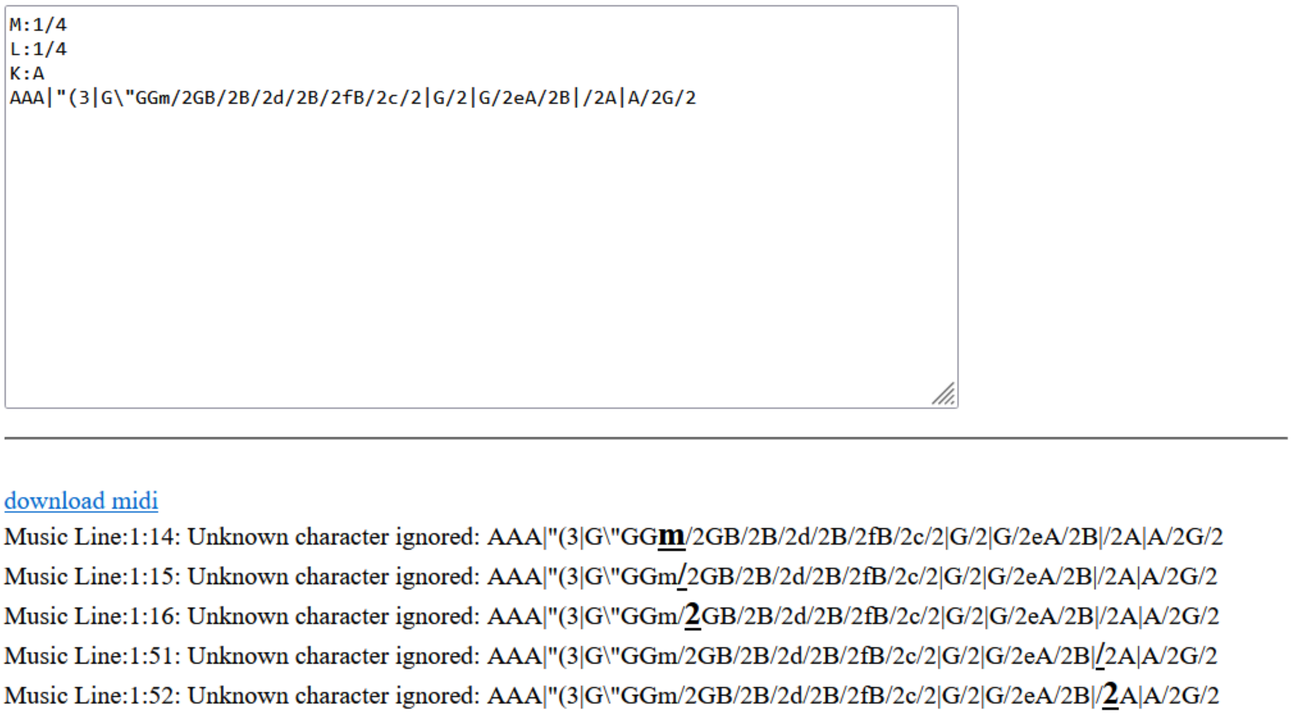
Figure 1: “The Scientist" in ABC Notation



```
X: 1
T: The Scientist
Q: 80
C: Coldplay
M: 4/4
K: F
z4 F G F [FC']-|[FC'] [FA]6/2 FGF[AC']-|[AC'] [FA]2 F FGFA-|A A2 A A2 G[DF]|
```

download midi

Figure 2: ABC Notation Errors in Sample 1



```
M:1/4
L:1/4
K:A
AAA|"(3|G\"GGm/2GB/2B/2d/2B/2fB/2c/2|G/2|G/2eA/2B|/2A|A/2G/2
```

download midi

Music Line:1:14: Unknown character ignored: AAA|"(3|G\"GG**m**/2GB/2B/2d/2B/2fB/2c/2|G/2|G/2eA/2B|/2A|A/2G/2

Music Line:1:15: Unknown character ignored: AAA|"(3|G\"GGm**/**2GB/2B/2d/2B/2fB/2c/2|G/2|G/2eA/2B|/2A|A/2G/2

Music Line:1:16: Unknown character ignored: AAA|"(3|G\"GGm/**2**GB/2B/2d/2B/2fB/2c/2|G/2|G/2eA/2B|/2A|A/2G/2

Music Line:1:51: Unknown character ignored: AAA|"(3|G\"GGm/2GB/2B/2d/2B/2fB/2c/2|G/2|G/2eA/2B|**/**2A|A/2G/2

Music Line:1:52: Unknown character ignored: AAA|"(3|G\"GGm/2GB/2B/2d/2B/2fB/2c/2|G/2|G/2eA/2B|/**2**A|A/2G/2