Group members: Ethan Joseph (eaj73), Sam Willenson (shw58), Luca Koval (lsk83)
Project Category: Music Generation
Project Title: Eat My Jazz

AML Project Milestone

Motivation

We tackle a problem of music generation, specifically that of "call and response." Given four bars of user-generated musical input, we generate a four-bar "response." The motivation for this project came from a problem faced and realized by one of our group members: Often in practice, playing music happens with a group of people. Each person can build off of and harmonize with everyone else. If you're practicing by yourself, however, it becomes much harder to build off of other melodies, harmonies, etc. simply because there are no other people around at that moment. Specifically, as a musician, you might often want to follow a "call and response" format: You play harmony, and then someone or a group of people respond with a similar, but slightly different harmony. Again, if you're practicing by yourself, it is very hard to get a natural response back to your "call." Our project tackles this problem. Given four bars of input music created by the user, we will create a play back with a four-bar musical "response."

Method

We decomposed the problem of music generation into a language modeling problem. The "language" is a sequence of musical notes with information on how long to play each note, and how hard to strike each note. Since transformer neural networks are state-of-the-art on language modeling tasks, we aim to train a transformer neural network to understand and generate this language using a causal language modeling objective (i.e. predict the next note in the sequence given all previous notes).

Our dataset for this task is an open source dataset of jazz MIDI files, which is a format for music that can be interpreted and played digitally. We split the files into train, validation, and testing splits with 70% train, 15% validation, 15% test. We wrote a script to convert these MIDI files into text sequences of notes, and we use the default GPT2 Byte-Pair-Encoding tokenizer to convert this text into a tensor we can feed to our model.

We then finetune a transformer model on a causal language modeling objective on the converted text dataset. The causal language modeling objective is essentially a classification problem, where we want the network to classify the most likely note to occur next in the sequence out of all possible notes, so our loss function is cross-entropy loss. Fine tuning is simply further training a model that was already "pre-trained" on a much larger dataset. We wanted to finetune a model rather than train one from scratch since we theorized that the model may have already been exposed to note names during its pre-training, and as a result learned some associations between notes that we hoped to capitalize on.

Finally, we evaluated the generation performance on the test set by providing the model with a prompt of the first 4 bars of each test example, and asking it to continue the sequence. To serve as an automatic evaluation metric, we decided to calculate the BLEU score between the note sequences our model generated and the actual continuation in the dataset.

Preliminary Experiments

For this initial milestone, we simplified the language to just consist of note names; an example sequence could be: "Gb3, E5, C4". We wrote a script to convert the MIDI files in our dataset into text sequences of notes, using the pretty_midi package.

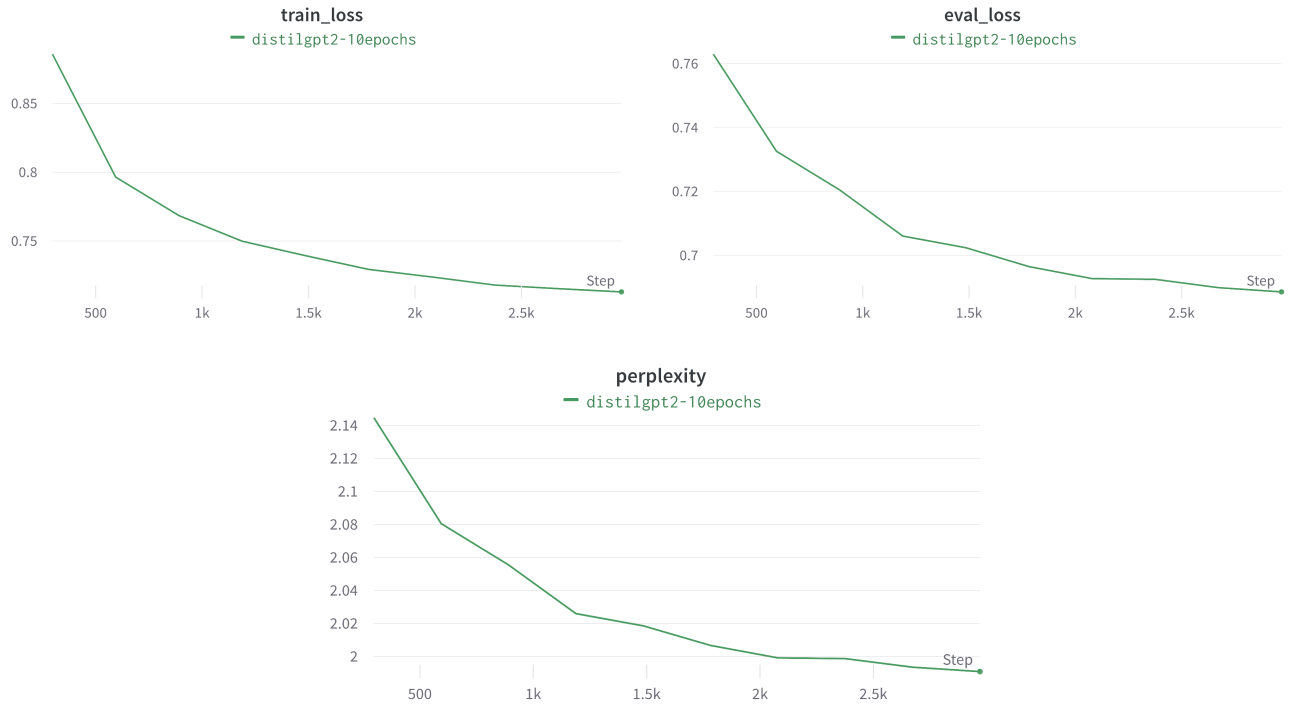
After some research exploring causal language model architectures, we decided to start with a distilGPT2[1] model as our baseline. We chose this since it was small enough to fit on a google colab GPU, while still showing great performance on language generation tasks. We finetuned a distilGPT2 model for 10 epochs, and evaluated perplexity on the validation set at the end of each epoch. Below are plots from the training, which can be viewed in greater detail at this weights and biases page[2]:
https://wandb.ai/yeeeb/trading-fours

During our evaluation, we calculated the BLEU score of our generations on the test set: **10.308** (out of 100). This is a poor result, which indicates that there is a lot of room for improvement on this task, but this is an adequate baseline nonetheless since it validates our methodology of treating music generation as a language modeling task and shows that a language model can at least generate these note sequences. We discuss our ideas for improvement in the next section.

---

[1] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In NeurIPS EMC^2 Workshop.

[2] We hosted the model on Hugging Face where you can interact with it through the inference panel: https://huggingface.co/yeeb/distilgpt2_trading-fours?text=E3+B3+E4+F#4.



Future Work

One step that we have yet to take, but want to for the final report, is incorporating duration and velocity of notes. Right now we do not account for any sort of timing information, but this is incredibly important in music! As of right now, we're not accounting for any sort of rhythm whatsoever. Rhythm is a core part of music, including the "call and response" format, so we will try and incorporate rhythm into our model. We can accomplish this by incorporating timing into the text token for each note, e.g. "Gb3, E5" -> "Gb3s0p25, E5s3p20", indicating that we play Gb3 for less than a second with 25 velocity, and play E5 for 3 seconds with 20 velocity.

Another improvement is to replace the tokenizer. We noticed that the byte-pair-encoding tokenizer has a tendency to split one note into separate tokens, for example "C#4" gets tokenized to "C, #4". This might not be ideal for the model, so we would replace the tokenizer with a simpler whitespace split tokenizer.

We will also try to modify attention mechanisms to further improve the accuracy of our generations. There has been research that shows using a unique relative self-attention mechanism rather than standard self-attention improves performance on music

generation tasks[3], since it allows the model to focus more on relational features, which is incredibly important for music where the relation between notes is what creates harmony.

Finally, we will develop a way to convert the newly generated text sequences back into MIDI files, once again using pretty_midi. After we have our generated MIDI files we can listen to them in conjunction with one another. We have already described our more objective evaluation method, but our final implementation will also include a subjective one. We will simply listen to these MIDI files and note how good of a job our code does at matching the key/theme of the input data. For a more unbiased evaluation method, we will generate our own recordings of what we as human musicians would play in response to an input song. We will generate a few samples from each of us and compare this to the output from the algorithm. We will find participants to listen to a random sampling of these audio files and ask them to identify which ones they believe are generated from an algorithm, and which were played by a human.

[3] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, & Douglas Eck (2018). An Improved Relative Self-Attention Mechanism for Transformer with Application to Music Generation. CoRR, abs/1809.04281.