Thomas Jiralerspong

Deliverable 3

Version 1

My initial model had one LSTM layer with 1024 nodes, with that layer taking in data 50 timesteps at a time. It was fully connected to a dense layer with as many nodes as there were different (pitch, duration) combinations, which turned out to be around 13 000 different classes. I used Adam as optimizer with a learning rate of 0.01. This model trained very slowly and stabilized after only 5 epochs with a cross-categorical entropy loss of around 6 and a training accuracy of around 0.03. When I tried generating music with this model, I realized it was constantly predicting the same (pitch, duration) combination (probably the most common one in the dataset) and was therefore pretty much useless.

Version 2

To improve my model, I tried reducing the number of possible classes it had to classify by rounding the time of the notes to the nearest 100. Although this reduced the rhythmic diversity of the music the model could produce, it could still produce 10 different note durations, which I thought was enough rhythmic diversity for my purposes. This reduced the number of possible (pitch, duration) combinations to around 1000, which was much more manageable. However, the model didn't seem to train any better, stabilizing again after around 10 epochs with a cross-categorical entropy loss of around 5 and a training accuracy of around 0.05. This was better than the previous version of my model, but, when I tried generating music with it, it still constantly predicted the same note, making it pretty useless.

Version 3

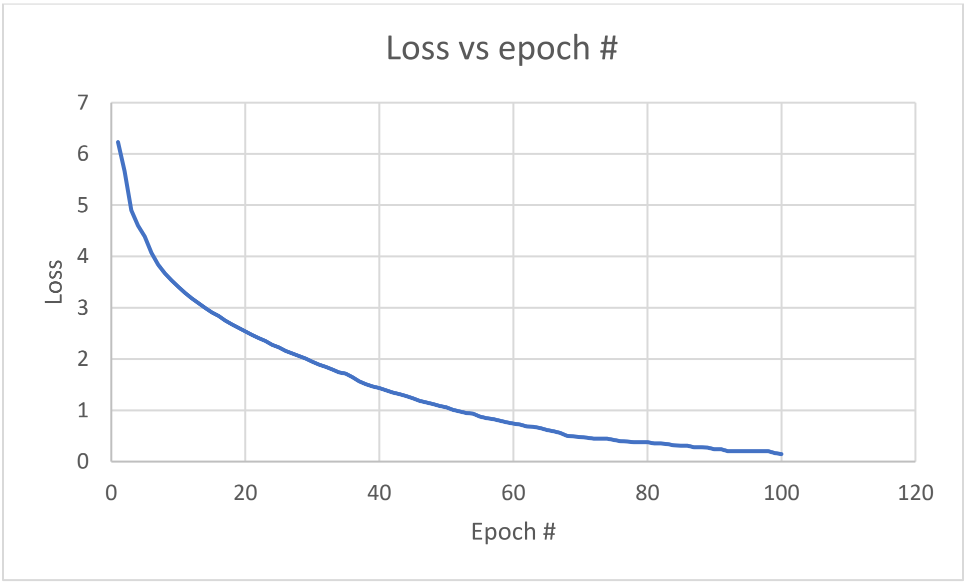
Faced with this failure in training my model, I thought that perhaps my model wasn't deep enough, so I added 2 additional LSTM layers, for a total of 3 LSTM layers, respectively having 2048, 1024 and 512 nodes. This model had over 30 000 000 parameters and therefore took forever to train, but didn't perform much better than the other models, stabilizing after around 7 epochs with a cross-categorical entropy loss of around 4.5 and a training accuracy of around 0.08, which was again a little better than the previous model, but still not very useful, since it again still constantly predicted the same note.

Version 4

Seeing that increasing the depth of my model did not improve its performance, while making it a lot more time-consuming to train, I reduced it to 2 LSTM layers with 512 nodes and 256 nodes respectively. I then tried changing the learning rate of my optimizer from 0.01 to 0.001. This unexpectedly significantly improved the performance of my model, making it stabilize at around 15 epochs with a cross-categorical entropy loss of around 3 and a training accuracy of around 0.15. This was much better than before, and, for the first time, the music generated by this version of the model had different notes. However, after the first few notes, it stabilized and the model again constantly predicted the same note.

Version 5

There was now only one parameter that I had not yet changed from my initial model: the number of timesteps in each sequence fed to the LSTM. I changed this from 50 to 150 and the results improved dramatically. I trained this model for 100 epochs, and the loss steadily decreased, eventually stabilizing around 0.15. The training accuracy also steadily increased, stabilizing around 0.93. The graph of the loss vs. the epoch number can be seen below.



I realize that, at later epochs, the model was probably overfit to the training data, but I saved the weights at every 5 epochs to be able to generate music with the model using weights from different numbers of training epochs.

Some of the music generated by this model can be found in the “results” folder of this repository. The start of each piece seems to be a distorted version of one of the pieces the model was trained on. This is due to overfitting, as the model essentially rewrites the data it was trained on. However, after some time, the generated music seems to be entirely new.

The music generated by this model seems to have a good sense of rhythm, with many repeated rhythmic patterns, but not a very good sense of harmony, as the chords played are often very dissonant. However, sometimes, the harmonies in the music generated by the model do make sense, even when it is generating something completely new, indicating that it did acquire some sense of harmony from the data, though not enough to produce convincing tonal classical music.

This could be due to two things.

First, the MIDI file format makes it hard to get an exact duration for each note, since some notes are ended by playing them at a volume of 0, effectively ending them but making it hard for my model which does not take the volume of the input notes into account. This does somewhat affect the harmonies of the pieces, so this slightly corrupted data might be making it hard for the model to acquire a sense of harmony.

Second, I initially trained this version of my model exclusively on 15 pieces by Beethoven, which may not have been enough for it to acquire a good sense of harmony. I will therefore try to train it on a much bigger dataset consisting of around 60 pieces by Beethoven, Mozart, Brahms, and Haydn. However, this training will take a long time, and I will therefore not have the results in time for this deliverable. My hope is that my final model will be able to generate relatively enjoyable and relatively original piano music to listen to.

Final demonstration proposal

My final product will be a web app with a button that allows visitors to click it to generate an original piano piece in the form of a MIDI file, which they can then download or listen to directly on the website. I will probably use Flask to develop this web app, as I already have some previous experience with it.

If I have time, I would also like to allow people to upload their own MIDI files to use as the starting sequences for the model to generate a piece, but, if I don't have time, I will just take random sequences from the dataset each time instead.