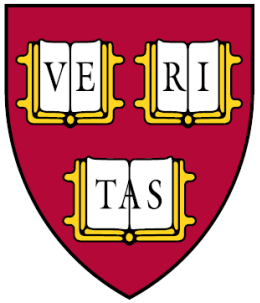
Final Project

Automatic Audio Transcription

Christopher Fuentes



CSCI S-89 Introduction to Deep Learning
Summer 2022
Harvard Summer School

Introduction

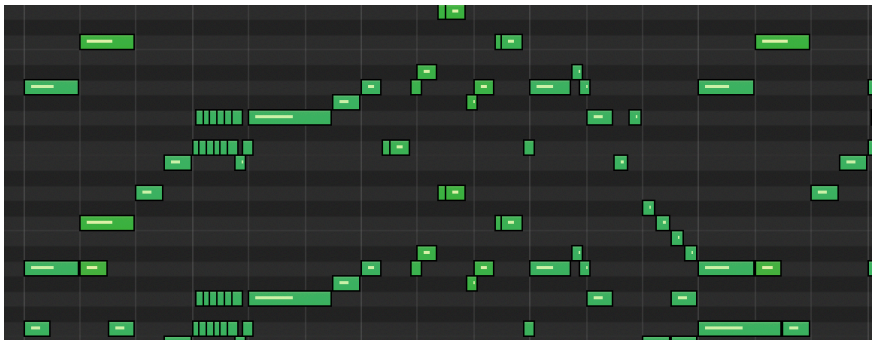
Problem: Can we train a neural network that is able to convert .wav audio to .midi format?

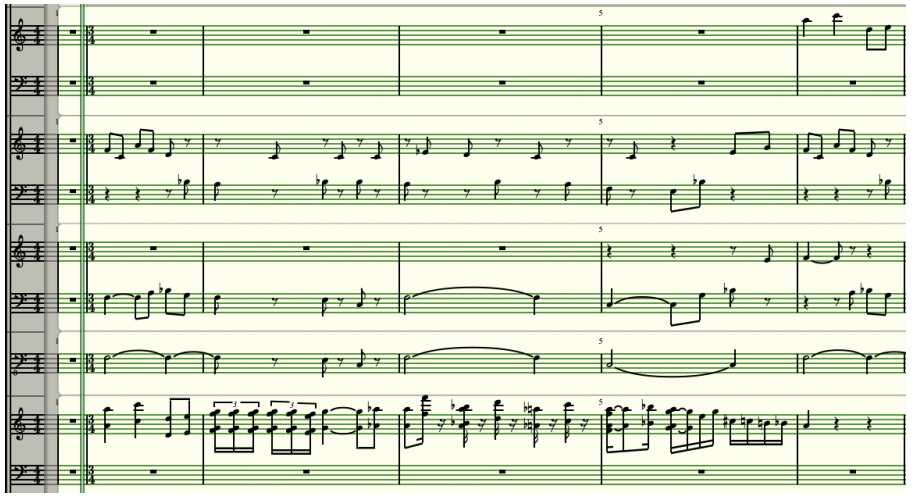
Definitions

.wav file: A digital format that captures audio as a sequence of floating point integers. The sequence is an approximation for a physical waveform that would result from the vibrations caused by sound (the numbers indicate the amplitude of the oscillations, which then produces a set of frequencies). It is a continuous and unstructured format which is visually represented like this:

Definitions

.midi file: A digital format that represents audio *symbolically*. Each note is represented as a token that contains a definition of the notes velocity, pitch, program (instrument), start time, and end time. Because the format is structured and symbolic, it can be easily be represented as a sequence of relative notes or even as sheet music:





These are renderings of the same midi that produced the waveform on the previous slide

Objective

Because midi is just a set of instructions, it is up to the midi player to give voicing to the notes using a sound library. Thus, the same midi file could be rendered as piano, guitar, birds singing or any other available sound library.

This makes midi a powerful format which can both be used by humans to easily understand music (by rendering it as a piano roll or sheet music) as well as for midi players to convert it to listenable audio.

In other words, we can easily convert from midi format into many other representations of audio. However, creating a midi file requires a musician to manually sit and input each note by hand–a very time-consuming process.

My objective was therefore to train a model which could convert from waveform audio into midi format. In other words, **automatic transcription of recorded audio into structured symbolic representations.**

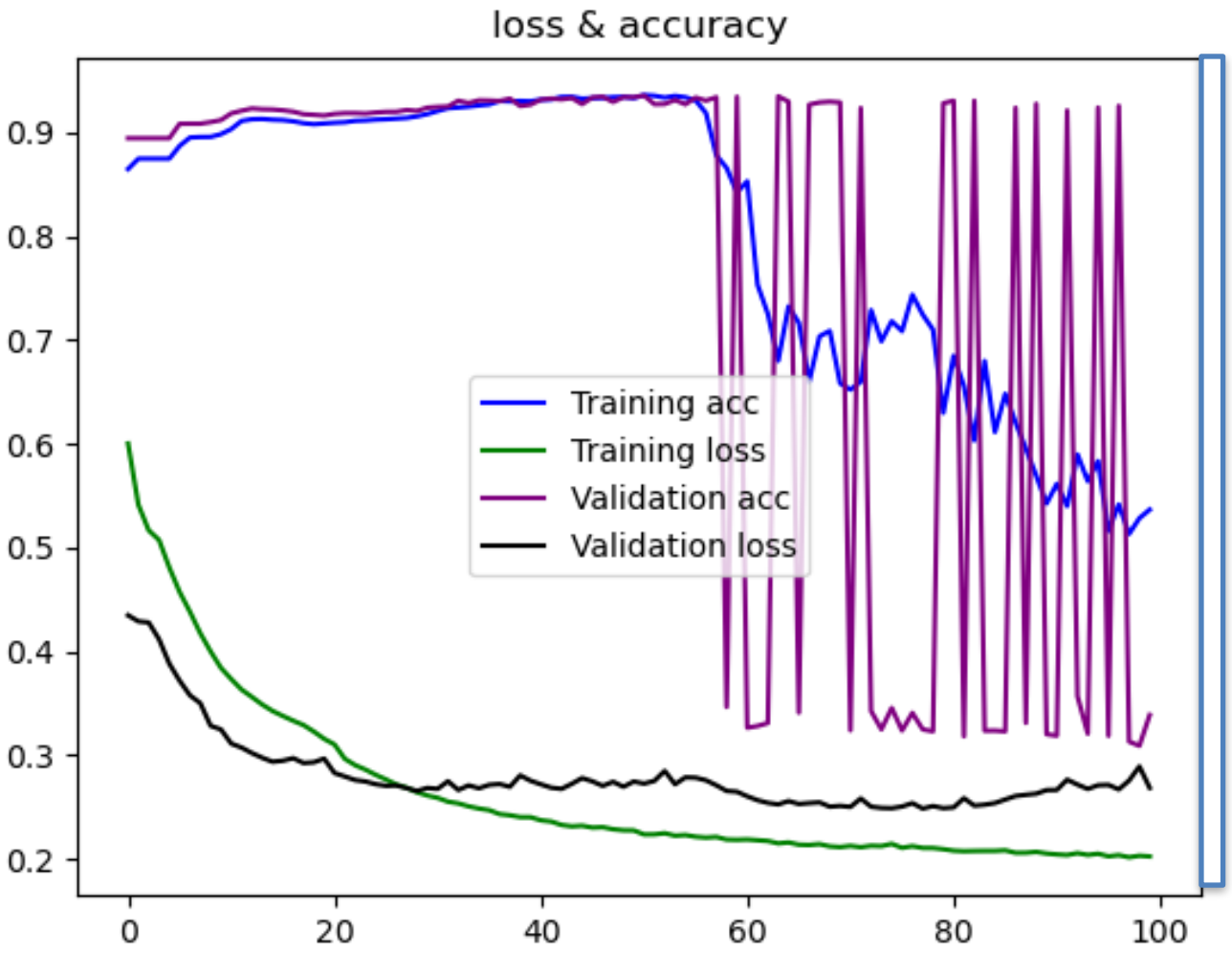
Methodology

- I modelled the problem as a simple machine translation problem: ‘wav’ is the input language and ‘midi’ is the output language.
- I took 1 second slices of the wav data and mapped them to 1 second slices of midi events
- I utilised the MT3 library to encode each midi signal as a set of integers (e.g. { program = 1, velocity = 0, pitch = 52 } could be uniquely mapped to the number like 1300 via a Codec)
- I then built an autoencoder for both wav data and midi data, and combined them to create a translator between the formats.
- The results were terrible. The midi was off-pitch and didn’t follow the song’s rhythm
- The network also had a lot of trouble learning when a midi note should end, so the predicted notes would last minutes each

Methodology, Part II

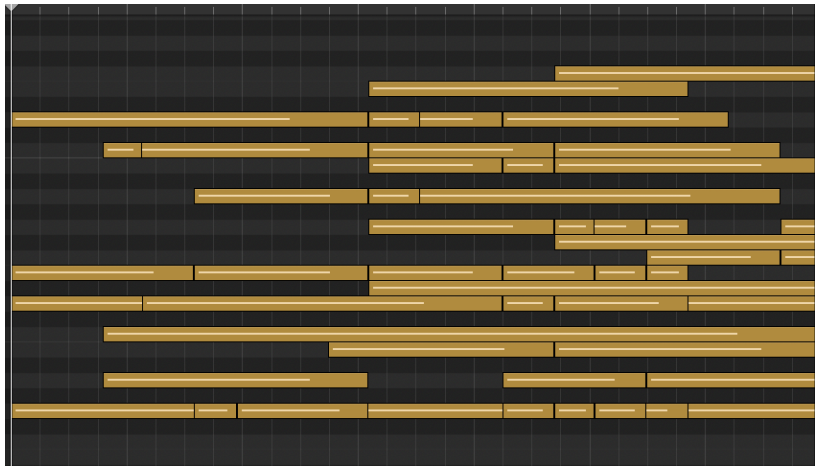
- Made some changes and tried again
- Took smaller slices of data (0.2 seconds)
- Based network off of a simple but proven machine translation model
- Codified midi events using one-hot encoding to ensure we always get reasonably formed events (for a data sample there are usually only a few dozen discrete midi events)
- Used only 16 hidden nodes (input vectors were 800 floats)
- Initially, training loss decreases while accuracy increases. Then, accuracy starts decreasing and fluctuating wildly as training loss continues to decrease.

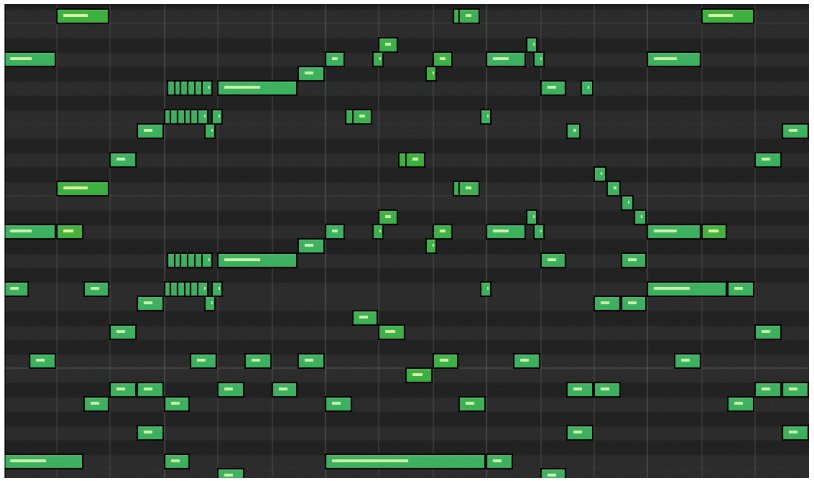
Results


loss & accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100
Training acc
Training loss
Validation acc
Validation loss

Results

- Reconstructed midi sample (left) vs original (right)





- The difference of colour is significant, as it represents note velocity.
- The reconstruction confused most of the instruments into a single track, missed many notes and got the timings wrong.
- However, it did generally get the key correct.

Analysis: what went wrong

Data Problems

- *Silence*: Representing both notes and silences in the stream of midi events caused the data labels to be very skewed: a lightly trained network could achieve 90% accuracy by simply guessing “silence” for every input (this also explains why accuracy starts high and then decreases - as it begins predicting non-silence, accuracy goes down).
- *Polyphony*: the music I was training on has several instruments playing up to several tones at the same time. While this is more typical of music, it also makes training significantly harder, as it means that for any given slice of audio input, there is no bound for how many corresponding midi events should exist (similar to language translation where one word can map to a few, but to a larger order of magnitude)
- *Data volume*: Raw .wav data is just a sequence of floats, and lots of them (a normal audio file has 44,100 floating point values per second of audio). Individual frequency samples are not directly mappable to anything meaningful, and I speculate that I could have benefitted from applying some pre-processing (e.g. averaging or similar)

Analysis: what went wrong

Network Problems

- *Type:* Believing the problem to be a sequence-to-sequence mapping problem, I relied on LSTM layers. This makes sense at a couple of levels (e.g. a given sample in a waveform can be predicted by the previous samples; a midi note end event can be predicted by a midi start event), however didn't seem to work well for translation between the two formats. Omnizart's network, by contrast, relies mostly on Convolutional layers
- *Simplicity*: The network I used has 3 layers comprising its encoder and decoder (2 LSTM and a Dense output). Omnizart's network, by contrast, utilises over 200 layers with many skip connections, dropouts etc.
- *Tokens:* While many language-translation tasks utilise 'start' and 'end' tokens to indicate the bounds of a translated output, there isn't a natural 'end' token in a random sample of midi data. We can't therefore determine when exactly a "translation" should stop.

Analysis: what went wrong

The Domain

- It turns out this is a very difficult and not fully solved domain
- Leading research, like Google's Magenta, can only hit about 82% accuracy.
- Even Omnizart, a state-of-the-art automatic transcriber, struggles a bit on the same examples
- Found a bug in the Magenta MT3 library

Ideas for Improvement

- Experiment with convolutional layers to reduce audio input dimensionality into a smaller set of features
- Experiment with inserting “start” and “stop” tokens to all of the midi samples so we can dynamically determine sequence length
- Experiment with “word embeddings” to represent the midi tokens (even though the dimensionality is already small, perhaps it’d perform better if it were further reduced)
- Implement batch token prediction to improve performance

Conclusion

- Transcribing polyphonic (multi-sound) data was perhaps too ambitious of a target to start with
- Research on the domain is still fairly bleeding edge
- There are enough datasets and open source libraries to help explore the domain further; however the mileage can vary
- There isn't consensus on how the problem should be modelled
- Training a network across a broad dataset will require some powerful computing resources, given the density of the input data

Overall, though I am not satisfied with the quality of the results I was able to achieve, I believe that with some further experimentation and research, better results are possible.

YouTube Video Presentation

https://youtu.be/NOpcXcRIMYA