MAIS202 - Deliverable 2 - Model Validation

Maya Scott-Lourenço

October 21st, 2020

1 Problem Statement

The goal of the project is to generate new polyphonic Pokémon music using the Google Magenta library model PolyphonyRNN[1] trained on the NDS/GBA era Pokémon OST[2]. The aim is to generate audio tracks that are reminiscent of the musical patterns present in music composed for Pokémon games.

2 Data Preprocessing

MIDI files contain a sequence of NOTE ON/OFF instructions for 128 pitches as well as a channel number indicating the instrument. This information can be used to get the ON notes within a frame in descending MIDI pitch and translate it into a wordvector used as input for the model.[3] 837 audio files were ripped directly from the GBA/NDS era Pokémon games[2] which use MIDI as the native audio format. Later games in the series don't encode audio in file types supported by polyphonyRNN and there are also no easily accessible models for training polyphonic music that utilize waveform audio. There was an attempt to use online software to convert waveforms to midi in order to increase the dataset but the loss in audio data and inaccuracies were just too large to be acceptable. As such the dataset remains unchanged from Deliverable 1.

2.1 Attempt 1

Data was preprocessed using two Python libraries for manipulating MIDI files: music21[4] and MIDO[5] for different tasks. First, all the drum tracks were removed from the files as the model is not equipped to deal with percussion. The remaining instrument tracks were all moved to the piano channel as polyphonyRNN will discard tracks with more than 1 channel during the conversion to `SequenceExamples`[1], leaving me with 834 files for input. Each file should also be transposed to one key, providing key invariance of the examples, making them more generic and reducing the sparsity in the training data[6]. The model's `create_dataset()` function luckily takes care of transposing all scores to C-major / A-minor and quantizing the time into sixteenth notes[6] for us so these tasks were omitted from my preprocessing. PolyphonyRNN splices the tracks at time signature changes, treating them as separate examples and I ended up with the following info message after calling the `create_dataset()` function:

```
Processed 834 inputs total. Produced 5341 outputs.
```

Essentially, by putting all the different instruments into one track, there were a lot of time signature changes. These 5341 outputs left me with 16 GB of training data that caused Google Colab's GPU to run out of RAM (lmao) when trying to train the model. I'm able to train in 5 min increments which results in a few steps but ultimately this wasn't viable.

2.2 Attempt 2

Since the Pokémon OST is so varied in its instrument usage, I could not just take one MIDI channel i.e. "Channel 5 - Guitar" and call it a day. Additionally, there was no foolproof script I was able write that would extract the MIDI instrument channel(s) that most represent the music. Attempts at this task resulted in

MIDI tracks with just a bassline or half of the melody as the original track has it split between various string instruments, few of which were reminiscent of the original tracks. So I settled for just giving polyphonyRNN the no-drum audio tracks and hoping for the best, deciding on a 10% hold-out validation[3] for the first run.

```
Processed 834 inputs total. Produced 104 outputs.
```

I wanted to compare this to the percentage of outputs produced from other video game music that use less channels and ran `create_dataset()` on a NES music database[7] which has a maximum of 4 instrument channels. Unfortunately, with no preprocessing it was also terrible:

```
Processed 5278 inputs total. Produced 198 outputs.
```

The documentation is quite elusive as to what is and isn't acceptable and only provides info on the desired format, track length and number of channels. Overall a terrible reduction in data but I was running out of time at this point so I went with it.

3 Machine learning model

3.1 Framework and Tools

PolyphonyRNN is an implementation of BachBot, an RNN with LSTM memory cells[3] depicted in Figure 1. A RNN adds the notion of memory to a feed forward network. The addition of a memory cell allows us to take the hidden layer activation from the previous time and pass it back in as an input for the current time. The hidden state is propagated forward in time and acts as a form of memory allowing the model to remember where it is in the composition and generate more realistic notions of phrasing and conclusions to phrases[6]. The model uses a backpropagation algorithm that relies on stochastic gradient descent to optimize note predictions[3]. Lastly, it uses dropout, a regularization method to select a random percentage of neurons and their connections to drop from the LSTM, to reduce overfitting[8] These features make polyphonyRNN a realistic model for approaching the outlined goal. The default settings were used for the first run and the model was trained for 1000 steps. Default hyperparameters: `batch_size=64, rnn_layer_sizes=[256, 256, 256], dropout_keep_prob=0.5, learning_rate=0.001`

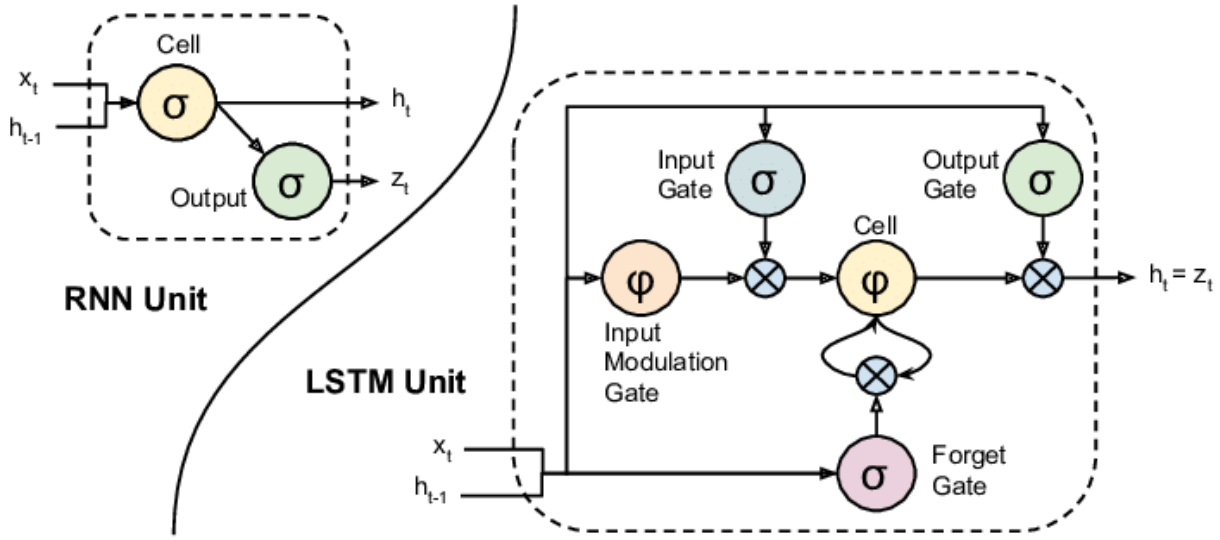

Figure 1: A diagram of a basic RNN cell (left) and an LSTM memory cell (right)[9]

3.2 Validation Methods

The model uses cross-entropy loss on held-out data to evaluate the performance quantitatively[3]. As the number of steps increases the model becomes better at creating outputs that sound more like music. Quantitatively we see this result in the increase of accuracy as the number of steps increases. (Figure 2) These measures of performance do a good job of predicting the musicality of the output and are often used in music generation[6] but don't give any indication as to whether the goal of producing Pokémon music specifically is being met. The moderately low loss and moderately high accuracy seem to suggest the model is not overfitting but the outputs do have common elements but this is likely from the very small data size due to the ineffectiveness of the data preprocessing and not from the model's implementation.

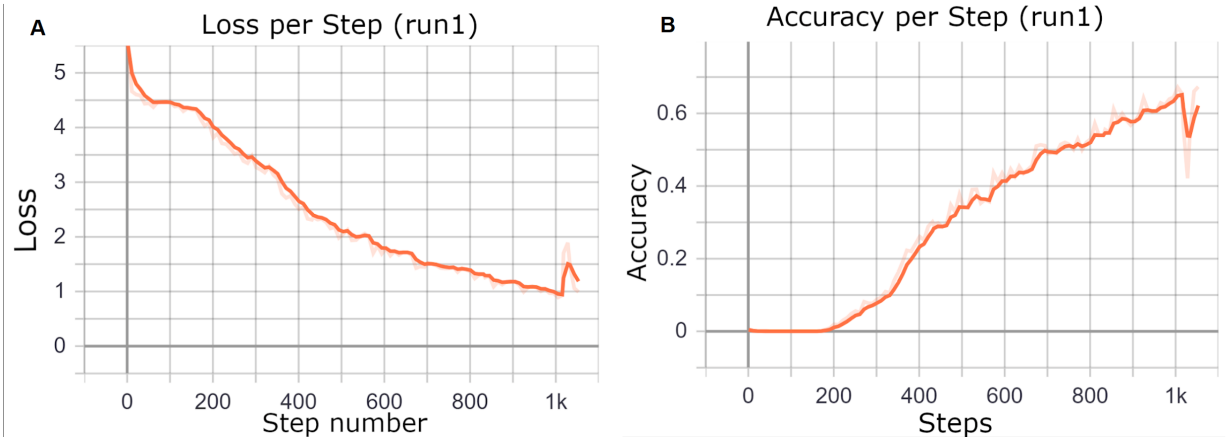

Figure 2: Loss(A) and Accuracy(B) plotted with respect to the number of steps the model has completed

3.3 Challenges

The biggest challenge was realizing my expectations were a bit out of proportion with the goal. The actual coding part of the model was seamless as Magenta provides good documentation and commands for running the model despite the lack of information on data processing. The bigger coding struggle was trying to process the MIDI files in an efficient way and work within the quirks of each MIDI manipulation library's limitations.

4 Preliminary results

Please prepare your ears :pepeJAM: Here's a soundcloud playlist of some tracks directly output from the model:
https://soundcloud.com/maya-sl-743088155/sets/ml-generated-pkmn-music

Here's a playlist with 5 tracks where I quickly split some of the main melodic sections into a few instruments common to the dataset.
https://soundcloud.com/maya-sl-743088155/sets/pkmn-ml-edited

I asked 5 friends to do a musical discrimination test[6] and the ML tracks were identified correctly 100% of the time by each participant (except 1 track!) regardless of the participants knowledge of the Pokémon OST. The biggest giveaway is that the original tracks are very familiar to those who have played the games and have a noticeable melodic structure even to those that have not. It's difficult to rigorously evaluate the models performance due to the nature of the outputs but it is evident that since the model was made with Bach chorales in mind which only feature 4 simultaneous melodies, the melodic structure of the input tracks with >4 instruments (Figure 3) many of which are not contributing to the melody is not ideal.

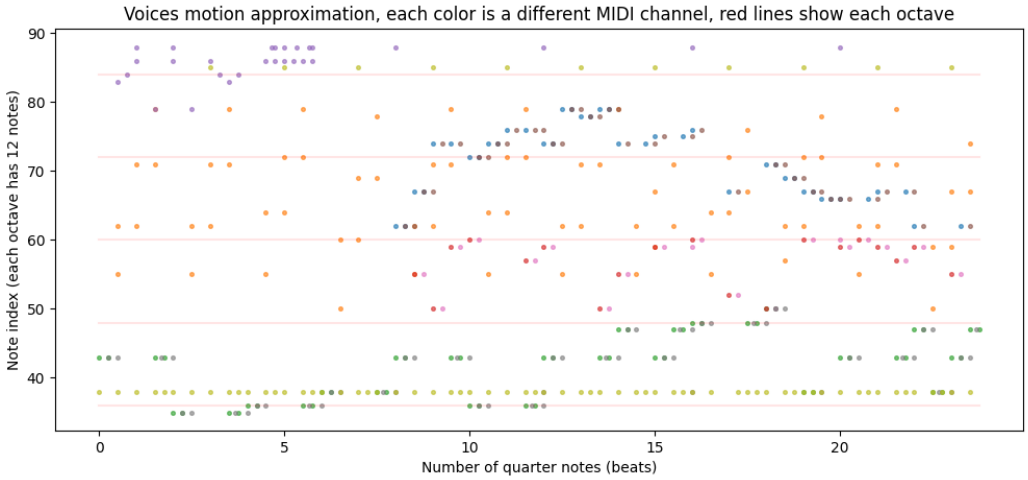

Figure 3: Voice motion approximation of Route 2 [Spring]. Different from the typical SATB Bach Chorale Structure

5 Next Steps

The immediate pro of the first training run was that music was produced that does have some melodic moments. Having the outputs in a single piano channel often makes it hard to get a "Pokémon" feeling from them as the entire discography is composed of music that uses multiple instruments. Interestingly, even with my rushed and naive separation of different melodic sections into different midi channels, there is a improvement in the output tracks resemblance to Pokémon music. The overfitting is a bit glaring upon listening to enough of the output tracks as the notes are very repetitive and generated melodies will appear in tracks primed on different MIDI files. While the results are interesting thus far, I definitely want to train the model on more of the dataset. I would also like to experiment with a different `rnn-layer-size` and `batch-size` that are more reflective of the input dataset and not just the recommended hyperparameters if time permits. The focus however will be on increasing the dataset. I'd also create bundle files of the model throughout the training process so that I can go back to a previous iteration of the model in the case of overfitting.

References

[1] Magenta Team Google Brain. *Polyphony RNN, revision eed016a*. 2016. URL: `https://github.com/magenta/magenta/tree/master/magenta/models/polyphony_rnn`.

[2] *MIDI files of the Pokemon FRLG, RSE, DPPt, BW, HGSS OST*. URL: `https://drive.google.com/file/d/1n4aIi1KS4DPMjMMaz_UjqRKyjFKwKWOY/view?usp=sharing`.

[3] Feynman Liang et al. "Automatic Stylistic Composition of Bach Chorales with Deep LSTM". In: *18th International Society for Music Information Retrieval Conference*. Oct. 2017. URL: `https://www.microsoft.com/en-us/research/publication/automatic-stylistic-composition-of-bach-chorales-with-deep-lstm/`.

[4] Michael Scott Cuthbert and Christopher Ariza. "Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data." In: *ISMIR*. Ed. by J. Stephen Downie and Remco C. Veltkamp. International Society for Music Information Retrieval, 2010, pp. 637–642. ISBN: 978-90-393-53813. URL: `http://dblp.uni-trier.de/db/conf/ismir/ismir2010.html#CuthbertA10`.

[5] Rapolas Binkys Ole Martin Bjørndalen. *Mido - MIDI Objects for Python*. 2013. URL: `https://github.com/mido/mido/tree/stable#readme`.

[6] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. *Deep Learning Techniques for Music Generation – A Survey*. 2019. arXiv: 1709.01620 [cs.SD].

[7] Chris Donahue, Huanru Henry Mao, and Julian McAuley. "The NES Music Database: A multi-instrumental dataset with expressive performance attributes". In: *ISMIR*. 2018.

[8] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[9] Jeff Donahue et al. "Long-Term Recurrent Convolutional Networks for Visual Recognition and Description". In: *Arxiv* PP (Nov. 2014). DOI: 10.1109/TPAMI.2016.2599174.