**Multi-Agent Musicians With Recurrent Neural Networks**

Mark Karle

CS 6366 Distributed Artificial Intelligence

Fall 2016

**Introduction**

AI-generated music is a fascinating topic which has been done in various ways, such as with recurrent and/or convoluted-recurrent neural networks which learn from data, with multi-agent systems that treat each musical note as an agent and use reinforcement learning and music theory rules, as well as multi-agent systems that have a composer agent and drumming agents that co-evolve a rhythm together, etc. The first two essentially learn to improvise on a single instrument, which can be thought of as a single musician or agent. By creating an agent for several different instruments, a multi-agent band can be created. Each agent can create its own music that sounds good in its self-interests, but playing together will require communication in the form of things like pulse, rhythm, key, and chord progression. This project attempts to create such a multi-agent band by combining Long Short Term Memory neural networks trained on different instrument pieces, specifically guitar and bass parts from Beatles songs, by feeding some of their outputs to each other.

**System Description**

Long Short Term Memory neural networks seemed to be the best fit for this project. They can learn from real music instead of musical rules which plain reinforcement learning could use, and this is important because music varies far beyond the rules we have created for it. Also, LSTMs have a sense of time unlike feedforward networks and can uncover global musical structures (longer than one time step) unlike regular recurrent neural networks. In the blog post from [1] which is used as the base for this project, Johnson explains his use of a stack of identical LSTMs for each note, which in a sense are multiple agents (networks) working together. Taking this an abstraction level higher, the multiple-agents for this project combine these stacks which are trained on separate voices. One could use a central stack for the entire composition, but separating the voices may help each one specialize their musical roles better.

The LSTM stack in [1] works by having a network for each note. The inputs are the MIDI note value, the pitch class, whether the surrounding notes were played previously, the previously played pitch classes, and the beat. The first stack of hidden layers are recurrent in time. The second stack of hidden layers connects nearby notes' networks. Then the play probability and articulation probability are outputted. To generate music, a seed value is taken and the network model predicts the next play probability and articulation probability for each note. This code was forked from its repository [2] and then modified: when generating the music, two networks co-generate. This was done by taking the notes which are played from one model, generalizing them to their pitch classes, and then inputted to the other model's pitch class input nodes. Crossing inputs base on note value alone would confuse another model; a guitar using high octaves would not have been trained on notes from a bass' octaves, but it would have been trained on the pitch class the bass uses.

**Experimental Design**

The experimental goal was to find a difference between a centralized LSTM trained on multiple voice and multiple LSTMs trained on single voices which, when combined by crossing inputs and outputs, would create interesting music. The idea stems from real music: a musician does not know everything his colleagues are playing at every instance, but he does have a rough idea. In this case, the rough idea is simply pitch class. I expected a centralized model would have better harmony and the different voices would work together better and create more cohesive music, whereas separated models would specialize better and create music which is still cohesive, but will include more subtleties of its specific instrument.

To experiment with the models, a collection of Beatles sheet music was converted to MIDI format and stripped of drums and vocals. Then three training sets were created: one with both guitar and bass tracks, one with just guitar, and one with just bass. These files were then stripped of long, silent gaps which would be common for tracks representing solo guitar for example. Each of these three midi repositories were used to train a model. Then, the centralized model generated samples alone while the two models with single voices were used to generate samples together as described in the section above.

**Training Methods**

The neural networks were trained and the gradients were optimized with Adadelta, which is built-in to Theano, the deep learning Python library that was used. To prevent over-fitting, dropout was used. Each iteration, a random 50% of hidden nodes were removed so that nodes specialize better and do not depend on each other as much.

**Experimental Results**

The results are hard to define and certainly more information can be gleaned from listening to the samples generated, but I will describe what they sound like. Some samples were generated by the model which included multiple voices, and some were generated by the models which only had one voice. The first group of samples had more consonant chord structure. The articulation was more uniform among perceived voices, that is, high and low octaves played notes at similar times rather than independently; that being said, at times some voices would cut out and give the others solos. Also, these samples get stuck in the same chord for a long time and were repetitive. The second group of samples were able to create more independent parts but the counterpoint was not always consonant. Certainly the voice parts were more complex.

Some samples using just the guitar model and just the bass model were also run. They had more silence and got stuck in the same chord just as the centralized samples did.

**Discussion**

Samples generated from the centralized or from a single bass or single guitar model were less independent in their voice parts than those which combined the guitar and bass models. Perhaps at some point in training and minimizing error, the different octaves play off each other too much. The mixed model generated samples also had less repetition and got stuck in the same chord less. The other model in the generator may have acted as a sort of forcing term, and that caused both models to continue developing complicated parts. Perhaps since the centralized samples were aware of what all the voices were doing, some can cut out and the music continues, whereas the mixed model samples get an input from another model and thinks it is itself playing and continues playing off of that. In a sense, in the centralized samples, the voices are listening to each other, but in the mixed samples the voices are each trying to be the spotlight.

During the project process, a lot of hurdles were in the way. Training on my laptop and desktop were prohibitively slow (it would have taken weeks) due to lack of a GPU to process things. Then getting access and free credit for Amazon Web Service EC2 instances took longer than expected. Unfamiliarity with Theano, the deep learning library that was used, also slowed things down. There were also problems with the training, such as a model's propensity to just generate silence and needing to be retrained; further culling of bad training data (those with lots of silence) could have prevented it. The slow training made it harder to tinker with different configurations, so I was unable to try very many different ideas. If I were to do it differently, I would need to understand that this is not the type of project which can be procrastinated.

**Future Work**

The mixed model generation was done by giving the pitch classes outputted from one model to the inputs of another. While this probably does better than inputting the raw note, it likely does not do as well as inputting the overarching chord to the other model. A bass line may hit many notes but have in mind a particular chord that these notes fit around. Telling the guitar each note is not necessary and perhaps gives too much information which can confuse the model, but giving the chord would allow the guitar to know what acceptable notes can be played. The LSTM does a good job of remembering far enough to not jump to a different chord each time step, but I think sharing an explicit chord to be played at each time step would improve things. The problem is, the pieces must be notated with the chord of a measure and then then networks must be trained on these newly notated pieces.

In the future, perhaps data can be scraped from actual sound clips rather than written out pieces. This would allow much more training data to be used instead of relying on someone notating music. Also, there is a reason a guitar plays a solo rather than a bass: music not only depends on notes, but also on timbre, which cannot be learned from MIDI files, but can be from raw sound files. This would help generate better music.

**Conclusions**

LSTM networks are shown to create interesting pieces of music from a training set of single voices, which can be thought of as centralized parts. However, in real life musicians are different people with their own thoughts and musical ideas. This work aims to recreate the multi-agent aspect of music by training multiple neural networks on different instruments and mixing them together. The mixed models specialize on their instruments better than the centralized models but are less cohesive. Although no explicit musical rules were given, the samples generated showed some interesting musicality. This is a good step towards more AI-generated music of our favorite bands.

**References**

[1] – Johnson, Daniel. Composing Music With Recurrent Neural Networks (8/2/2015), <http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/> (DOI)

[2] – Johnson, Daniel. Biaxial Recurrent Neural Networks for Music Composition, <https://github.com/hexahedria/biaxial-rnn-music-composition> (DOI)