Inspiratron.org - Natural language processing, machine learning and cybersecurity

SchumaNN: Recurrent neural networks composing music

by Nikola Miloševi? - Sunday, January 01, 2017

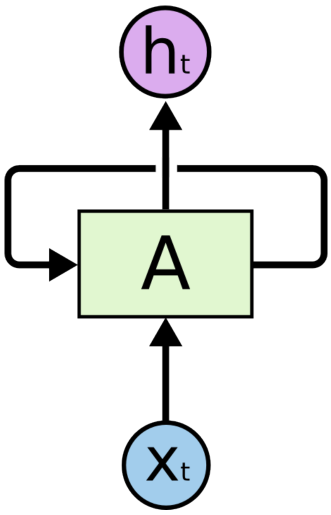
https://inspiratron.org/blog/2017/01/01/schumann-rnn-composing-music/

With 2 friends of mine (Team was: Maksim Belousov, Mike Phuycharoen, Nikola Milosevic (myself)) on 12th and 13th November I participated at the GreatUniHack that was held in John Dalton building of Manchester Metropolitan University. We were implementing idea of machine learning-based music composer, for which we later came with name SchumaNN. For some time I wanted to experiment with this idea and this 24 hour hackathon came as a perfect match. We had couple of meetings prior to hackathon when we discussed some solutions and papers/blog posts we read. Some of the approaches we found are:

- https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn/
- http://www.aclweb.org/anthology/W14-0901
- https://arxiv.org/pdf/1611.00138.pdf

The decision we went at the end is to create a recurrent neural network (RNN) that will learn from MIDI files of classical music.

How recurrent neural networks (RNN) work?



Recurrent neural network

Recurrent neural network is a type of artificial neural network where connection between units form directed cycle. The structure of one node is presented on the image bellow.

This structure allows for internal stages that are dependent on previous actions. In other word it allows dynamic temporal behavior and some simulation of memory. Dynamic temporal behavior and internal memory makes recurrent neural networks applicable for music composition, in which as input we will provide first couple of seconds and our program should continue and compose some listenable music. The basic idea is that for the provided a number of notes, the algorithm should provide next note. Then it will move window by one note and try to predict what happens next and so on. We set a time-frame for which the music should be generated, at the end of which we stop our algorithm. The neural network part is quite straight-forward, following standard approach and guidelines. For the implementation we used Keras.io library that was on top of the TensorFlow.

Modelling data structure for learning

As data model I would refer the structure of data that is fed into the recurrent neural network. This was actually the hardest part, since we changed the model several times during the hackathon, since many of the models we tried didn't work. As input to our algorithm we used MIDI files of classical compositions for piano.

We were thinking about two libraries to read MIDI files. One was JFugue, which is excellent library for Java. However, since the rest of the approach was developed in Python, we ended up using Python-MIDI.

As input to our program we need to define sample rate and spread of keynotes that will be taken into account. In MIDI file, usual 7 octaves of the piano spread between codes 24 and 102 in MIDI system. We used sample rate to split time of the song into small chunks (depending on sample rate). For each sample we extract the line with max_codes-min_codes binary values. For each code we will assign 1 if the key is pressed and 0, if the key is not pressed for that code. Example of part of the line would be:

0000000000100001000001001010000...

These lines we will feed one by one to our recurrent neural network. The network is creating output in the same format. However, network will output probabilities for keys being pressed that is in range between 0 and 1. Usually these probabilities, especially if the data is not trained enough will be quite small. So we made an hack in each cycle, making each note pressed in case output of the recurrent neural network is larger than 0. This way we are getting more outputted notes and it turned out that it did not affected generated melody too much. Also, for since the output lines are based on sample rate and there is one line for each discreet time moment, in case there are multiple time moments (samples) that have same not pressed one after the other, we merge this to one long note (lasting as many time moments as the note was pressed).

The code we developed is available on GitHub: https://github.com/mbelousov/schumann

As data set for learning we used following data for training: http://www.piano-midi.de/

Learning and results

Since we lacked time for training we at the end trained our algorithm over quite a few epochs but just with one song (another hack, but oh well it is hackathon). Although as I was thinking about this, this maybe even makes sense. In order for someone to learn to compose melodically like some composer, one needs to listen melodic structures over and over again. So one song was not even that terrible idea. The number of epochs we trained during hackathon was about 100 and it gave some quite melodic results that people in audience liked. We run training on AWS for which we spent about 30GBP. We were unable to get advertised $100 from Major League Hacking coupon, since there was noone on Amazon side to activate the code. We wrote even couple of emails to support and guy from Major League Hacking tried to helped, but no meaningful response came from Amazon before the end of the hackathon.

For inputs we provided beginnings of some songs from the data, so again couple of seconds of MIDI files of classical composers such as Schumann, Mozart, Tchaikovsky, etc. You can listen to couple of generated melodies below. Interesting part is, that even though it was learning from classical song, and input was given from classical song, the generated songs are more some kind of modern piano jazz music. Hope you will enjoy it and that this will fuel some further research in the area, even though, this is very basic approach developed in some 24 hours. At some point we may extend this work and write some paper, but for now here is this blog post to have a record of what we did.