Android Mobile Malware Detection using Neural Network Classification

Austin Abeyta

August 14*, 2018*

Capstone sponsor Dong Si*,*

University of Washington Bothell

abeyta2@uw.edu

Abstract—

Music has been heavily dependent on computers since the 80's, and with each new advancement in computers, new variations in music soon follow. How will the future of pattern recognition effect the music industry? This capstone explored a musical interface that will help artist explore drum loops created from a deep neural network. Hao-Wen Dong has created a model using generative adversarial networks and hopes it can be improved, I will use this open source project as a starting point for my own improvements. I focused on drum generation using the dataset provide by Hao-Wen Dong. After creating a custom decoder encoder model I was able to reduce the average sample size of the data by, 98.6%, all while losing very little musical information. This allowed for large amount of computation to be performed where before was un feasible.

Keywords—

Machine Learning, Neural Networks, Classifier, Midi, Piano Roll, Quantization, Time Steps. Pitch Numbs, Muscial Bar

I. INTRODUCTION

After hearing "Daddy's Car" , a song entirely composed by artificial intelligence in the style of the Beatles, I became fascinated in the future of AI created music. Music has always been a large part of my life and I produce as well as play piano and drums. I was shocked at the how well AI's could recreate moods and styles of music. That is when I began to teach myself TensorFlow in a long process of understanding the growing relationship between music and AI.

After a presentation on music and machine learning for University of Washington's Data Analysis and Intelligent Systems Group, Professor Dong Si suggested that I research Generative Adversarial Networks. These networks were producing amazing high quality pictures.

Fig 1. GAN generated celebrity faces by NVIDIA

I was interested to find out why music generation with these models were far less impressive than photo generation. I found a GAN that was , in my opinion , the best music generator called MuseGAN. MuseGAN was created by Hao-wen Dong and the code is open source and available to the public. I got in contact with Hao-wen and began learning the architecture and dataset of MuseGAN. This paper explores my music model, and decoder inspired by the MuseGAN model for creating drum samples.

II. PROBLEM SET EXPLORATION

A. Problem

I have been producing music for over 10 years, and throughout that time I have constantly ran into writers block. It is a common experience in all artistic pursuits. Generative Adversarial Networks, once trained can produce large volumes of music much faster than any human would be able to. These produced musical samples fall short of what an experienced musician could create. However, they do offer artists hundreds of fresh never heard before songs, no matter the quality. These songs could be used for artists suffering from writers block as a starting point for a new original song. This type of human computer interaction is called AI assisted creativity. Sony, Google and IBM have all done research in AI's ability to help in creative fields like, script writing, music production, lyric production, painting generation, and many others.

I believe that as models get better and the machine learning progresses, AI assisted creativity will be an extremely valuable tool for artists of all disciplines in the near future. I will be exploring how to best create an AI assisted creativity model for producers and drummers. This model will be able to create large volumes of drum tracks to spark inspiration from producers and drummers.

B. Other Machine-Learning Based Implementations

Google has released Magenta, a AI assisted creativity music platform. Their goal is to "Allow people to produce completely new kinds of music and art, in much the way that keyboards, drum machines, and cameras did." Magenta is able to generate songs, images and other materials using TensorFlow, googles machine learning language.

Magentas applications vary greatly, one very interesting application is called Nsynth. It combines machine learning and sound synthesis, allowing for an entirely new way of generating sound. The project contains 305,979 different musical notes. Each note has it's own pitch and is played at 5 different velocities. They use machine learning to combine these different sounds in ways never before done. Using the software you can create a whole new instrument by combining a flute with a guitar. Or even a keyboard with a cat's meow. Nsynth is one great example of the possibilities with AI assisted creativity.

My project is focused on a specific type of machine learning network called a Generative Adversarial Network these have also been explored for generating music. Other models have also been extremely successful, like WaveNet a program created by the DeepMind research team. It generates raw audio wave files one sample at a time. Raw audio is often not generated because it doesn't scale very well. 16,000 samples are often required for just one second of content. The samples created by Wavnet are the most compelling in terms of music. If processing power continues to grow I believe full songs generated by Wavenet could make it on the radio.

The draw back of Wavenet is that once the sample is created it can't be tampered with at all, it is the end and beginning of all creative processes. This is why it is great for generating music, but falls short of assisting creativity in music. This is why I chose Midi files as my output format for my GAN network.

C. Generative Adversarial Network

Various neural networks have been used for generation of outputs. These networks include NADE, MADE, PixelRNN, GANS and more. Most agree that Generative Adversarial Networks (GANs) produce the best generative results so far. This section will describe the GAN model, how it trains, and it's fail states.

Model

GAN networks consist of two separate networks, the discriminator, and the generator. Both are in constant competition with each other, hence the adversarial part of GAN. The generator produces outputs which are classified by the discriminator as either real or fake. The generator tries to improve until it can trick the discriminator into thinking it is a real image.

The generator starts from input noise, and uses the gradient decent of the discriminator to improve itself to become more "real". Or at least more real to the discriminator. The random point from which the generator produces something is called the latent space. An interesting quality of the latent space in a GAN generator is that; a direction moved in the space varies output in some way us humans recognize as an important quality. For example, in a GAN used to generate cat images, a direction in the latent space related to the cat's color, or size in picture.

Training

Training for GANs are notoriously hard, we want neither network to dominate the other. An equilibrium or straddle must be found, this would be where the discriminator is right only 50 percent sure if the generated pictures are real or fake. This is called the Nash equilibrium. The GAN plays a min max game where both the generator and discriminator fight over the same number, one wants the number to be high, one wants the number to be low. In the GAN's case the number being fought over is the error rate of the discriminator. The cost function is described by Ian Goodfellow, the inventor of GANs, as the "cross entropy between the discriminator's predictions and the correct labels in the binary classification task of discriminating real data from fake data."

Failure modes

The GAN can run into several different failure modes. they fall into two categories, one can be thought of as the discriminator winning and the other can be thought of as the generator winning. Mode collapse fig2. is where the generator finds a particular way to fool the discriminator. It therefore begins to create a very similar set of images that always fool the discriminator. Causing the generation of samples that have too little diversity in them.

Another failed mode is when the discriminator becomes too good at determining if something is real or fake. This state causes there to be no meaningful gradient for the generator to learn off of, training of the generator will slow down dramatically or stop completely.

I encountered both failure modes while training my GAN network and have sections dedicated to choices I made on preventing these failure modes

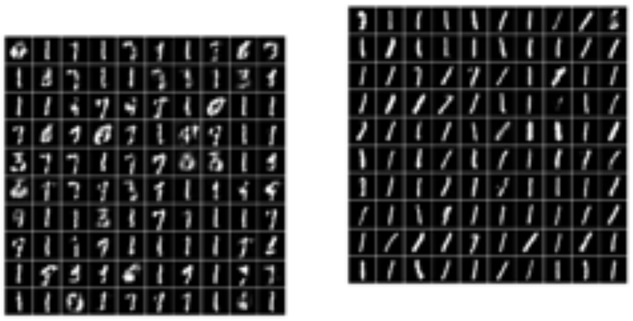


fig 2. Example of Mode Collapse in MNIST GAN

D. Music background

Music is an art form that is entirely dependent on time, and can be represented in in various forms. Sheet music, guitar tabs, and piano rolls are all possible representations of musical data and they all follow a similar pattern. The are represented on a 2d surface. The X-axis being time with the Y-axis being pitch numbers, the higher the pitch the higher the musical tone.

The array of musical information used in MusicGAN follows the same pattern. Each section of music is composed in an array with the first dimension being time steps and the second being number of pitches possible. The array is then cast into a 2-d binary image referred to as a piano roll fig 3. This piano roll image is fed into a neural network as an image. Convolutions are preformed on the image in an attempt to learn the patterns that exist within.

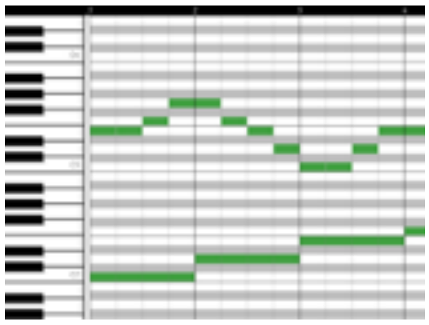
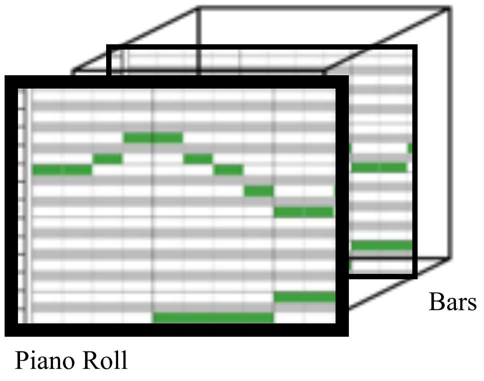


fig 3. Piano roll

A bar of music is a specific time section of the song determined by the measure. The majority of songs written today are in the 4/4 time signature, meaning 4 quarter note counts per bar. All of the music evaluated this project are in 4/4 time. The bars are the dark grey vertical lines on the piano roll example fig 3. Songs are usually structured in a predictable number of bars. A common formula would be Intro: 4 bars, Verse: 16 bars, Chorus:8 bars. A common practice would be for a Rappers to get paid for 16 bar verses when featured on songs.

The discriminator and generator and in competition to find the patterns within the 2d image and identify or recreate them. My final model represents the data in a 3 dimensional space where the depth dimension is bars.



E Data

The original dataset was taken from Hao-Wen Dong's MuseGAN project. The data was taken from the Lakh Pianoroll Dataset(LPD), which contains 174,154 midi track piano rolls. They gathered 21,425 piano rolls in the "Rock" genre and cross referenced them with the million song database. This data set is structured into an array with dimensions [21425, 6, 4, 96, 84]. The array can be reshaped into [21425, 6, 384, 84] and will represent [Sample number, music bars, time steps, pitchnums]

A typical drum sample that exists within this data set is a 2d binary matrix that corresponds to a piano roll looks like. fig 5

F MuseGAN

MuseGAN has three different architectures; the jamming model, composer model, and hybrid model. The jamming model has 5 independent musical elements, each with their own generator and discriminator. The composer model has one single generator creating all music tracks, and one single discriminator. The hybrid model is a combination of the two, each track has a shared vector and it's own generator and a single discriminator.

Since music is usually longer than just a single bar, there were two solutions for creating longer phrases of music. Generation from scratch, a vector stores the temporal information and generates bars sequentially. This is like growing the generator in another dimension of time each with a bar step. The second method use a track conditional generation. It uses a track provided by a human to receive temporal information and fill in the rest of the song. Since the condition generation has high dimensions an encoder is trained to map the time vector of the user provided track with the time vector of the generated one.

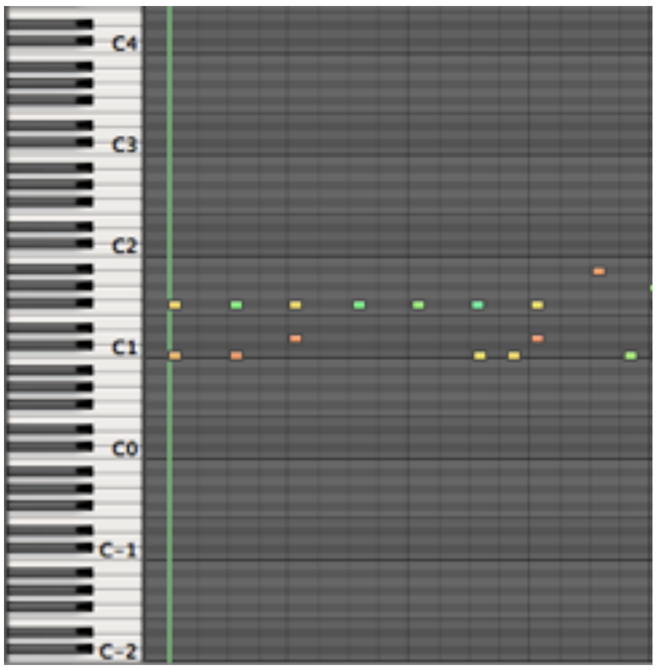


fig. 5 Drum track from sample 715 from MuseGan data set

MuseGAN generates full songs using this piano roll technique. Piano, bass, drums ,strings, and more each have their own piano roll which is combined into a single array. Then reconstructed into a midi song.

H Improvements to MuseGAN for Drums

MuseGAN's results were impressive but are still far from fooling human ears. I believe the network had issues recreating music because the project was very ambitious with the number of tracks being played at once, which the creators were well aware of "It is our goal to avoid as much as possible such simplifications." (referring to the number of tracks played at once as well as other simplifications) [3]. Another difficulty was that the data sets sparseness made it hard for convolutions to pick up patterns and relationships between pixels spaced far apart.

In a deep convolutional network, a convolutional window travels along the image to produce a feature map. These feature maps are pooled together and further convolutions are preformed on them.

We can treat every spot within the 2d piano roll space as a pixel in a picture. Then convolutions will detect relationships with pixels that are close enough together to fall into the same convolutional window. Therefore pixels close together are more closely related, and patterns in their configuration can be discovered more easily. That is not to mention that after multiple convolutions relationships between sections of the pictures a great distance apart cannot be recognized. However, a relationship between a single pixel with another on opposite ends of the picture will be difficult to detect. This is the major issue with viewing these piano rolls as pictures.

III. CLASSIFIER IMPLEMENTATION

A. PreProcessing

The original dataset was taken from Hao-Wen Dong's MuseGAN project. The data was taken from the Lakh Pianoroll Dataset(LPD), which contains 174,154 midi track piano rolls. They gathered 21,425 piano rolls in the "Rock" genre and cross referenced them with the million song database.

Data

- Use *symbolic timing*, which discards tempo information (see here for more details)
- Discard velocity information (using binary-valued piano-rolls)
- 84 possibilities for note pitch (from C1 to B7)
- Merge tracks into 5 categories: *Bass*, *Drums*, *Guitar*, *Piano* and *Strings*
- Consider only songs with an *rock* tag
- Collect musically meaningful 4-bar phrases for the temporal model by segmenting the piano-rolls with *structure features* proposed in [1]

I further pre processed this 21,425 piano rolls using several methods until finally settling on [10,715] midi piano tracks of drum only samples.

Using a custom decoder I was able to dramatically reduce the number of samples needed to represent a 4 bar drum loop. By doing this I was able to dramatically save on computation time and network simplicity. These improvements allowed for the network to generate better samples. The decoder worked in two steps quantization and drum mapping.

Quantization is referring to midi is an operation of fixing all notes hit in a song to fall into a specific grid. Notes can only be played at specific time slots which are defined by the percentage of a whole bar of music they are. For example, the original data was quantized to 96th, meaning that the musical bar was split into 96 different sections, these sections contain all the notes played during that bar. MuseGAN creators did this so that the musical timing of triplets could be included. However, triplets rarely occur in popular music and add a a large amount of data to the data set.

I decided to re-quantize the data to 16th notes, lowering the dimensionality of the final dataset by 83%. Fig2 shows the piano roll data before the quantization and Fig 3 shows it after.

The final step was to create a drum map hash table that simplified the number of notes needed to be kept track of. A midi drum track has various different instruments to choose from that exist on piano keys. Unlike piano, or other melodic instruments played on midi, the piano note does not correlate to the pitch. As different piano notes are hit, different drum noises are triggered. There are kick drums, snares, high hats, etc. The difference between one kick drum sound vs another does no make a difference in the beat the model will be identifying and generating. Therefore, I restricted the allowed drum noises to one option for each drum element. I chose the most common seven rum elements; kick drum, snare, closed high hat, open high hat, crash, tom, and percussion. Since percussion embodies a wide variety of sounds like bells, horns, and triangles, by reducing it to one sound I will be losing some musical data. However, it is the least common used out of the select drum elements and it's rhythmic quality will still be kept.

By removing the ability for the midi to represent different kick drum sounds I am able to reduce the pitch dimension of the sample array from 84 to 7. The final size 7 array ended up looking like

index	0	1	2	3	4	5	6
instr.	hat	kick	snare	crash	open	tom	perc

Since all the midi notes are binary values, there are a limited amount of possible states. These states can be thought of as a permutation of 2 values over the number of pitches. A similar approach was used by Keunwo Choi in an LSTM , the model was trained on midi drum files from the band Metallica. An example of this can be shown on a random sample from my decoded data set.

[1 , 1 , 0 , 0 , 0 , 0 , 0]
[0 , 0 , 0 , 0 , 0 , 0 , 0]
[1 , 0 , 1 , 0 , 0 , 0 , 0]
[0 , 0 , 0 , 0 , 0 , 0 , 0]
[1 , 1 , 0 , 0 , 0 , 0 , 0]

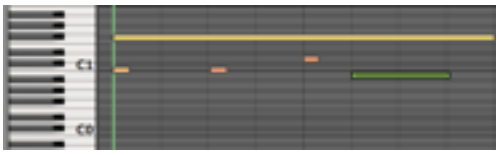
This can be rewritten as
[State 1]
[State 2]
[State 3]
[State 2]
[State 1]

The total number of possible states on a this data set would be a permutation of 0 and 1 with a length of number of pitches. N would be the number of possible pitches and r would be the 2 binary options each pitch has

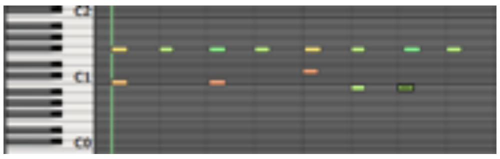
$$P(n, r) = \frac{n!}{(n-r)!}$$

The encoding reduces the total number of states from 6,642 to 42. By creating a custom filter for each state, and using it as a convolutional window, I was able to detect patterns that were computational impossible using the other data set and my GPU. The custom filters were the binary permutations of the possible drum states. I have not heard of a state encoding example using filters like the one proposed in this model. This unique architecture is what I contribute to the success of this model.

Decoding followed the same results backwards until the data was the same as the original [384,82] length array, except for one added step. Since the time steps were decoded so closely together the midi decoder was unable to distinguish when one note stopped and another began, if the note was in the same pitch. This leads to one long note instead of smaller samples.



Before lace with zero array



After lace with zero array

Therefore, I had to add a step of lacing the final output with rows of zeros so the end notes would be easily detected.

B. Model

I originally planned to modify the MuseGAN architecture into something that was suited for my needs. However, after evaluating the dataset, and the training time required to train MuseGAN it was clear that the pre processing, and architecture changes I wanted to make would be difficult to test. So I decided to create a bran new simple GAN from scratch to test my theories on how to improve MuseGAN.

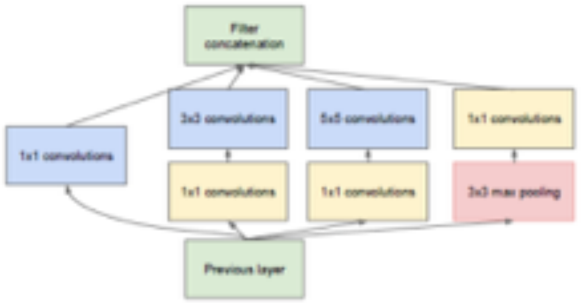
Learning from past projects my approach was to start very small and incrementally build the complexity of the GAN. Several different models were chosen and evaluated based on their generated midi tracks.Because of the subjectivity of the sample sizes rhythms were ranked on a 1-10 scale by two separated people and averaged. The network with the highest score was kept and improved further.

My approach was to start from elementary networks and build up from there, recording the progress as the networks got more complicated. I started with a simple network of fully connected layers, which resulted in hectic music that didn’t contain the patterns expected in drum music.

The second network, like MuseGan viewed the mid information as an image. Using this image a deep convolutional generator and discriminator was built. This is where I used the custom filters and saw a major improvement, however drum results failed to be consistent from beginning to end. To solve this issue I reshaped the data into a 3d matrix where depth was a section of 16 time steps. The new data became a [4,16,7] 3d matrix and I performed 3d convolutions on this matrix.

The drums samples created by this model were very consistent through the entire output. These drum samples were very close to actual drum samples played by musicians, some were indistinguishable. However, some were too frantic and particular patterns would show up in these outputs over and over, signaling mode collapse.

To help the discriminator recognize the mode collapse I decided to add some extra layers and use the GoogLenNet architecture. The architecture won the 2014 imagine competition, all with using 12x less parameters than the previous winner. I picked this because I should be able to train on my GPU and that it would boost the discriminators ability to distinguish real and fake samples.



The GoogLeNet architecture performs multiple different filter sizes and convolutions on the same image, before combining them together.

After adopting this architecture my discriminator become too powerful and I entered a fail mode. This caused an issue with vanishing gradients in the generator. I basically jumped from one failure mode to another. Doing some research I found that by using a Wasserstein loss function in my discriminator I was able to solve this mode collapse. However, I did sacrifice on training time.

This helped total mode collapse, but the diversity of my outputs were still not exceptable. I am currently trouble shooting how to remove a particular pattern that the generator heavily favors. It will appear as a section of a generation or sometimes the majority of the generated drum loop.

IV. RESULTS

My results have been compiled into a file library of generated midi files, that were evaluated using a survey of questions aimed at determining how close the generated samples are to an actual drum track. Real drum tracks where filtered into the list of 10 midi samples each person listened to. The survey had 3 real drum samples and 7 samples generated by my program. They were rated from 1-10 on sounding like an authentic drum beat, and enjoyable to listen to. Then the averages were recorded at put into a table.

Drum	Average score: sounds like drum beat	Average score: enjoyable to listen to
Fake	7.8	7.5
Real	9.6	9.2

I believe this shows the effectiveness of the model in it's ability to generate realistic samples for musicians to start from. The drum sample generate by the program often contain unique elements hard to re create. I strongly believe musicians can use this as a tool for creating no unique sounding songs.

I put this to the test and created a track using un-modified samples from the program, and one using modified drums samples. Both songs are entirely unique from my musical style and I do not believe I could create them without AI assistance. I will release both on my Soundcloud account.

V. CONCLUSIONS AND FUTURE DIRECTION

A. Conclusion

With encoding and decoding the reduced dataset, GAN models are more likely to produce more realistic drum rhythms. This can be attributed to the simplification of the model output as well It is inconclusive wether this will work as well for melodic mono- pitch instruments like bass, or melodic poly pitch instruments like pianos playing chords. I believe that the the lest amount of dimensions present in the training dataset while maintaining the musical qualities of the track are best for convolutions. When using musical models using 2d imaging and convolutions would be to get the pitch data as close together as possible, as well as data in separated time steps. This is why the 3d model of musical data is more effective in finding patterns in notes spaced far apart in timesteps, or on separated bars.

The 3d model combined with the state filters allowed the model to recognize the data more clearly in both the generator and the discriminator. I am unsure if the custom state filter model created from the possible permutations of states will scale well for different instruments with a larger number of possible states.

This tool was used effectively to assist the creative process of two separated songs. Hundreds of thousand of unique drum samples able to be generated in a fraction of the time it would take for a human to create. This data base is the first step in the process of creating a fully functioning application for musicians and producers .

B. Future Direction

In the future I plan to expand upon this model and include Bass, Chords, and Lead parts of a song. I plan to create a separate model for each instrument with their own data sets and architectures specific for detecting patterns specific to their instruments. After each GAN is trained they will combined one at a time to form a committee. The next GAN I span on creating is a bass gan. The bass is also known to be the fusion of rhythm and melody bridging the gap between the drums and melody of a song.

I will explore the custom permutation filters for a GAN trained entirely on bass midi samples with the sample time step dimensions and the drum samples. The number of permutations will be a lot higher than the encoded drum samples and this will be a test of the scalability of such an approach.

I would also like to include a graphical user interface that allows the cycling and saving of individual tracks. This would allow greater versatility for musicians using the application, and would eventually lead to the application learning the what the musician liked. I will finally pitch this final product to software companies that specialize in production.

REFERENCES

1. Joan Serrá, Meinard Müller, Peter Grosche and Josep Ll. Arcos, "Unsupervised Detection of Music Boundaries by Time Series Structure Features," in *AAAI Conference on Artificial Intelligence* (AAAI), 2012
2. Goodfellow, et al. "Generative Adversarial Networks." [1402.1128] Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition, 10 June 2014, arxiv.org/abs/1406.2661
3. D., H., H., Y., L., & Y. (2017, November 24). MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. Retrieved from https://arxiv.org/abs/1709.06298
4. C., K., F., G., S., & M. (2016, April 18). Text-based LSTM networks for Automatic Music Composition. Retrieved from https://arxiv.org/abs/1604.05358