Generating Videogame Music Using Deep Auto Encoders and Long Short-Term Memory Recurrent Neural Networks*

1st Jared Frazier
dept. Computer Science
Middle Tennessee State University
Murfreesboro, Tennessee
jf5s@mtmail.mtsu.edu

2nd Romario Nan
dept. Computer Science)
Middle Tennessee State University
Murfreesboro, Tennessee
rrn2h@mtmail.mtsu.edu

3rd Ethan Lawing
dept. Physics and Astronomy
Middle Tennessee State University
Murfreesboro, Tennessee
eml3y@mtmail.mtsu.edu

4th Deven Kennedy
dept. Computer Science
Middle Tennessee State University
Murfreesboro, Tennessee
dak3s@mtmail.mtsu.edu

***Abstract*—Two different networks were implemented for the generation of videogame themed music: deep autoencoder (DAE) and long short-term memory recurrent neural networks (LSTM-RNN). Musical instrument digital interface (MIDI) videogame music from the Nintendo Entertainment System (NES) game Castlevania is used to generate music using DAE or LSTM-RNN. The quantitative evaluation of both networks is still in progress; however, both DAE and LSTM-RNN were able to produce melodies that were qualitatively distinct yet musically similar to Castlevania MIDI input.**

***Index Terms*—autoencoders, long short-term memory, recurrent neural networks, music generation**

I. Introduction and Background

Can the machine be a force of creativity? Can it take the very heat of music and learn from it, to create something new? These were the questions asked in the conception of this project. Our network is a generative model. It has been trained using examples, which it then attempts to make something akin to, but not identical to. The more complex and interesting original ideas can be generated by neural networks, the wider of an application they will have. If neural nets can write music, then what can't they do? Such is our goal. We aim to train a Neural Network that writes original music.

II. Methods

A. Dataset

Thousands of musical instrument digital interface (MIDI) formatted files from video games produced by the consumer electronics company Nintendo are available online through VGMusic [1]. Of these files, 177 files from the Nintendo Entertainment System (NES) game Castlevania were used as the dataset to produce thematically similar songs.

By default, files of MIDI format cannot be used to train neural networks [2], [3]. Several methods are available for processing MIDI files; however, the music21 library for Python was used.

For this project, only single-instrument polyphonic music generation was attempted. Thus, only the musical notes (singular pitches) and chords (simultaneous aggregation of notes) from piano melodies in a MIDI file were extracted. Each piano melody from a given song was normalized by transposing (uniformly moving pitches up or down) to the musical key of C major.

The number of unique notes/chords over the entire dataset was found in order to frame the task of music generation as a multi-class classification problem. There were a total of 1759 unique notes/chords in the data set, so a one-hot labeled vector was used to represent a given note/chord.

B. Deep Autoencoder

Autoencoders are artificial neural networks capable of learning so-called dense representations (also known as codings or latent representations) [4]. The autoencoder is composed of two distinct parts, encoder and decoder, which each ablate input and then reconstruct it respectively. Autoencoders are an unsupervised learning method since there are no explicitly desired targets, and therefore no labels are provided to the network during training or testing. Thus, an autoencoder may be used to generate creative content such as music since the autoencoder learns an identity function that is characteristic to the problem set [5].

As with other neural network architectures, autoencoders that have multiple hidden layers are more capable of learning complex latent representations. A caveat for building a deep autoencoder (DAE), however, is that a more powerful DAE may learn to map a single encoded input to a single arbitrary number and the decoder simply reverses the mapping.

For our project, DAE consisting of three hidden layers (encoding, bottleneck/latent layer, decoding) was used for the generation of music. As is in the literature, tanh and softmax activation functions with categorical cross entropy loss and RMSProp optimizer were used as hyperparameters [5]. The architecture for this is shown in Fig. 1.

C. Long Short-Term Memory Recurrent Neural Network

Recurrent neural networks (RNNs) are unlike feedforward neural networks in that RNNs possess connections pointing both forward and backward in the network [6] Vanilla RNNs notoriously suffer from the vanishing gradient problem, which prevents weight-updates in earlier layers of multilayer networks. The resolution of this problem via long short-term memory (LSTM) architecture has since made LSTM-RNNs ideal for many tasks including music generation [3], [7]. Since musical melodies are inherently contextual—that is they rely on the context of previous notes to produce a coherent melody—some recursive update of weights is a necessity. LSTM-RNN extends the memory of a traditional RNNs by using memory cells that are capable of "remembering" information in more than just the immediately previous layer [2].

The network architecture was constructed based on several previous reports [3], [8], [9]. However, most reports used classical music (a much more well-investigated area in general) as the basis for their music generation. The LSTM-RNN architecture is available in Fig. 3. The tanh activation function was used for hidden layers and softmax (with categorical cross entropy loss function) was used for output. The optimizer for this study was RMSProp. Compared to other reports, the LSTM-RNN in our report utilizes a shallower network.

D. Hyperparameter Tuning

Tuning activity for music generation is inherently subjective as there is no pre-existing evaluation measure for the quality of music generated [5]. Thus, random input as a "starting point" for a model was sampled from a test set of chord/note sequences and then used to generate a completely new piece of music. The quality of the piece was evaluated based on how tonally similar the generated piece was to the sampled test input. In agreeance with previous music generation reports, the depth of both models was decreased or increased manually to observe the effect on musical quality [10], [11].

E. Performance Metrics

We will be using an evaluation toolbox developed by Li-Chia Yang and Alexander Lerch [12]. This toolbox has an immense array of metrics to use that can be classified as either an objective or subjective metric. The subjective metrics are specialized for comparing two sets of data (i.e. comparing generated data and test data). The most notable concepts of these metrics are the computations of the intra-set and inter-set distances, the Kullback-Leibler Divergence (KLD), and the overlapped area (OA).

In order to find the two different types of distances a "pairwise exhaustive cross-validation is performed for each feature." I will also quote Yang and Lerch to explain what intra- and inter-set distances are: "If the cross-validation is computed within one set of data, we will refer to it as intra-set distances. If each sample of one set is compared with all samples of the other set, we call it the inter-set distances." The KLD and OA are similarity measurements. In order to explain what they show, imagine we have sets of generated data from two models, M1 and M2, and the training data. We calculate the intra- and inter-set distances for each set of data. Then we compute the KLD and OA for the inter-set distances of the data for M1 and the training, and we do the same for M2's data and the training data. The M1/training KLD was smaller than the M2/training KLD. The M1/training OA was larger than the M2/training OA. This implies that the generated data from M1 is more similar to the training data than the generated data from M2. If all the data were music, that means the music M1 generated is more similar to the training music than the music from M2. That could mean the music from M1 is more human-like than the music from M2.

The objective aspect of this toolbox takes advantage of the varying parts of music that make a full song. Take for example the pitch count and note count for a song. The pitch count is the total number of different pitches used in a song. If a song used the pitches C, F, and G throughout the entire song, it would have a pitch count of three (testing will help check if it actually comes out like this). The note count is the total number of notes used. If that same song used C four times, F one time, and G three times, it would have a note count of eight. Now that we have these varying musical structures, we can perform the computations mentioned above on specific aspects of songs and visualize more accurately where different data sets are outperforming others, or possibly where they both struggles.

III. Results and Discussion

A. Qualitative Analysis

In the case of music generation, sequences of notes/chords with uniform rhythm were successfully generated using both the DAE and the LSTM-RNN. The DAE was able to produce a melody with greater frequency of polyphony (multiple notes at the same time) as well as generate simple harmony. Harmony, in this sense, means the combination of notes to produce a pleasing, non-dissonant sound. In both the DAE and LSTM-RNN generated music there is some local sense of coherence in the music. This can primarily be attributed to constraining the output of both networks to notes/chords in the key of C major. Despite this, Castlevania's music is known for its unique dissonant sounds, that is notes and chords that exist at what could be described as clashing musical intervals (e.g., B natural and B flat played simultaneously produce a dissonant sound). As a result, the music depicted in Fig. 2. shows that the LSTM-RNN produced melody is much less coherent in terms of harmony than that of the melody produced by the DAE in shown in Fig. 4.

Additionally, since a relatively small portion of a Castlevania song was used as input due to large breaks in piano

melody, both the DAE and LSTM-RNN generated songs have a mechanical sound to them and do not have a distinctive melodic beginning or ending.

For both models, overfitting was qualitatively apparent when deeper models were built. For example, for the DAE, a single hidden encoding layer and a single hidden decoding layer were required in order to prevent the direct copying of input to output. If more hidden layers for DAE were included, the DAE was unable able to learn the so-called latent representations in the input that lead to the musical quality of the output. The Castlevania dataset itself may account for the limited generalizability of both models and low quality musical representations since nearly all the piano songs that could be sampled had only short, repetitive melodies. The melodies in the dataset lend to musical quality due to the context of other instruments, but since only single-instrument polyphonic music generation was attempted for our study, we lost musical quality. Deeper, wider networks would very likely improve the models; however, with the limited sample size, musical quality tended to diminish overall with such hyperparameter changes.

B. Quantitative Analysis and Future Work

Quantitative analysis of the generated models will be completed using the toolset described in the Performance Metrics methods subsection. Quantitative metrics have been delayed because strong performance with respect to a metric does not necessarily indicate that a piece will be of high musical quality. However, in addition to expanding the size of the network and the datasets, this is a task that will be completed before the demo.

We plan to expand the model size to see if a larger size has better results. We could also use WaveNet for the model instead of LSTM. WaveNet is a system developed by Google to generate data from original data. WaveNet is a generative model, consequently it is well-suited for the task of music generation. While our study used tanh and then softmax for activation functions, we may follow other studies in their use ReLU or use a different optimization function such as Adadelta. Finally, we may use other videogame music from different games (as opposed to all from the same franchise) to train a network on a more varied dataset.

IV. Conclusions

Two different artificial neural networks were implemented for the generation of music in the style of the NES game Castlevania. The stacked autoencoder, while generating more harmonious music, also produced music which was less distinct than its input. In contrast, the long short-term memory recurrent neural network produced more distinct but dissonant music. The musical quality of both generated samples should be categorized as relatively poor to satisfactory. The primary lessons learned from this project are that generative tasks do not have as clear metrical boundaries for evaluation as do other regression or classification tasks. Additionally, data acquisition and wrangling (processing) consisted of a substantial portion of this project. We learned that a major factor in the success of a machine learning project is the quality of the data and how well the problem/task can be defined.

V. Figures

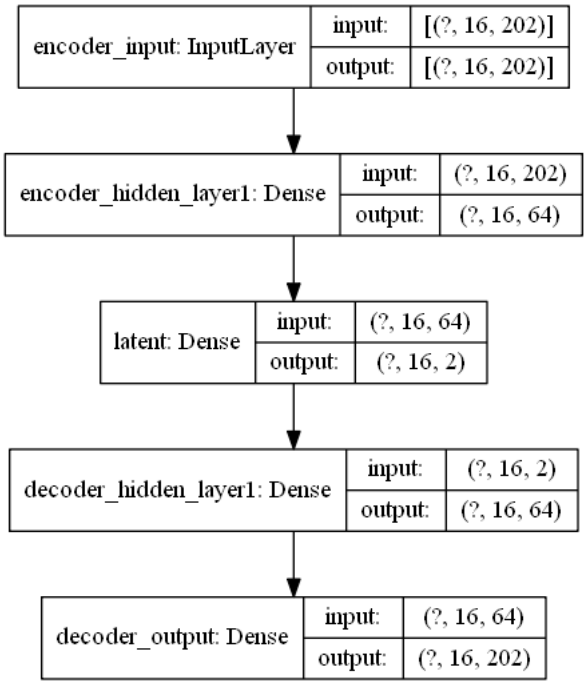

Fig. 1. Autoencoder Model Architecture

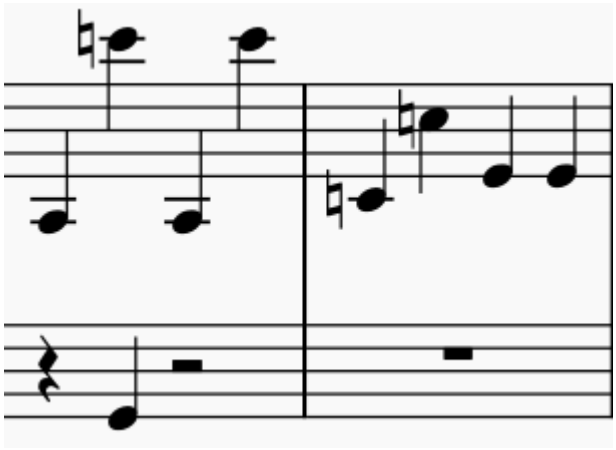

Fig. 2. LSTM-RNN Snippet of Generated Music

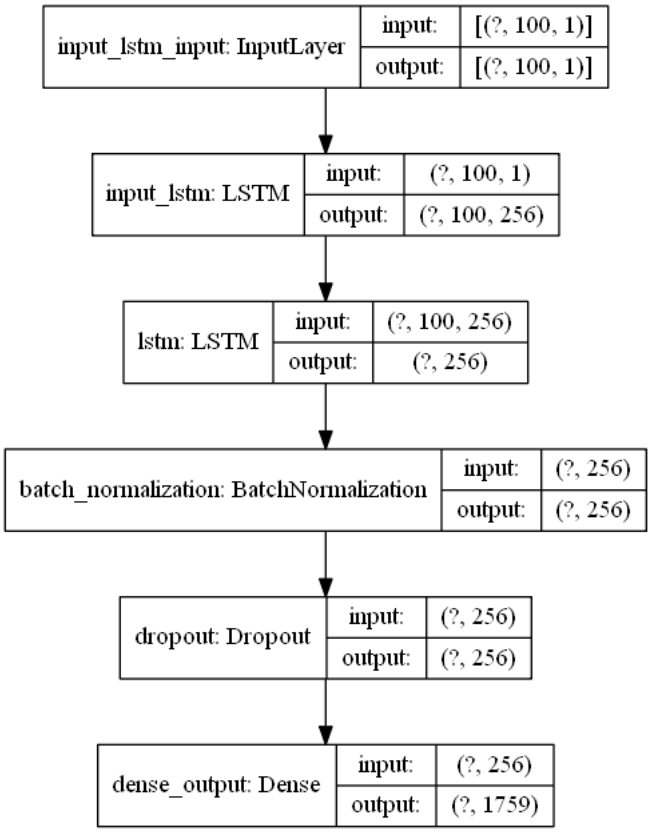

Fig. 3. LSTM-RNN Model Architecture

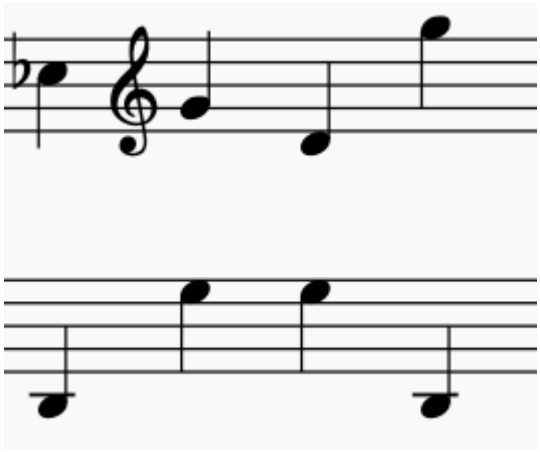

Fig. 4. Autoencoder Snippet of Generated Music

REFERENCES

[1] M. Newman, "Video Game Music Archive: Nintendo Music," VGMusic, 1996. [Online]. Available: https://www.vgmusic.com/music/console/nintendo/nes/. [Accessed 08 March 2021].

[2] S. AlSaigal, S. Aljanhi and N. Hewahi, "Generation of Music Pieces Using Machine Learning: Long Short-Term Memory Neural Networks Approach," Arab Journal of Basic and Applied Sciences, vol. 26, no. 1, pp. 397-413, 2019.

[3] N. Mauthes, VGM-RNN: Recurrent Neural Networks for Video Game Music Generation Generation, Master's Projects, 2018, p. 595.

[4] A. Geron, "Chapter 17: Representation Learning and Generative Learning Using Autoencoders and GANS," in Hands-on Machine Learning with Scikit-Learn, Keras and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd ed., Sebastopol, O'Reilly Media, Inc, 2019, pp. 567-574.

[5] J. Briot, G. Hadjeres and F. Pachet, "Deep Learning Techniques for Music Generation - A Survey," arXiv:1709.01620, 2017.

[6] A. Geron, "Chapter 15: Processing Sequences Using RNNs and CNNs," in Hands-on Machine Learning with Scikit-Learn, Keras and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd ed., Sebastopol, O'Reilly Media, Inc., 2019, pp. 497-499.

[7] S. Hochreiter, "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions," International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 6, no. 2, pp. 107-116, 1998.

[8] J. Svegliato and S. Witty, "Deep Jammer: A Music Generation Model," University of Massachusetts, Amherst, 2016. D. Kang, J. Kim and S. Ringdahl, "Project milestone: Generating music with Machine Learning," Stanford University, Stanford, 2018.

[9] A. Ycart and E. Benetos, "A Study on LSTM Networks for Polyphonic Music Sequence Modelling," in 18th International Society for Music Information Retrieval Conference (ISMIR), Suzhou, 2017.

[10] A. Huang and R. Wu, "Deep Learning for Music," Stanford University, Stanford.

[11] L. Yang and A. Lerch, "On the Evaluation of Generative Models in Music," Neural Computing and Application, vol. 32, no. 9, p. 12, 2018.

[12] A. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior and K. Kavukcuoglu, "Wavenet: A Generative Model for Raw Audio," arXiv:1609.03499, 2016.

[13] J. Ba, J. Kiros and G. Hinton, "Layer Normalization," arXiv:1607.06450 , 2016.