Music Composition using an Autoencoder Network [*]

Jahya Burke[1], Yisheng Ji[2], Shiwei Rong[3], Shuangcheng Yang[4] and Alan Yessenbayev[5]

Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, USA

Email: {[1]j1burke, [2]y3ji, [3]srong, [4]shy003, [5]ayessenb}@ucsd.edu

Abstract

Our team utilized an autoencoder neural network architecture to solve a problem of music composition. We explored the effects of training the network with MIDI tracks from video game compositions developed for different types of hardware, while also observing changes in the hyper-parameters of the network (layers sizes, learning rate, etc.).

Artificial Autoencoder, *Generative Model*

1. Introduction

The power of artificial intelligence and machine learning increased productivity in various fields of human endeavour, while also displaying great potential to assist art creators. We tackled the problem of music composition because it is relevant and generative in its nature. We discovered the autoencoder approach from YouTube content creator CodeParade [4], which captured our attention because it is an unusual approach to analyzing time-series data. Since musical data is usually thought of as time-series data, Recurrent Neural Networks are usually networks of choice for these types of data [5].

For the purpose of this project, we used an autoencoder for music composition. An autoencoder is a type of artificial neural network used to learn efficient data encodings in an unsupervised manner. The autoencoder is trained to represent the output of the network as close as possible to its original input. This reconstruction property allows the network to perform dimensionality reduction by having a bottleneck layer with few nodes. The nodes on the bottleneck layer form a dense representation of the data, since they can be used to reconstruct an input. With linear activation, the bottleneck layer of the autoencoder represents the principal components of the data [5]. Recently, the autoencoder has become wildly used for learning generative models of data.

When autoencoders are applied to music generation, we can train the composer model to learn a dense representation of music and then use the decoder to generate music. To be successful at this, we must establish a solid autoencoder architecture, identify a meaningful dataset and train the network effectively. We need to setup the structure of the autoencoder model first. Addtionally, training datasets are used to get weights for several hidden layers. As a result, we should get a dimensionality reduction autoencoder neural network that can still reconstruct the original input data. In the end, we take the decoder part and use the activation layers as control inputs. The goal is to establish an AI music composition writer that is unique and harmonically pleasant.

2. Methods

2.1. Data

In this project, we downloaded video game .midi files from *VGmusic.com* [6] and used a Python package called *Mido* [2] to translate MIDI files into a 96 notes by 96 ticks square form as our data. Figure 1 shows the form of our data structure.

2.2. Mathematical Setup

Classical music and jazz are more free form without definite structure like verse chorus and bridge. Thus we started on video music since there is a lot of available data online and it is usually catchy and repetitive with strong music structure. We were mostly interested in creating looping music, as autoencoder architecture is powerful for learning patterns of the over all data. In some sense, we hoped that musical snippets of fixed length have a fractal structure that autoencoder will potentially learn.

[*]ECE 271B Project, San Diego, US, 2019

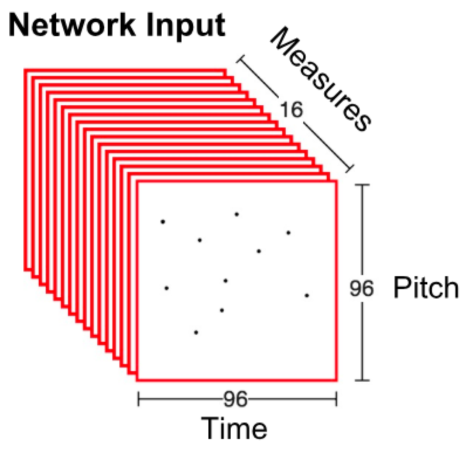


Figure 1: The data structure for the audio sample

The two techniques utilized are the Convolution Neural Network (CNN) and Long-Short Term Memory (LSTM). Since we are dealing with spatially structured data and time-structured data, both of these methods produce inaccurate results on structured music. For CNNs the assumption is made that there is a close relationship between pixels and their immediate neighbors. This is very true in images, but it is not true for piano rolls. The note most correlated with a single note is generally much farther away in the same measure or on different measures, so locality is not meaningful. As for LSTM, it can do great with free form music since you only need to know the recent context to generate the next notes. However, since we are trying to generate structured songs the network needs to know the entire context of the song at the same time. Therefore, we decided to use an autoencoder network because of its ability to learn the underlying structure of a data set. Figure 2 displays our

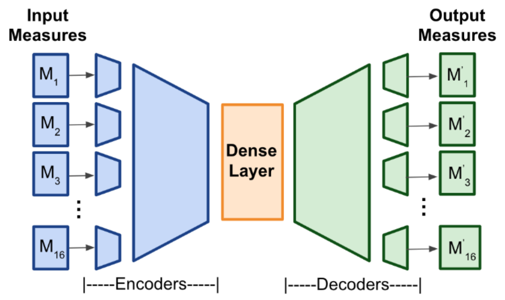


Figure 2: Network Architecture

network architecture. What we ended up doing is creating a dense neural network to encode each measure into a feature vector. Feeding those into a dense autoencoder, which then outputs another feature vector that finally gets, converted back a measure. It is like two autoencoders in the same network. The autoencoder is very powerful generative method in probabilistic modeling of raw audio because it is capable of randomly generating new data that looks similar to the training data. In the following example, we describe our autoencoder structure. This architecture can attain consistent long-term structure and therefore is ideal for our video game music composition task. For encoding and decoding the measures we picked 200 dimensions and for decoding and encoding the song we chose 120 dimensions. These values, which have a much lower dimensionality than the input data, seem to be a good balance. The structure now looks similar to multilayer perceptron. There is input layer, an output layer and 3 hidden layers connecting them. For an autoencoder network, the number of nodes in output layer is the same amount as the input layer and with the purpose of reconstructing its own inputs.

In our autoencoder architecture, at each step a stack of dilated convolutions predicts the next sample of audio from a fixed-size input of prior sample values. The joint probability of the audio x is factorized as a product of conditional probabilities:

$$p(x) = \sum_{i=1}^{N} p(x_i | x_1, ..., x_{N-1}) \tag{1}$$

Our training goal is to minimize reconstruction errors or called loss and find the optimal weights values for each layer inside the autoencoder model. We do a feed-forward pass to compute activation functions at all hidden layers to the end of output layer to get an output vector x'. Next, calculate out the difference or the loss between the input vector x and output vector x' using cross entropy. Then back-propagate the error through the neural net and update each individual layers weight. After we finished training

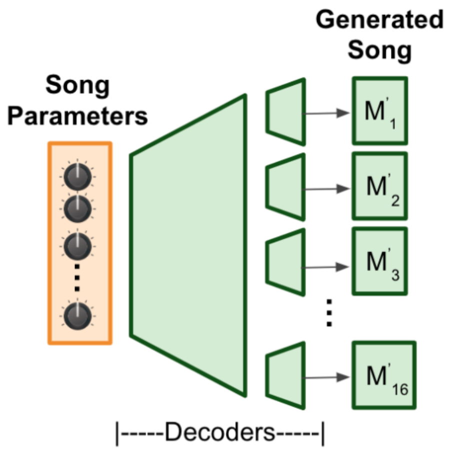


Figure 3: Neural Composer

the encoder-decoder, we use the decoder part to act as the composer and the bottleneck layer acts as the control inputs of our composer, as shown in Figure 3.

2.3. Technical Approach

If the MIDI file consists of multiple track, we mapped all of the tracks. If the file did not conform to our structure, we would pad the audio sample to get to the size of 16 measures.

To implement the architecture we use *Keras* [3] with *Tensorflow* backend [1]. We use binary cross-entropy loss function with RMSProp optimizer of step size 0.001. We use mini-batch gradient descent with the batch size of 200. No dropout is applied during training.

After our composer finished training, we ran various inputs through it to produce music. We trained 3 different networks using 3 different subsets of data (Piano Only, PlayStation 1, and all data). As the decoder produces probabilities, we set a threshold (0.5 in our case). If the probability value is above that threshold, we play the note. We set the tempo, the velocity, and the instrument manually. Our architecture only supports staccato (with each note sharply detached or separated from the others) style of play, as we only registered the ”key down” event in the MIDI file during conversion to piano roll format.

3. Experiments

As was stated previously, the network was trained on 3 datasets: Piano music, PlayStation 1 music and all music. The loss achieved during training can be seen in Figure 4, 5 and 6 respectively. All three networks exhibit similar trends in the loss during training. The training loss continues to reduce across epochs, but the validation loss does not improve significantly from the starting value. This is an indication that we are over fitting our model to the training data. For our application, this will result in some of the compositions performing poorly in regards to uniqueness. This is a result that we noted when listening to resulting music and evaluating it qualitatively as well.

3.1. Insight

One of our goals was for our music to sound harmonic and correct when looped, as this is a characteristic of video game music. We decided to rely on music theory to assess how well the compositions performed when looped. In music theory, the key signature of the composition acts as a declaration of coordinate system. In a sense, we are interested in the distance between the notes and the base note of the key signature (usually the root of the chord played in the base line). For our purposes, perfect unions, fourths and fifths produce the most harmonic transition between chords and notes. [7]. This respectively translates to 0, 5 and 7 semitones (pitch steps) ± 12 semitones (octave is 12 semitones) from the base note of the composition. Usually whatever is being played in the lower pitches is the bass line that keeps the beat of the composition. So if the baseline notes are kept consistent across loops you can note that the composition will produce a harmonic transition. Figure 7 shows

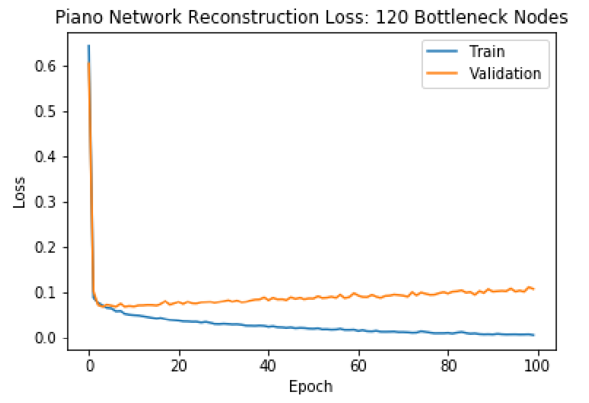


Figure 4: Loss for networked trained with Piano music.

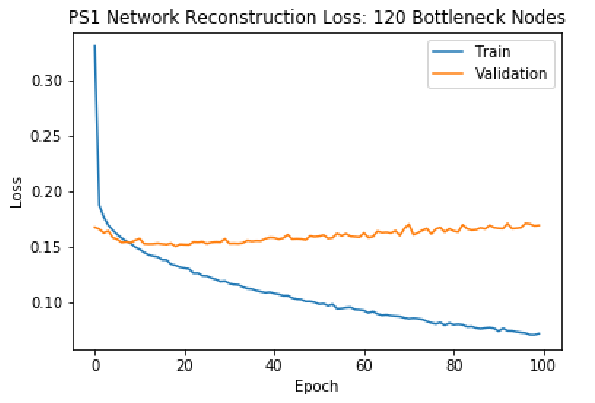


Figure 5: Loss for networked trained with PS1 music.

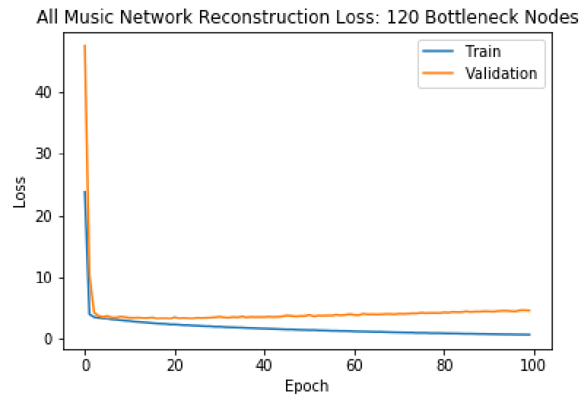


Figure 6: Loss for networked trained with all music.

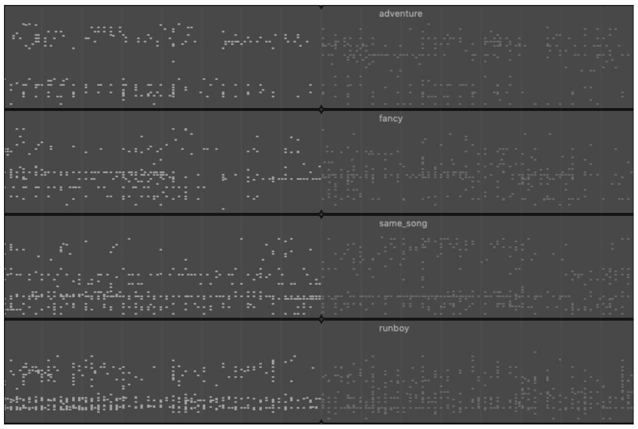


Figure 7: Output Composition Piano Rolls

the several outputs of our network looped on itself. Each row is a different composition. The darker gray points indicate the transition to the repeated portion. We can see that harmonically networks did not learn anything complicated as they aim for unison intervals in the bass line of the compositions. If you look to the lower notes they are consistent in the transition from the first loop to the second loop. We can see that this relationship is maintained for each network that we tested. This shows that our networks were successful in their aim to produce harmonically sound looping music.

3.2. Parameter Tuning

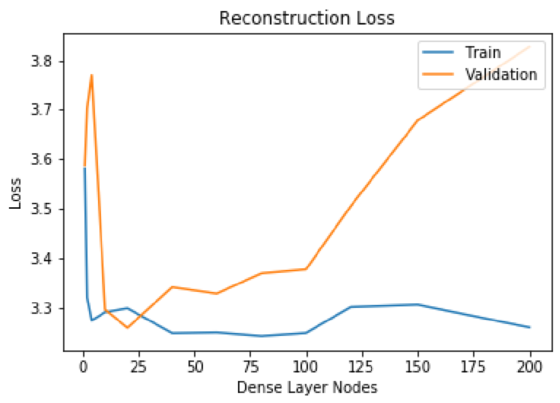


Figure 8: Bottleneck layer loss analysis.

In order to assure the validity of our model structure, we chose to investigate how many hidden nodes were needed to accurately reconstruct the data. The number of nodes in the bottle neck layer reflect the dimensionality of the input space. If too few nodes are used it will result in an inability to reconstruct the data. If too many nodes are used it is a waste of resources. To measure the number of nodes needed in the bottle neck layer we looked at the reconstruction loss of the autoencoder. The reconstruction loss utilizes binary cross entropy loss to measure how well the output of the network matches the input.

Figure 8 shows the reconstruction loss on the validation set and the test set for networks with different number of nodes in the bottleneck layer. It can be seen that the reconstruction loss is high when less than 25 nodes are used or when more than 125 nodes are used. This suggests that a minimum number of nodes in the bottleneck layer is 25. With fewer nodes the network cannot accurately reconstruct the data and thus will likely perform poorly in generating music. In using more than 125 nodes it is possible that there is not enough data to accurately represent such a large feature space. To account for these findings we chose to use 100 nodes in our bottleneck layer. Given these findings, this should be enough to represent the data well.

3.3. Validation

Finally, we decided to run a subjective test to see how people react to different compositions that were presented. We designed an experiment. We created a Google Form and uploaded 6 compositions online, 2 from each subset we trained on. Then we shuffled them and presented them in the survey in an arbitrary order. We asked respondents whether the compositions sound like music, whether the liked them, and also asked them to rate whether compositions were close to completion and how original they were, both on the scale from 1 to 10.

The results of the survey are presented at the last page of the report. Our survey had 31 respondents. For the pair of binary questions, we see that dataset with all compositions has the least appeal to the public with 61% of people reporting distaste and 40% of people reporting that the compositions did not sound like music. The network trained on piano music is the most popular with 59% of people reporting they enjoyed the compositions and 77% of people reporting that the compositions sounded like music. The network trained on music from the playstation 1 followed relatively closely behind the piano music network with 51% of people reporting they enjoyed the compositions and 74% reporting that the compositions sound like music.

Interestingly, one of the outputs of the composer overfitted so drastically, that it ended up being an arrangement of ”Chocobo Theme” by *Nobuo Uematsu*. We decided to throw that composition into the survey as well, as a part of the piano subset. The results of the survey supported this realization. In general people regarded that composition as more musically pleasing but less unique.

The plot for scores show the mean and the standard deviation of the scores received by each subset and the overfitted example. We can see that as the quality of songs deteriorate people call compositions more original and less finished.

The overfitted example is an obvious outlier, being the less original and the most finished. This shows, that subjective survey could detect features that are of interest to us and proves that at this state our tool is more of a music idea generator than full-on composer.

4. Conclusion

This project shed light onto possible solutions that can be implemented with regards to machine-assisted music composition. As we saw, our composer picks up on simple harmonic patterns and plays safe with regards to new musical ideas. It is possible that with further training our composer would start using more complex harmonic intervals, however, there is danger of over fitting.

We see that choosing a song of particular structure (certain subset of data) is better for the quality of compositions, as musical ideas remain grounded and liked by people. Training the network with all data is not that useful as differences in structure and form of compositions wash out appealing details from the produced compositions.

Essentially, this network could be a module in a bigger musical solution, that would help the process of composition take off. Further research needs to head in the direction of adding a recursive component to the network, to encode the temporal component into the structure of the network.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Ole Martin Bjorndalen. Mido - midi objects for python. `https://mido.readthedocs.io`.

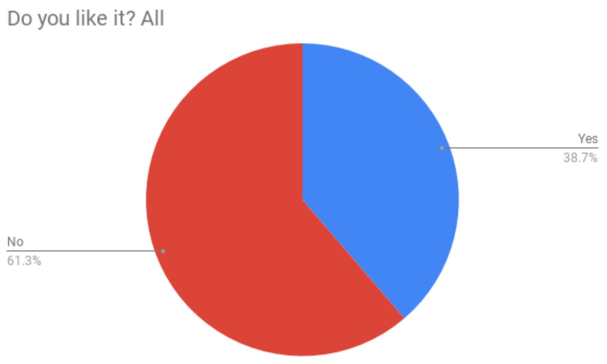
[3] Francois Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

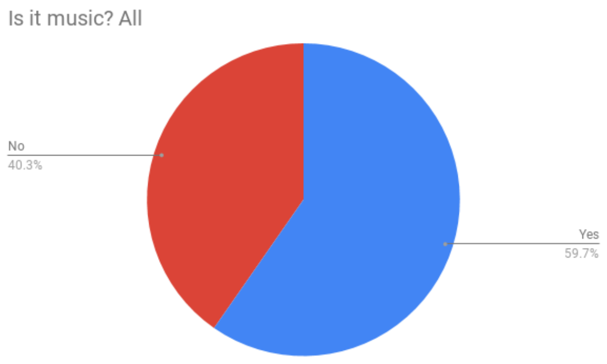
[4] CodeParade. Generating songs with neural networks (neural composer), July 2018.

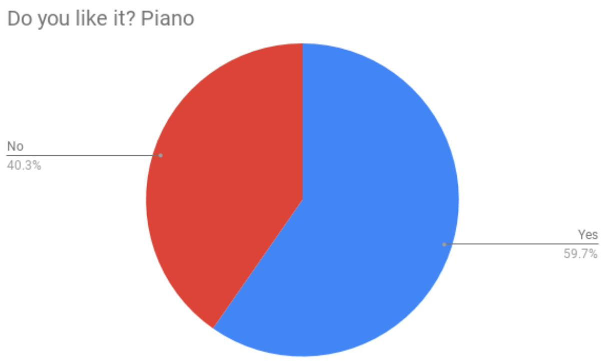
[5] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st edition, 2017.

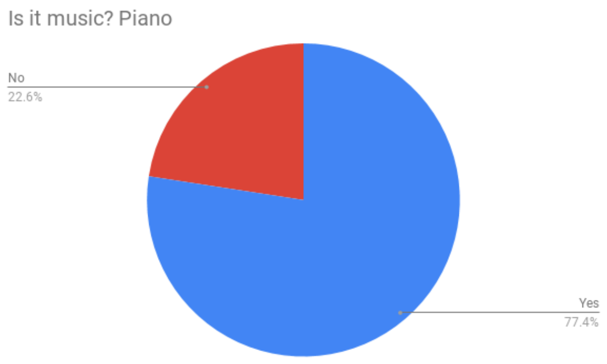
[6] Mike Newman, Shane Evans, Mark Carroll, Jace Hill, and Daylon Camarena. Video game music archive, 2017.

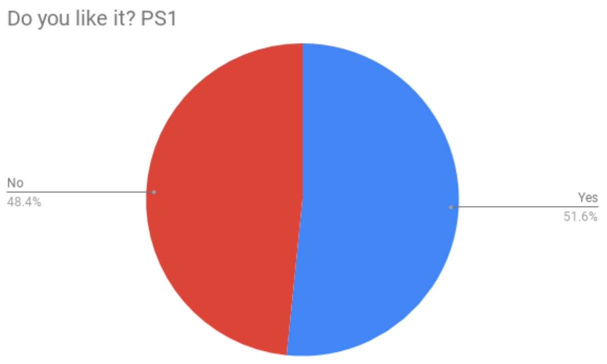
[7] Godfrey Weber. Definition of perfect consonance in godfrey weber's general music teacher, 1841.

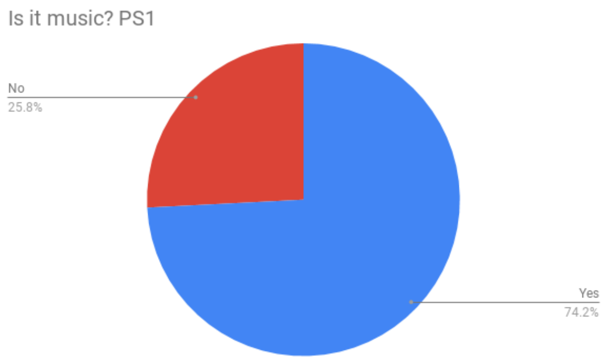

Do you like it? All
Yes
38.7%
No
61.3%

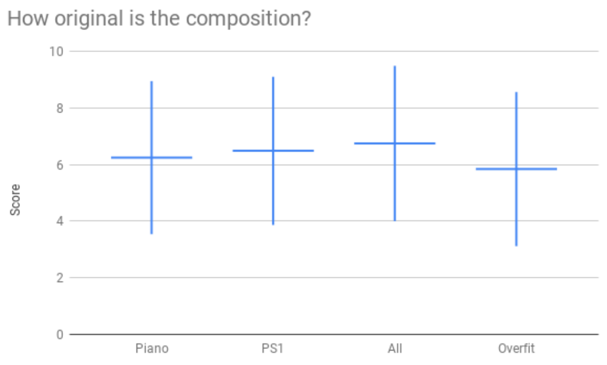

Is it music? All
No
40.3%
Yes
59.7%

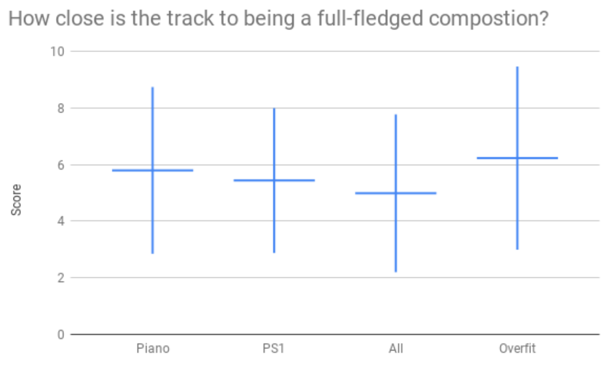

Do you like it? Piano
No
40.3%
Yes
59.7%


Is it music? Piano
No
22.6%
Yes
77.4%


Do you like it? PS1
No
48.4%
Yes
51.6%


Is it music? PS1
No
25.8%
Yes
74.2%


How original is the composition?
Score
10
8
6
4
2
0
Piano
PS1
All
Overfit


How close is the track to being a full-fledged compostion?
Score
10
8
6
4
2
0
Piano
PS1
All
Overfit