Listen to your Data: Turning Chemical Dynamics Simulations into Music

Austin Atsango (atsango) [1] **K Grace Johnson (kgjohn)** [1] **Soren Holm (sorenh)** [1]

1. Introduction

Transforming data into sound (sonification) has shown potential in elucidating patterns that would otherwise be easy to miss.[1] It presents a different way of interacting with and understanding data, and an accessible option for the visually impaired. While there has been work on sonifying data, i.e. directly mapping data values to pitch, generating actual music from data has been largely unexplored.

In this project, our goal is to translate simulation data into a musical form while preserving trends in the data. We hope to generate melodies that both aid in understanding the data and are pleasant to listen to. Specifically, the goals are 1) to generate music, i.e. melodies that are indistinguishable from those composed by humans, and 2) to have those melodies reflect trends in the underlying data.

Simulation data is a natural choice for musification, as both the data and a melody can be considered as some quantity changing in time. In a simple melody, this quantity is the pitch of the note played. In this project, we use trajectories from a quantum dynamics simulation of a small organic molecule stilbene (Figure 1)[2] as it decays from an electronic excited state to the ground state.

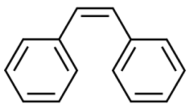

Figure 1. (Z)-Stilbene

We use two approaches. First, we take an unsupervised classification approach and cluster short snippets of a trajectory using a Gaussian Mixture Model (GMM) with the EM algorithm in order to discover motifs within a trajectory, then match these motifs to similar ones from a composed melody. In the second approach, we use a supervised model (either softmax regression or an LSTM RNN trained on composed melodies) to predict the next note in a song, biased by the trajectory values. We evaluate the success of these approaches with a survey designed to assess the two goals of the project: whether the samples generated are musical, and whether they reflect the trajectory data.

[1]Department of Chemistry, Stanford University.

2. Related Work

Sonifying data is certainly not a new concept.[3,4] It has been used for data analysis in a number of fields, including protein folding.[1] There is even an online platform[5] which allows users to upload data and generate sound. Users can select features like pitch range, key signature, and tempo. These sonification processes, however, usually do not result in something which sounds composed, as they are a direct mapping from data to notes in a key signature.

Recently, there have been many efforts to generate music using machine learning, specifically using generative models such as VAEs, LSTMs, and GRUs.[6,7,8] Deep Belief Networks (DBNs) have been combined with RNNs for use in music generation.[9] In these formulations, the goal is to predict the next note of the melody based on the previous notes. There have also been efforts to transfer musical style using LSTMs, e.g. from jazz to classical, or vise versa.[10] A survey of RNNs by Chung et al[11] found GRUs and LSTMs to be the best reccurent networks for polyphonic music modeling.

While there have been interesting developments in both fields discussed above, to our knowledge there has been no significant effort in combining the two. We are therefore interested in extending a generative model to make musical samples based on given simulation data.

3. Dataset and Features

Musical data was obtained in the MIDI format. The full dataset consists of 405 songs: 312 are classical piano pieces downloaded from http://www.piano-midi.de/, and 93 are piano pieces from the Final Fantasy video game, a dataset used by Skuli.[12] A smaller dataset was constructed out of 17 pieces of music by the composer Clementi. All preprocessing was done using the `music21`[13] and `mido`[16] packages. The MIDI format contains features such as pitch, tempo, and rhythm required to successfully decode a song. In order to more easily compare to trajectory data, we simplified the MIDI files significantly.

An example of an initial piano piece is shown in Figure 2. Here, one can observe two streams of note sequences

played by the right and left hands respectively. A single stream indicates what note or chord (a set of notes) should be played at a particular time.

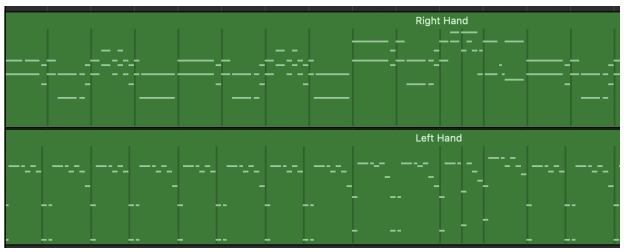

Figure 2. MIDI file representation in GarageBand

We first reduce the MIDI file to a single channel: the right hand of the piano, as this usually contains the melody of the song. Once we have the single stream, we eliminate chords as follows: if several notes are played at the same time in a chord, we take the note with the highest pitch. This leaves us with a relatively simple melody sequence as shown in Figure 3:

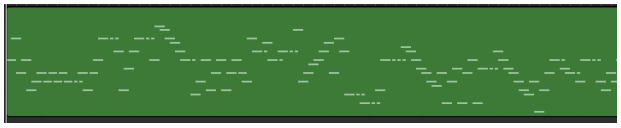

Figure 3. Simplified version of MIDI file

The notes are then mapped to integer values and represented as a 1D numpy array. The integer values are normalized to fall between 0 and 1. As a final step, the 1D musical pieces are split into sequences of 50 consecutive notes. This is achieved by a convolutional approach where a 1D window of length 50 is slid across a musical piece and a 50-note sequence is captured at every step. These 50-note sequences serve as the features while the note that came immediately after a sequence serves as the corresponding label. This process yielded 302407 examples for the full dataset and 8933 examples for the Clementi set. During training, we used an 80:20 training/validation split. Our model was trained to predict the next note given an initial sequence of 50 notes. The length of our sequence (50) was chosen as a compromise between accuracy and computational cost: while longer note sequences are expected to give better predictions, they also require more resources to train. Because the number of notes is finite (with pitch values between 0 and 127), this was treated as a classification problem, and the labels were one-hot encoded.

Chemical dynamics data was generated in the research group of one of the authors,[2] and consists of a set of trajectories with potential energies of the ground and excited state at every timestep of the simulation. Trajectory data is normalized such that the minimum and maximum values match the lowest and highest pitches seen in most songs: 50 and 90 on the pitch scale. While the simulations record events on the femtosecond timescale, we simply represent each timestep as one note. Figure 4 shows an example of a normalized ground state trajectory.

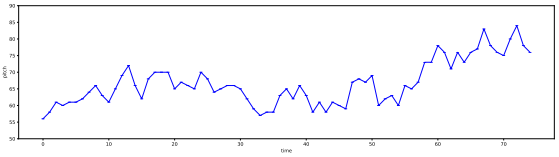

Figure 4. Chemical dynamics simulation of the ground state potential energy of stilbene.

4. Methods

Here, we discuss classification techniques, the music generation process, and our application of the E-M algorithm to sonification.

4.1. Softmax Regression

We used softmax regression as a baseline for note prediction. In this model, the response variable y is assumed to be drawn from a multinomial distribution where each possible y_i value has a corresponding probability ϕ_i. The goal, then, is to maximize the log-likelihood:

$$l(\theta) = \sum_{i=1}^{n} \log \prod_{l=1}^{k} \left(\frac{\exp(\theta_l^\top x^{(i)})}{\sum_{j=1}^{k} \exp(\theta_j^\top x^{(i)})} \right)^{1\{y^{(i)}=l\}}$$

Where $p(y = i|x) = \phi_i$ is given by:

$$p(y = i|x) = \phi_i = \frac{\exp(\theta_i^\top x)}{\sum_{j=1}^{k} \exp(\theta_j^\top x)}$$

4.2. LSTM-RNN

Simple Recurrent Neural Networks (RNNs) are useful for modeling sequences because they are able to retain a memory of previous inputs by looping the output from a previous step (h_{t-1}) back into the network with the current step's input x_t. Simple RNNs perform very well in the short term, but they have problems modeling long-term dependencies in data.

Long Short-Term Memory (LSTMs) are much better suited for handling long-term dependencies. The basic structure of an LSTM cell is shown in Figure 9. LSTMs work by maintaining a memory c_t and using gates (input, output, and forget) to determine what information to store or discard.

First, the forget gate multiplies the LSTM state c_{t-1} by the sigmoid-activated vector below. Values of 0 mean "completely forget" while 1 means "completely keep"

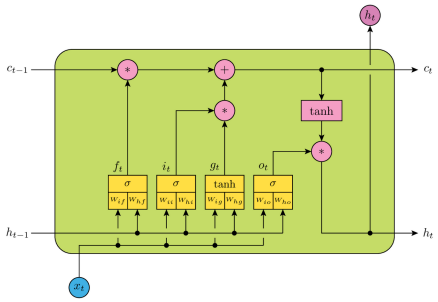

Figure 5. LSTM cell architecture[13]

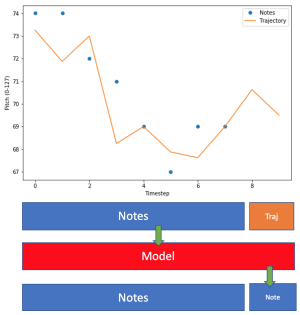

Figure 6. Scheme for generating music based on trajectory

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f),$$

where x_t is the current input and h_{t-1} is the previous output.

The input gate then decides what new information to add to c_{t-1}. This gate contains both a sigmoid layer and a tanh layer whose product is added to the LSTM state.

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ C_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_C) \end{aligned}$$

Finally, the output gate decides the output h_t . The output value depends on the hidden state h_{t-1}, the input x_t, and the current LSTM state c_t. It is evaluated as follows:

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= o_t \times \tanh(C_t) \end{aligned}$$

4.3. Music Generation

With the neural networks trained to generate music in a certain style, we needed a way to bias the music generation towards fitting the trajectory data. This required that we found a way to influence the prediction of the network at prediction time. The RMS distance between two consecutive notes in our dataset is only 3.80 in the pitch space of 0 to 127. It is likely that the trained model will have learned this relationship, and we can exploit this to push the predicted note towards the trajectory by giving a trajectory value as the last element of the input sequence. Ideally, the network would come up with a compromise note that is close to the trajectory but still musical in accordance with its training.

To start the music generation, we picked a random training example from the validation set with the last note exchanged for the first point in the trajectory data. The sequence was then fed into the trained model to generate the first note. The music generation was propagated as illustrated in Figure 6, by concatenating the last 49 predicted notes with the next data-point in the trajectory and then predicting the next note with the model.

4.4. Gaussian Mixture Model

We would like to compare music generated with the above predictive models to something more straightforward and naive. We would also like to characterize motifs within the trajectories. For this we use a Gaussian Mixture Model, and cluster snippets from all trajectories using the Expectation-Maximization (E-M) algorithm. This model involves maximizing the log-likelihood of the data by alternately updating $Q(z) = p(z|x;\theta)$ and θ. This has the effect of updating and then maximizing the lower bound for the log-likelihood given by:

$$ELBO(Q,\theta) = \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

The entire trajectory dataset, normalized into the range of pitch in music, was cut into snippets of length 6 data points. The features used for clustering of these snippets were the pitch and the gradient of the trajectory data points. The classes were associated with a snippet of music from a Clementi piece that minimized the L2 norm between the pitch and gradient features of the center of the clustering class and a music snippet. A trajectory was then transformed into music by substituting the trajectory snippets with the corresponding music snippet of its class.

5. Results and Discussion

5.1. Music Generation

5.1.1. Softmax Regression

Our baseline approach was to model the note sequences as 50-dimensional vector inputs and train a softmax regression to predict the next note given a vector of the 50 previous notes. For the softmax model, we used the Adam optimizer with a learning rate of 0.001, chosen because other learning rates did not seem to appreciably change the final outcome of the regression. The evaluation metric used was accuracy (the proportion of predicted next notes that matched the actual next note), and the loss function was categorical

cross-entropy. The results are shown in Figure 7.

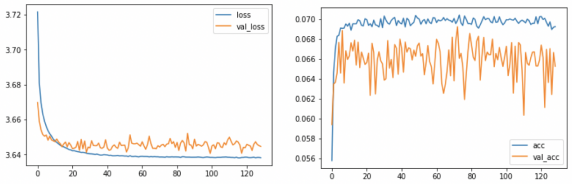

Figure 7. Loss (left) and accuracy (right) progression against training epochs for softmax regression. Training values are shown in blue and validation values are shown in orange.

The training accuracy for this model peaks around 0.070, while the validation accuracy averages around 0.068. While the model does not seem to be overfitting, the extemely low peak training accuracy confirms that this model is not suited for music generation. This is to be expected because modeling notes over time as points in a high-dimensional vector space is unlikely to aid our understanding of note progression. When softmax is trained on a smaller set of similar musical pieces, the confusion matrix shown in figure 8 is obtained. From this, we can observe that the model tends to predict only a handful of frequently occurring notes.

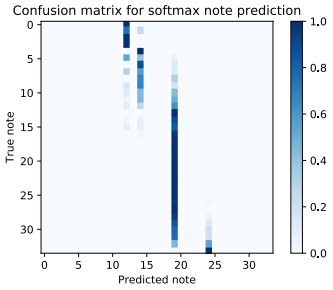

Figure 8. Normalized confusion matrix for the softmax model

5.1.2. LSTM

With inspiration from work by Skuli,[12] we created a sequential model with two LSTM layers and a final fully connected output layer with a softmax activation. The first LSTM layer had 256 hidden units, the second LSTM layer had 38 (the number of different notes in the training set) hidden units and the output layer also had 38 units. All learning parameters were identical to the softmax regression.

The divergence of the training loss and validation loss (Figure 9) is a clear sign of overfitting. Overfitting was significant with all tested variations of this model including variations with dropout layers and L2 regularization. We settled on this specific model as it was the simplest model that could achieve high training accuracy (Figure 9). It was observed that a high training accuracy was needed before the model was capable of generating interesting and varied music.

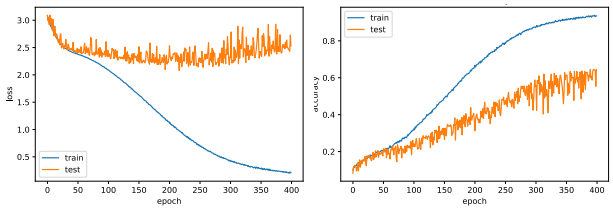

Figure 9. Loss (left) and accuracy (right) progression against training epochs for LSTM. Training values are shown in blue and validation values are shown in orange.

Training on a smaller set of music pieces similar in composition (the Clementi dataset) resulted in better validation accuracy than training on a large data set with more varying music pieces. For that reason, all subsequent evaluation and music generation was done using the Clementi dataset.

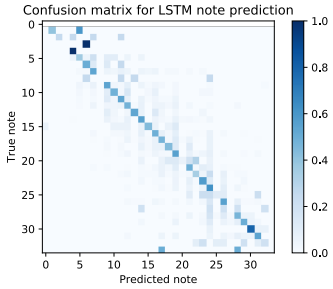

Figure 10. Normalized confusion matrix for the LSTM

To evaluate the behaviour of our model, we created a normalized confusion matrix (Figure 10) on the validation set. Here, we observe that the model performs better than suggested by the validation accuracy metric. Although we only achieve a validation accuracy of slightly more than 60%, the model generally predicts notes that are very close to the true note in pitch space.

5.2. GMM

Figure 11 shows the resulting populations of trajectory snippets of length 6 for each of the 16 classes used with the EM algorithm. We explored using different numbers of classes and snippet lengths, but this combination resulted in a relatively even distribution, which is desirable when matching to snippets in a composed piece so as to avoid repetitive motifs. Results of the samples generated with this method are discussed below.

5.3. Survey

Our goals in this project were to 1) generate music, and 2) have that music reflect a given trajectory. To analyze the success of our models in these two goals, we created a survey asking participants first to decide whether a given

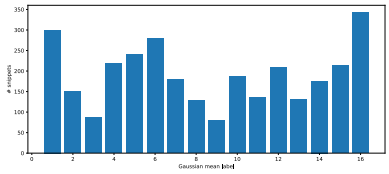

Figure 11. Population of trajectory snippets for EM with 16 GMM means.

audio clip was composed by a human or generated by a computer. Results from the Turing test portion of the survey can be seen in Table 1.

GENERATED	
LSTM FULL	56%
LSTM SUBSET	64%
GENERATED BASED ON TRAJECTORY	
SOFTMAX REGRESSION	13%
GMM	45%
LSTM SUBSET	42%
REAL MUSIC (CONTROL)	57%

Table 1. Turing test for generated music: results of a survey with 40 participants showing % responding the sample was composed by a human.

First, we generated music directly from the LSTM without trajectory information to see if we could simply generate realistic music (first two rows of Table 1). Both LSTMs are the same architecture, but one was trained on random samples from the full musical data set, while the other was trained on a subset: all pieces by the composer Clementi, whose melodies are regular and all occur in the right hand of the piano (i.e. they are conducive to the format of our simplified MIDI data). We see the samples generated from the subset LSTM pass the Turing test more successfully than the LSTM trained on the full data set. This could be an issue of not having a model complex enough to handle the variability in melodies, but also could be a by-product of our automated music simplification process, which does not perfectly select for a melody out of a musical piece. Because our goal is to generate believable music, we used the subset LSTM to generate music based on trajectories.

Comparing the audio clips generated based on the a trajectory, the softmax regression model produced samples that were least believable as human-composed, followed by the LSTM, then the GMM. The GMM is the highest likely because the snippets from which it was generated were part of a human-composed piece (a Clementi melody).

The LSTM-generated samples based on the trajectory are less believable as human composed than the samples not based on the trajectory. However, they still pass the Turing test 42% of the time, a moderate success especially considering that real music only passes 57% of the time. This brings up an issue for future consideration: rhythm, tempo, and chords must be very important to the musicality of a sample, as when we remove them in the controls, the samples do not safely pass the Turing test.

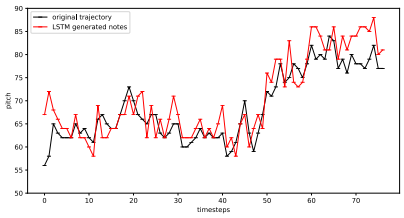

Figure 12. Trajectory and resulting LSTM-generated sample.

To assess the second goal, we asked participants to listen to a given audio clip and attempt to match it to the trajectory from which it was generated. Similar questions were used by Middleton to analyze the sonification of protein folding data.[1] Participants would see a plot like the black line in Figure 12, and hear an audio clip corresponding to the red line. Figure 12 shows a generated sample from the LSTM. In the survey, 88% of participants matched the LSTM sample to the correct trajectory, 53% for the GMM, and only 25% for the softmax regression. According to this survey, our LSTM model was the most successful in achieving goal 2.

6. Conclusions and Future Work

Of the models tested, the LSTM RNN was most successful at generating music that reflected trends in a given dynamics trajectory. Softmax regression produced samples with the same note repeated, which were neither musical nor reflective of trajectory data. The GMM approach had roughly the same success as the LSTM, but cannot truly be considered music generation, as it sampled snippets from composed pieces. It should be noted that to more fully analyze the success of the models in achieving both goals outlined, we would need a survey with a much larger sample size both in number participants and number of audio clips. Future efforts on this project would include curating a larger dataset with distinctive melodies, and exploring other generative models such as GANs or VAEs. A preliminary application of GRUs to this problem yielded promising results which could be improved by experimenting with different hyperparameters in order to optimize the model. Furthermore, as we saw with the control on our Turing test, reducing a piece to simply pitch and time removes much of the musicality. We would also want to extend the model to train not just on pitch, but also on rhythm, chords, and other expressive information, then explore methods of interpreting the trajectory data with these additional features.

7. Contributions

- Austin: model exploration for the problem, including softmax, LSTMs, and other RNNs
- Soren: MIDI data collection, LSTM construction and analysis
- Grace: dynamics data collection, GMM, survey construction and results, poster lead

8. Code and Music

Code used for this project can be found here: https://github.com/GraceJohnson/cs229-final-project

The survey is here: https://forms.gle/bcyshHBYC3nv4naTA, and you can listen to the composed and generated music used in the survey here: https://soundcloud.com/user-278795430/sets/listen-to-your-data

References

[1] Bywater, Robert P., and Jonathan N. Middleton. "Melody discrimination and protein fold classification." Heliyon 2.10 (2016): e00175.

[2] Weir, H., Williams, M., Parrish, R., and Martinez, T.J. "Nonadiabatic dynamics of photoexcited cis-Stilbene using ab initio multiple spawning." *In prep.* (2019).

[3] Hermann, T., and Helge R.. "Listen to your data: Model-based sonification for data analysis." Advances in intelligent computing and multimedia systems (1999).

[4] Hermann, T.. Sonification for exploratory data analysis. Diss. 2002.

[5] TwoTone. TwoTone, twotone.io/.

[6] Johnston, D. (2015, Aug). Composing music with recurrent neural networks. Retrieved from http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/

[7] AIVA. Retrieved from https://www.aiva.ai/

[8] *Magenta*.(n.d.). https://magenta.tensorflow.org/. Retrieved from https://magenta.tensorflow.org/

[9] Mogren, O. (2016). C-RNN-GAN: Continuous recurrent neural networks with adversarial training. arXiv preprint arXiv:1611.09904.

[10] Malik, I., and Carl H.. "Neural translation of musical style." arXiv preprint arXiv:1708.03535 (2017).

[11] Chung, J., Gulcehre, C., Cho, K., Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

[12] Skuli, S. How to Generate Music Using a LSTM Neural Network in Keras. Towards Data Science. (2017). Retrieved from shorturl.at/hiGK6.

[13] Cuthbert, M. S., & Ariza, C. (n.d.). music21: A toolkit for computer-aided musicology and symbolic music data.

[14] Holzner, *A. LSTM Cells in Pytorch*. Retrieved from https://medium.com/@andre.holzner/lstm-cells-in-pytorch-fab924a78b1c

[15] Fabian P., et al., Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830 (2011)

[16] Bjorndalen O., Mido, https://mido.readthedocs.io/en/latest/index.html

[17] Chollet, F. et al (2015), Keras, https://keras.io