Jazz Solo Generation using Long Short-Term Memory Recurrent Neural Networks

Grant Simmons
Department of Electrical & Computer Engineering
Stevens Institute of Technology
Hoboken, NJ
gsimmons@stevens.edu

Ilana Zane
Department of Electrical Engineering
Stevens Institute of Technology
Hoboken, NJ
izane@stevens.edu

Justin Ho
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ
jho1@stevens.edu

***Abstract*—Music and musicality are uniquely human experiences, with much in the way of allowing people to express and impart thoughts and ideas through the arrangement of sounds. We attempt to encode music—specifically jazz solos—in such a way to facilitate deep learning through a Long Short-Term Memory recurrent neural network model to observe if we can impart artificial musicality in machines in generating novel artificial jazz solos.**

I. Introduction

Replicating human behaviors using computers has remained an elusive goal for engineers. With the advent of deep learning techniques, computers become more capable of generating seemingly human ideas by the day. Jazz music is an art form fueled by real-time improvisation. The greats of Jazz music spend their lives mastering the intricacies of instruments, music theory, and quick thinking to put forward some of the most impressive feats in music. Decoding the structure and patterns behind their solos has perplexed researchers for decades, and jazz solos continue to be best formulated by the humans who dedicate their life to the craft. We venture to create jazz solos using modern machine learning techniques and evaluate their statistical and audible similarities to some of the best solos of all time. By evaluating the successes and failures of such an effort, we intend reflect on the aspects of great jazz solos that make them so notoriously difficult to rival and formulate suggestions for future ventures in music generation.

We examine the solos of various jazz musicians by utilizing the Weimar Jazz Database [1], a project developed by the Jazzomat Research Project, which offers a free SQLITE3 database of famous jazz solo transcriptions specifically formatted for convenient statistical analysis. This database contains MIDI representations of over 450 jazz solos and contains additional contextual information crucial for comprehending each solo's structure. The MIDI format allows for the convenient extraction and analysis of the melody by describing the properties of each note in sequence. This format facilitates the solo's convenient quantitative and statistical analysis by providing discrete, quantized values describing the properties of each note.

At a fundamental level, the algorithm we implement attempts to predict each successive note in a solo by analyzing the qualities and context of the current note. After generating a prediction, the model is trained based on its ability to correctly generate the immediately succeeding note in the solo. This form of learning is conducive to the Recurrent Neural Network (RNN) model, which consumes as input an analysis of the model's previous state when predicting new values. A common RNN for music generation is the Long Short-Term Memory (LSTM) model, which can learn to recognize an important input, store it in the long-term state, preserve it for as long as it is needed, and extract it whenever needed. Because of this, the LSTM layers are successful at capturing long-term patterns in audio recordings and more [2] We compare the results of the LSTM model against the already existing song samples to evaluate the performance of the model. Following solo generation, the melody is recombined with a backing track that will be generated with an outside source to assist in providing context to the melody during evaluation.

The first main algorithm we implemented was an ARIMA model that analyzed a jazz solo as a time series and made predictions for future notes. The results show that the current model is underfitted, but provided information about the stationary properties and trends of the data. The second algorithm we implemented is the RNN in order to make stronger and more realistic predictions for future notes. The third algorithm we implemented is a refinement of our second algorithm—we use a deeper RNN model using techniques from existing research to better generalize our model to the dataset, which reduced overfitting and created an overall more accurate and musical model over the simpler RNN.

II. Related Work

A number of approaches to jazz solo generation have been proposed and experimented in an attempt to encode musicality in machine learning models. Great efforts have been made into finding the best fit approach across a great swath of areas in machine learning, from neural network approaches to evolutionary approaches to probabilistic approaches.

Deep Learning. Deep learning is undoubtedly an extremely strong tool in identifying and replicating the underlying patterns present in datasets. Despite the inherent opacity problem of deep learning models, the power of these models are a perfect fit for problems that cannot be properly understood by humans, which would ideally fit in encoding musicality and be applicable to our goal in jazz solo generation—however, despite the power of deep learning, these models often need a very large dataset to work with to properly learn the underlying

patterns, which does not exist for jazz solos. An approach to address this problem to this problem is through the use of "transfer learning" [3], where a model takes knowledge learned from one task and applies said knowledge to the goal task. However, Hung et al. finds that this approach has marginal returns [3].

Genetic Algorithms. While in a different sense than deep learning models, genetic algorithm models can also be powerful tools in processing and searching for solutions in datasets heuristically. However, the use of heuristics is where the problem arises in genetic models: since genetic algorithms use the concept of "survival of the fittest" to decide which candidates "survive," the fitness heuristic must be defined, which is can be largely subjective in the realm of music, as Biles recognizes in his own implementation, where he himself dictates fitness of the generated solos [4]. Some have attempted to abstract fitness heuristics, but without some specialized genetic operators, these abstracted fitness heuristics can be unwieldy and produce uninteresting results [5].

Markov Chains. Markov chain approaches have been extensively tested in the past in the realm of jazz generation, owing to it being computationally cheap to run and its ease of implementation. Many of these experiments have found that the Markovian approach to music generation produces decent to good outputs, even going as far to state that "there is indeed a strong Markovian dimension in musical surface in most genres of tonal music, including jazz" [6]. These models are defined by their "random walk" method, generating the next step in an output probabilistically, which fits with the improvisational nature of jazz. However, the randomness in the random walk method is also exactly where the weaknesses lie in Markov chain models for music generation, as these models will not always tend towards the most probable solutions over time, which will effectively require the deployment of deeper, finer controls over the generation process [6].

III. Our Solution

A. Description of Dataset

Datasets of transcribed jazz solos, especially free and open source datasets, are particularly scarce due to the scale of the detail required to create an accurate transcription. One promising dataset was painstakingly compiled by The Jazzomat Research Project, an organization dedicated to jazz analysis, who painstakingly transcribed 456 solos of various artists and styles. In their dataset, solos were transcribed, converted into a MIDI representation, and further organized into a SQLITE3 database that contains descriptive metadata of each piece, including the form, genre, composer, solo instrument, and a complete MIDI representation of the solo. In the MIDI format, each note played by each instrument is represented by a table of values indicating various qualities of that note, such as the volume, pitch, onset, duration, attack, and so on. By querying the database to extract a tune and its relevant data, the tune can then be individually processed to improve a model's predictive and generative capabilities.

The SQLITE3 database contains multiple tables, including the `melody` and `section` tables. The melody table, for example, contains the notes of every transcribed solo, sorted by melody ID and event number (i.e. sequential note). The section table contains abstracted information about the solo, such as the underlying chords, the chorus number, and the measure number, sorted by melody ID and a range of corresponding melody events. In order to combine these database tables into a single object appropriate for our model, the dataset is preprocessed by querying all section data from the database and annotating the solo's notes with the corresponding section information. This provides a note level granularity with contextualized annotation to support the training algorithm described in the following section.

While the melody transcriptions are inherently audible information, the MIDI format allows for convenient visual analysis as a result of its discrete pitch representation. Statistical information, such as the pitch and duration distributions can be visualized by analyzing each note in the melody. Furthermore, by visualizing the pitch value over the duration of the solo, the contour of the solo can be arranged into a Gantt chart format, which can be visually examined to evaluate the fidelity and complexity of the solo. To observe the context surrounding the solo, the section information can be overlaid on the solo's contour graph to visually correlate the melody with the tune's measures, sections, and choruses. This analysis is helpful in ensuring the fidelity and correctness of the data entering the model.

B. ARIMA Model

For our project we are treating the jazz solos as a time series—we are analyzing the pitch of individual notes over a certain length of time. Our ultimate goal is to create an RNN to generate jazz solos, but we first created an ARIMA (autoregressive integrated moving average) model to gain more insight into our time-series and see if we can generate any solos. In Figure 1 we display the training set in orange and the testing set in green for one song from the dataset.

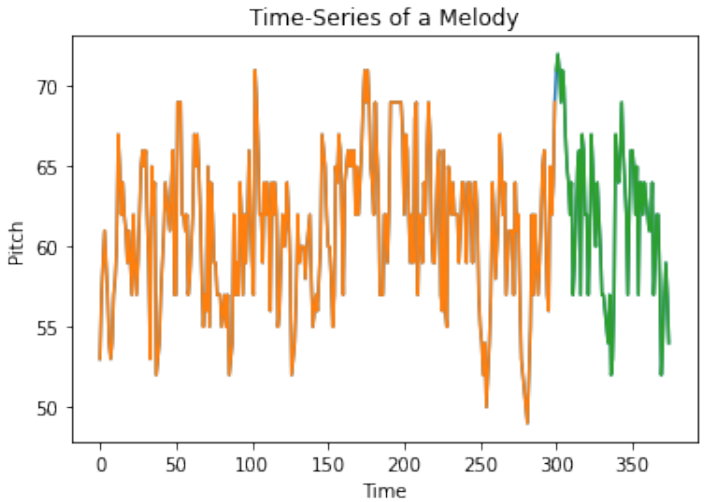

Figure 1: Singular Melody as Time Series

To create the ARIMA model, we choose the parameters p, q, and d, where p is the number of time lags of the AR

(auto-regressive) model, q is the order of the MA (moving-average) model, and d is the maximum number of non-seasonal differences. We conducted an ADF (augmented Dickey–Fuller test) which tests the stationary of a time series and further determines if we need to difference it. Our test determine that the series is not stationary and we can therefore set $d = \text{None}$. We chose p and q based off of the following partial-correlation and auto-correlation plots.

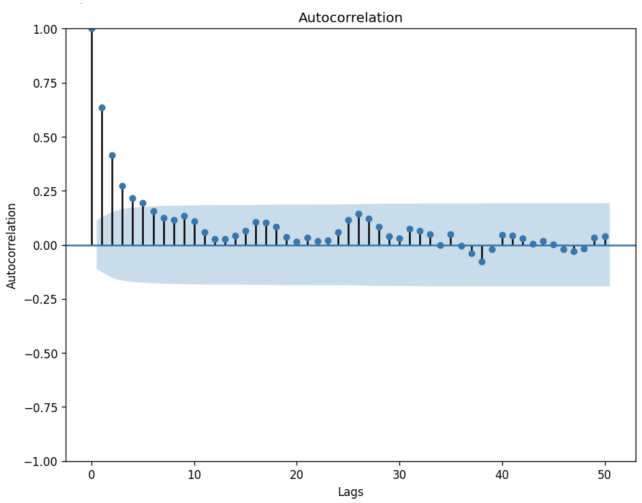

Figure 2: Autocorrelation Plot. Choose $p = 4$

Based on Figure 2 we can see that there are four significant lags so we choose the AR term to be 4. Next we created a partial autocorrelation plot to describe the MA.

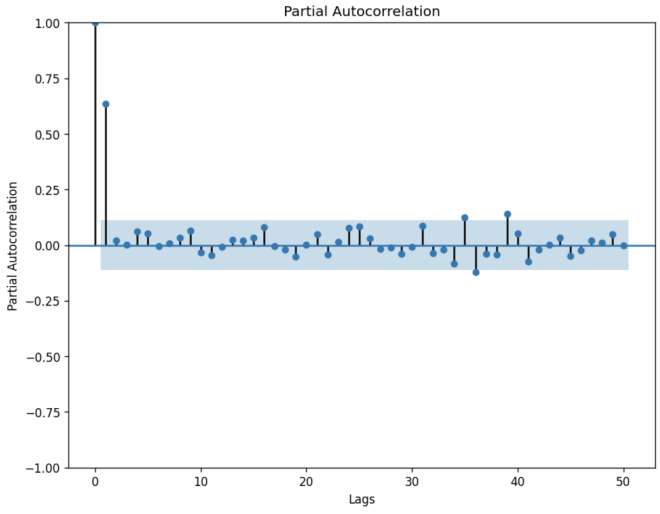

Figure 3: Partial Autocorrelation Plot. Choose $q = 2$

From Figure 3 there are two significant flags so we can say that the MA term is equal to 2. We create our model based off of these parameters and train it on 80% of the data and use the remaining 20% for testing.

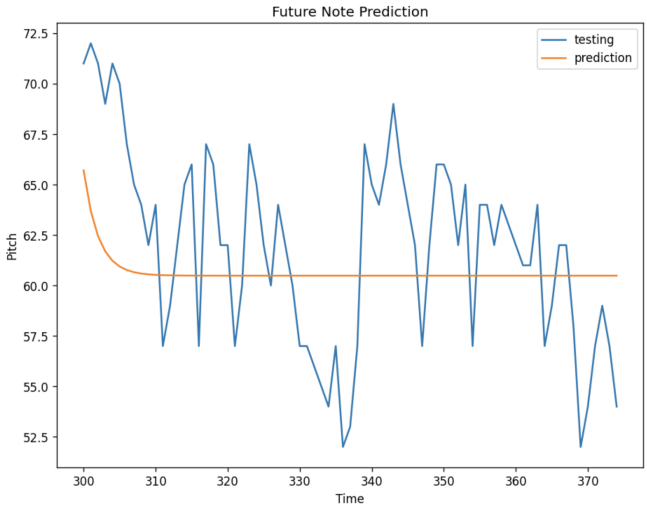

Figure 4: Prediction Results

Our ARIMA model results in poor predictions, probably because this is not the right model to make jazz solo predictions and the model is underfitting the data. Out of the 456 song samples, our model only used 91 songs which may not have been enough information for the model to generate its own unique predictions. However, we have uncovered some useful information related to the trends and stationary properties of our dataset. This will help us make a more accurate RNN model for the next segment of our project.

C. RNN Model

1) Training: In order to create the training set for the RNN, we had to collect the right attributes from our dataset. We decided to use the pitch, step and duration values. The pitch and duration values were provided through the database. The step value was calculated based off the difference between. We used this as it made more sense to represent the time difference between each note with respect to each other instead of with respect for the beginning of the song.

	pitch	step	duration
0	60.0	0.046939	0.036735
1	61.0	0.138776	0.122449
2	62.0	0.134785	0.118367
3	65.0	0.143673	0.117551
4	67.0	0.163991	0.150930
...	...	...	...
563	57.0	0.087755	0.067347
564	56.0	0.157143	0.065306
565	57.0	0.114286	0.085714
566	60.0	0.108163	0.081633
567	57.0	0.000000	0.093878

568 rows × 3 columns

Figure 5: Training Dataframe

The data frame was then turned into a tensor that would be used for the training set. The training set consists of

the notes from one song, which ranges between 300 to 600 notes. In order to expand the training set, we can use more than one song. However, due to time constraints we wanted to keep training as short as possible and therefore used one song. To train the model, we created a function called `create_sequences()`, that took in our dataset, sequence length, and a normalization value. Within this function, we get a subset of the training data (containing pitch, duration, and step) that is of a variable `sequence_length` (`data[:sequence_length]`), then we obtain the same information from `data[sequence_length + 1]`, which will be used as our label during training.

Our RNN will take this sequence of data, analyze the pitch, step and duration and then make predictions for the pitch, step and duration for the next note and compare it against the label. The RNN continuously takes sequences of data until it has reached the end of the training dataset. The library that we use to turn our generated notes into midi files have pitch numbers that range from 0 to 127, which is why when we create the sequences we set the normalization value to 128 and normalize our pitch values to be within this range.

2) Implementation: In order to calculate loss for the step and duration we use MSE and encourage the step and duration values to remain positive. For the pitch we use the SparseCategoricalCrossEntropy function from TensorFlow[7] to compute the cross entropy between our prediction and the label.

Our model consists of 5 layers: the input layer, one LSTM layer, and three dense layers for the duration, pitch and step [7]. We used the dense layers as hidden layers as we wanted to find any connections or patterns between our three input parameters. If we were using shorter sequences, just the three dense layers might have been fine, but we found that using the LSTM layer provided better performance as it can detect long-term dependencies in the data. The model's summary shows all of the model's layers including the names, output shapes, and its number of parameters.

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 25, 3)]           0         []

 lstm (LSTM)                 (None, 128)               67584     ['input_1[0][0]']

 duration (Dense)            (None, 1)                 129       ['lstm[0][0]']

 pitch (Dense)               (None, 128)               16512     ['lstm[0][0]']

 step (Dense)                (None, 1)                 129       ['lstm[0][0]']

==================================================================================================
Total params: 84,354
Trainable params: 84,354
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 6: Model Summary

When we compute the loss through MSE and cross-entropy, we find the individual loss values for the pitch, duration and step. Then, we sum them together to find the overall loss. We found that after training the overall loss was dominated by the pitch loss as pitch loss was above 0. In order to correct this we added weights of 0.05, 1.0 and 1.0 to the pitch, step and duration, respectively[7]. We also used the Adam optimizer. Below are the loss values before weights and optimization and then after.

```
3/3 [==============================] - 2s 80ms/step - loss: 5.8220 - duration_loss: 0.6876 - pi
{'duration_loss': 0.6876155734062195,
 'loss': 5.822010040283203,
 'pitch_loss': 4.835506916046143,
 'step_loss': 0.29888829588890076}
```

Figure 7: Before Weights and Optimization

```
3/3 [==============================] - 1s 28ms/step - loss: 0.2068 - duration_loss: 0.0234 - pitch_
{'duration_loss': 0.023447437211871147,
 'loss': 0.20676033198833466,
 'pitch_loss': 2.9596779346466064,
 'step_loss': 0.03532899543642998}
```

Figure 8: After Weights and Optimization

Figure 9 shows the resulting loss after training our model for 50 epochs. We implemented early stopping and found that MSE loss stopped changing significantly around 20 epochs, therefore we stopped training at that point in order to reduce overfitting.

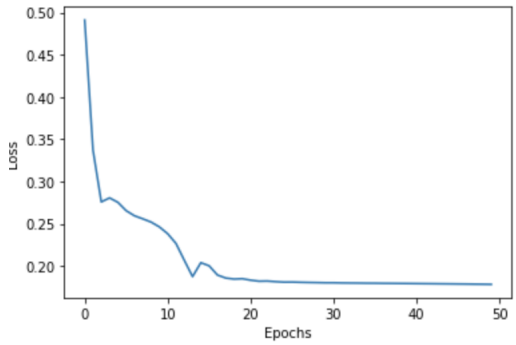

Figure 9: Epochs vs. Loss

Finally, in order to generate note predictions, we fed a sequence of data to our model and had it predict the next 120 notes. The number of notes generated can be changed. We found that 120 notes is about 30 seconds of audio and provided a good representation of how the RNN performed. When generating our note predictions we used a hyperparameter for the LSTM layer called temperature, which we set to 2.0. The temperature is used to control the randomness of predictions by scaling the logits before applying our softmax function. A higher temperature creates a more confident model, but also more conservative. A lower temperature leads to more diversity, but also more mistakes. Figure 10 shows the dataframe of generated notes that is later converted to a midi file that can be played back.

	pitch	step	duration	start	end
0	37	0.184326	0.127027	0.184326	0.311353
1	62	0.745468	0.425646	0.929795	1.355441
2	55	1.154603	0.664142	2.084398	2.748540
3	75	1.320891	0.789546	3.405289	4.194835
4	61	1.422397	0.799789	4.827686	5.627475
5	68	1.450695	0.847832	6.278381	7.126213
6	64	1.484224	0.851270	7.762605	8.613875
7	65	1.499976	0.870692	9.262581	10.133273
8	67	1.515806	0.879202	10.778387	11.657589
9	65	1.531487	0.882672	12.309874	13.192547

Figure 10: Dataframe of Generated Notes from RNN

D. Deeper Models

Our third model took inspiration from past researchers by implementing multiple layers of LSTM models and adding layers of dropout. By implementing a second LSTM layer and creating a layer of dropout between the layers, as well as between the last LSTM and the output layer, we can increase the model's ability to generalize to the solution space. This becomes evident during training, as the model's performance increases more gradually, presumably due to the random dropout and multilayer architecture. This trend is visualized in Figure 11 By increasing the number of LSTM and dropout layers, we can reduce model overfitting and develop a more applicable model.

By additionally modifying the training weights of the standard LSTM model, we also observed that the occurrence of outlying pitch values drastically decreased, creating more realistic solos, performed in a range much more similar to their respective instrument.

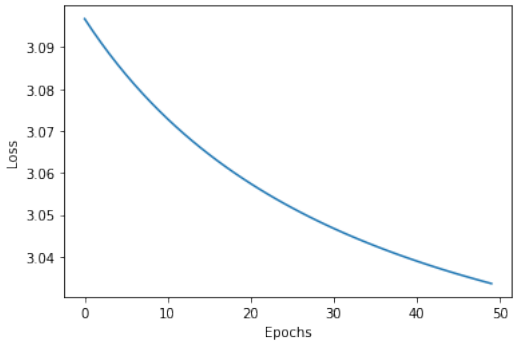

Figure 11: Epochs vs. Loss for a RNN with multiple LSTM layers and dropout

IV. Comparison

A key component of analyzing model performance is to compare the resulting solo distribution with the distribution of the training data. Such a correlation can demonstrate a similar use of pitches used to establish a melody. In our testing, we find that our generated solo contains a very similar pitch distribution to the training data, roughly centered around the same pitch. Whereas in the generated model the distribution of the pitch is more concentrated, the training dataset shows a more multi-modal distribution. This is in part due to the training dataset's consideration for chord changes in the underlying harmony, causing a periodic shift in the center pitch, or root note. Since our model does not consider the underlying chord, we believe that the model rather generalizes the distribution seen in the training data by averaging the numerical value of the pitch values on which it trains. This trend can be observed by comparing Figures 11 and 12.

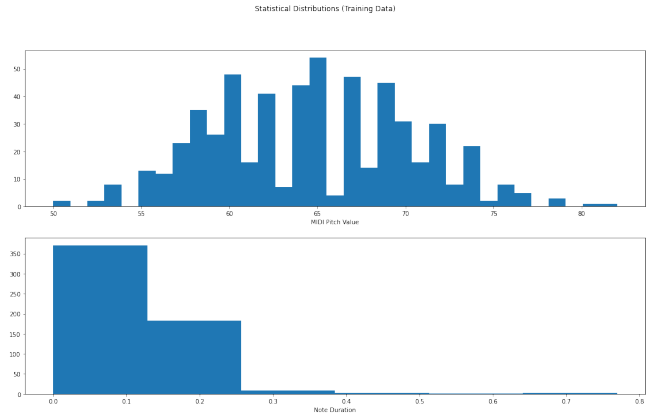

Figure 12: Training data note Pitch and Duration statistical distribution

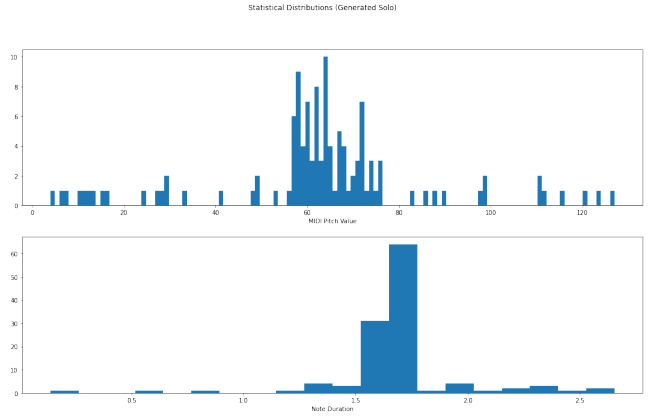

Figure 13: Generated solo note Pitch and Duration statistical distribution

This distribution is highly dependent on the temperature parameter, described above. Higher temperature values lead to a more varying solo result, in effect creating a wider pitch distribution. Figure 14 demonstrates distributions with temperature values of 0.1, 0.5, 3.0, and 10.0.

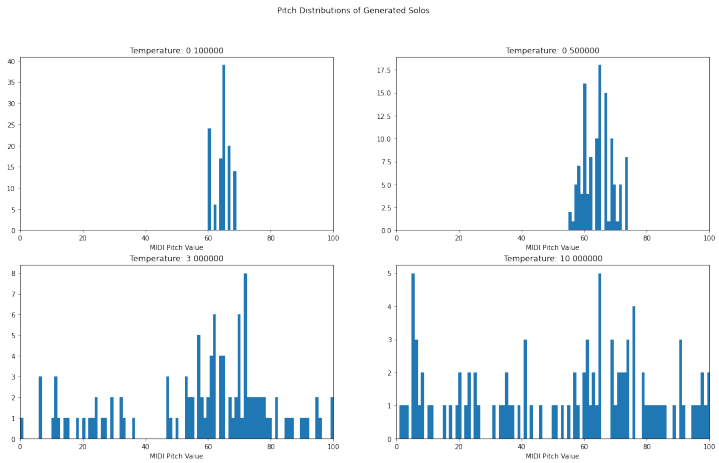

Figure 14: Pitch distributions for temperature values of 0.1, 0.5, 3.0, and 10.0

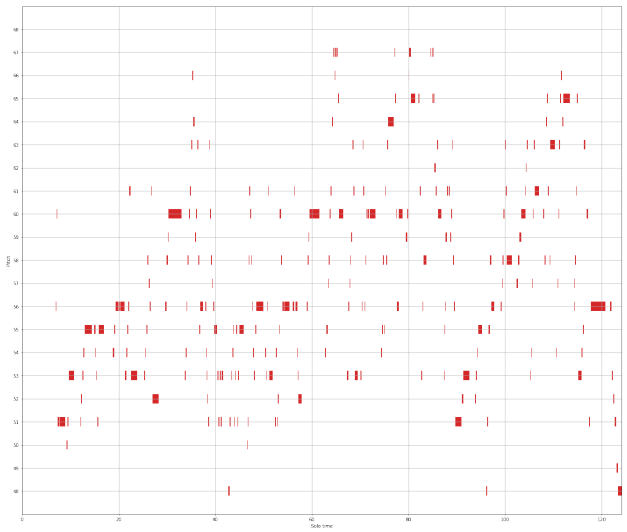

Figure 15: An example contour from the Weimar Jazz Database

With an immature model, we find that keeping smaller temperature values, generally below 0.7, results in a reasonable solo. Above this threshold, increasing variability is introduced in the solo, creating more sporadic distributions and more "avant-garde" solos.

Note duration distribution is also a key metric, as part of the solo's variability and surprise comes from varying note sounds. A curious trend we observed in our analysis uncovered a discrepancy between the training data and the generated solo. We noticed that in solo generation, the model preferred to generate notes with a duration value nearing zero. This could be a result of underfitting the model to the training dataset, but it may also be an oversight in how the data is preprocessed and presented to the model during training.

Aside from statistical measures, the "contour" of the solo, or how smoothly the pitch changes between notes, is also an important consideration, as it can provide an idea of whether the notes lie in an acceptable range for a given instrument or vary too wildly to create an appealing solo.

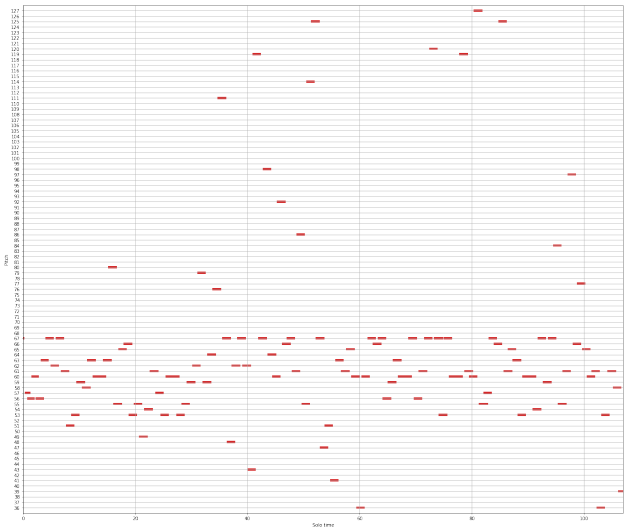

Figure 16: An example contour generated from the solo presented in Figure 13

These contours clearly demonstrate a range in which the model is more likely to generate a note, which generally lies within the range of notes presented during training. However, the trained model also generated notes outside this training range as a result of the inherent randomness of the model.

Arguably the most critical component for music generation is its musical comprehensibility. A solo may perfectly match the pitch and duration distributions of the training data, but if the solo does not make harmonic sense or sounds generally unpleasing, there may be room for improving the model.

We utilized the MIDITrail tool to visually and audibly analyze the generated solo. In general, we found that the solos, while maintaining relatively accurate note characteristic distributions compared to the training dataset, sounded a bit

random. While we can observe general contour trends, such ascending or descending phrases, that are generated with the help of the LSTM model, the pitches of these phrases do not seem to conform to a specific key or set of harmonically agreeable tones. This can be rectified by creating a lower and upper bound for the pitch range depending on the instrument, however that is outside the scope of this project.

Existing implementations have used LSTM models to generate jazz solos, such as the model developed by Shubham Gupta [8].We believe that the differences in our predictions are due to our model's lack of consideration for chord changes. By considering chord context along with the pitch, duration, and step values, their model considers additional context important for solo generation. Such a feature would be particularly helpful in generating musically sensible ascending and descending phrases.

Furthermore, Gupta's model results in a higher degree of note duration variation. We have not determined a root cause, but we found that notes generated by our model settle into a very regular and predictable cadence as the solo progresses—a trait typically undesirable for jazz music, but might be desirable for other genres of music such as classical music.

We also found that competing models will generally limit a solo's range to the range of the solo's instrument, as would be expected from a real instrumental performance. This is another shortcoming in our approach, as we do not limit or restructure the generation of solos to account for this constraint. We believe this may also play a role in generating a more audibly attractive solo.

V. Conclusion

Through our experimentation, we have demonstrated a methodology to train multiple machine learning models using improvised jazz solos represented in the MIDI format. Though previous attempts have been made to generate music with deep learning, we sought to train linear models and various neural networks to generate jazz solos based on the work of the Jazzomat Research Project [1], which hand-transcribed hundreds of classic jazz solos into a single database. Using these classic solos, we trained ARIMA models and LSTM-based neural networks to observe the patterns and themes of great jazz artists.

We quickly discovered that ARIMA models are not appropriate for solo generation, which is better suited for RNN-based models. While the LSTM model outperformed the ARIMA model by a significant margin, the results of our model fell well short of the musical greatness of Charlie Parker and Dizzie Gillespie, but it occasionally generates a phrase or idea that sounds like it may fit somewhere in their work. While we have not in this project demonstrated a complete solution to automate jazz improvisation, we present a framework that makes it possible with more appropriate algorithms and neural network architectures.

By conducting a statistical analysis of the generated solos, we found that the pitch and duration distributions were appropriate for the given training data, demonstrating that we had appropriately provided musical data to the network. What our solution lacks, however, is a sense of musicality which is a subjective quality that can be further studied. While the pitch, step, and duration distributions may fit what is expected from the training dataset, we found that many of the generated solos were different from what we expected. This helps illustrate the fact that simply because a network implementation arrives at a statistically appropriate conclusion, the algorithm may not be sufficient for the intended problem space. This is amplified by approaching a problem like jazz, where the content is generally free-form, improvised, and the success of the result is highly subjective.

VI. Future Directions

For researchers interested in pursuing this space, we provide suggestions to build on successes and failures seen by our model.

To improve upon the LSTM model demonstrated above, we first suggest experimenting with different note representation formats such as a normalized representation, in which all note pitch values are represented as an integer displacement from the chord's base note. For example, if the chord is A-7, and the soloist is playing a G, the integer displacement would be 10, as the G is 10 chromatic steps away from the base note A. Such an approach will allow solo phrases to better generalize to all chords of a song. With un-normalized note representations, a neural network may make false associations between solo phrases and chords when pitches are not normalized to the base note of a chord. Such a change will increase the amount of pre-processing required on the data but has the potential to improve the quality of the generated solos. It will also allow the model to generalize to multiple solos regardless of the training solo's key.

Furthermore, future models should train on a richer subset of the database data. Currently, the LSTM model attempts to generate new notes based solely on the pitches of a sequence of previous notes within the solo. By adding form, chorus, beat, and chord data to the training dataset, more appropriate pitch and duration values can be predicted, and therefore generated, by the model.

For result analysis, especially when solos are generated over chord progressions, we suggest also generating a backing track and merging the result to evaluate the solo's performance in context of the chord progression.

Current research in this area also focuses on the decomposition of jazz solos into ideas and phrases, developing a form of jazz "grammar" [9]. By conducting such a hierarchical analysis, solos can be generated in a form more akin to a musician's train of thought during improvisation, where they imagine harmonic and melodic ideas instead of individual notes. To facilitate this pattern of hierarchical analysis, we believe that employing a convolutional neural network (CNN) may be appropriate for extracting such features.

References

[1] M. Pfleiderer, K. Frieler, J. Abeßer, W.-G. Zaddach, and B. Burkhart, Eds., *Inside the Jazzomat - New Perspectives for Jazz Research*. Schott Campus, 2017.

[2] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras TensorFlow*. O'Reilly Media, 2019, ISBN: 9781492032649.

[3] H.-T. Hung, C.-Y. Wang, Y.-H. Yang, and H.-M. Wang, "Improving automatic jazz melody generation by transfer learning techniques," in *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2019, pp. 339–346. DOI: 10.1109/APSIPAASC47483.2019.9023224.

[4] J. Biles, "Genjam: A genetic algorithm for generating jazz solos," Jul. 1994.

[5] G. Papadopoulos and G. Wiggins, "A genetic algorithm for the generation of jazz melodies," 1998.

[6] F. Pachet, "Musical virtuosity and creativity," in *Computers and Creativity*, 2012, pp. 115–146. DOI: 10.1007/978-3-642-31727-9_5.

[7] "Generate music with an rnn:tensorflow core." (), [Online]. Available: https://www.tensorflow.org/tutorials/audio/music_generation#extract_notes (visited on 05/05/2022).

[8] S. Gupta, *Composing jazz music with deep learning*, May 2018. [Online]. Available: https://www.hackerearth.com/blog/developers/jazz-music-using-deep-learning/.

[9] J. Gillick, K. Tang, and R. M. Keller, "Machine learning of jazz grammars," *Computer Music Journal*, vol. 34, no. 3, pp. 56–66, 2010. DOI: 10.1162/comj_a_00006.