Fusion of Music Styles Using LSTM Recurrent Neural Networks

Jacob Sundstrom
UCSD(Music)
PID: A09060631
Email: jlsundst@ucsd.edu

Harsh Lal
UCSD(CSE)
PID: A53244610
Email: hlal@ucsd.edu

Nakul Tiruviluamala
UCSD(Music)
PID: A5321500
Email: ntiruvil@ucsd.edu

David Defilippo
UCSD(Music)
PID: A53200095
Email: ddefilip@ucsd.edu

Abstract—**Appeal of a musical composition is almost exclusively subjective in that it is a combination of the tastes, preferences, and history of an individual's experiences. That is, it is perceived and judged qualitatively in a different way by different individuals. In this project we propose to build a deep learning system which could take n different samples of a jazz soloist - especially a variety of samples of specific 'styles' - and generate sound using current input as well as feedback and memory from the past samples. This generation can then be judged by a 'human' agent and the parameters of the neural network could be adjusted accordingly to generate a fusion music that is more closer and appealing to agent's expectations. Recurrent neural networks with Long Short Term Memory (LSTMs) in particular have shown promise as a module that can learn long songs sequences, and generate new compositions based on the song's harmonic structure and the feedback inherent in the network. [5] We plan to explore the same through our experiments.**

I. Introduction

Music is an extraordinarily appealing auditory sensation and the classification of musical style can be virtually infinite. Since the dawn of music there has been a desire to fuse musical styles in different ways, taking either essential or superficial musical aspects of two or more compositions or performers. Usually highly skilled musicians mix different musical styles and compositions intuitively, creating a fusion of style that they themselves are satisfied with. In this work, we we try to combine different musical styles using deep learning techniques. All of the current work is performed using MIDI files but can be easily extended to incorporate new formats. It should be noted, moreover, that musical style is used in a broad sense in that individual players within a single genre can have differing "styles".

The most straightforward way to generate music with with a recurrent neural network (RNN) is to use the network as a single step predictor. The network can be trained to predict pitches at time $t + 1$ conditioned on pitches until time t as inputs. After the training is completed, the network can be seeded with random initial inputs (mostly from training samples) to generate novel musical compositions using subsequent output generated as input for next generation. This note-by note approach was first examined by Bharucha & Todd.[6], [7]

A generic feed-forward neural network would not be a good fit at composing music using the technique described above. The main reason behind this is the inability to memorize any information about the past, thereby not being able to keep track of variations & periodicity of a song's feature across time. In principle RNN does not suffer this limitation owing to its recurrent connections and hidden activation layers that may act memory elements, thereby exhibiting temporal behaviour. However they do not perform well at the task of modelling long duration temporal dynamics as studied by Mozer.[8], with most likely cause being the the problem of vanishing gradients associated with RNNs.[9]

So we explore the viability of using LSTM recurrent neural networks for the same and experiment with fusion of different musical styles. For this we extract features such as pitch, rhythm and amplitude from midi files as described in II-B. We then pre-process it and pass it to a neural network. We train neural networks separately for each of the features. We then generate the chosen features from target musical style and then combine them to generate the final output result as described in III-C. We then experiment with varying styles and describe out results in latter sections.

II. Dataset

The most important considerations in choosing a dataset were both musical and practical. Musically, the dataset had to be interesting enough to captivate a listener. Practically, the dataset had to be focused enough in order to train the network in a timely fashion.

The choice to use MIDI files as our dataset file was one of practical considerations. Sound files, such as MP3 and WAV, capture the waveform of the sound and reproduce the sound as it was recorded or synthesized. The waveform itself contains no musically meaningful material without extensive processing, beat tracking, onset detection, frequency analysis, etc. Moreover, the analysis and extraction methods used on signals is often imprecise and typically of an entire song with all instruments playing simultaneously. Extracting a single instrument from the texture and then accurately representing it quantitatively is extraordinarily difficult and creates additional difficulties of representing that information in a musically meaningful fashion.

The decision to utilize MIDI led us to the Weimar Jazzomat Database. It is one of the largest jazz midi datasets with a corpus of 456 transcribed solos. Jazz as a genre offered several advantages: specific soloists often have repeated rhythmic and harmonic ideas that carry through different songs, jazz standards offered a predictable harmonic palette with which to use, and transcribed jazz solos are often available for free. Additionally, the solos that were chosen were those of monophonic instruments, like that of the saxophone or trumpet and unlike that of the piano or guitar, since those instruments play more than one note at a time. This made the extraction of rhythmic data immensely easier and less complicated.

Given the sheer number of soloists and the time constraints of computing, we chose four players to analyze: Miles Davis, John Coltrane, Ornette Coleman, and Charlie Parker. These four giants of jazz were chosen for much the same reason, the first being that they each had a large number of transcibed solos in the database which allowed us to acquire the largest datasets possible. Moreover, and perhaps more importantly, they each have distinct and different styles and approaches to playing. As opposed to two players who, while individual, sound similar from a musical point of view, these players' approach is clear in any of the features we chose to extract.

In order to facilitate the largest datasets and given the stylistic choices specific to certain soloists, the files were grouped by soloist. This allowed us to easily maintain a repository for the harmonic, rhythmic, and dynamic information of a given soloist. There are, of course, certain caveats. Jazz standards, as a general principle, change key throughout the course of the tune. Therefore, our recurrent neural networks produce music that is never reminiscent of a particular song but instead evocative of a musical artist in general. The resultant music might be called more free as a result. Additionally, the MIDI file format makes parsing information on a temporal basis extremely difficult. It was therefore not including in the the final form of the pre-processed data.

A. Preprocessing

The biggest challenge of pre-processing was to somehow maintain the musical integrity of the data we extracted. This particular challenge proved to be too difficult to grapple with in the time allotted and the choice was made to simple normalize the data to render them song agnostic. That is, the representation of the data was not dependent on either the key, tempo, or other musical feature of the specific song. Within the MIDI file itself, 'note' events, such as note-on and note-off govern the resulting sound and it is from these note events that features were extracted.

B. Feature Engineering

Musical features are numerous and, depending on what one is seeking, can take many forms. In our case, we sought the most simple and basic traditional musical features: pitch, rhythm, and dynamic (or amplitude). The structure of MIDI files is such that these three features are prominently and clearly encoded. In a sound file that represents a waveform, extracting pitch information takes for form of FFT analysis which is then translated to musical pitch. This processes is difficult to manage and can introduce rounding errors, not to mention the difficulties with extracting a single instrument from a song texture. Moreover, re synthesizing such analysis extracted features into a sound file is riddled with additional challenges. Although one might be able to extract a pitch of an instrument, this says nothing about envelope characteristics, instrumental timbre, or accurate rhythmic representation. Thus, analyzing a sound file did not meet the ease-of-use criteria of our data.

MIDI files, on the other hand, contain in an easy to read format all the salient features we wanted access to in addition to being simple to parse. MIDI files are organized into 'tracks' which contain a single instrument, not necessarily monophonic. Additionally, a 'note' in MIDI-space is defined by two events: a 'note-on' and a 'note-off' that tell the MIDI engine when to start a note and when to terminate it. Each event contains three pieces of information: pitch, velocity, and delta time, which is the time elapsed between the last MIDI event and this event. Specifics on the extraction of pitch, rhythm, and amplitude are in sections II-B1, II-B2, and II-B3, respectively. The important thing to note, however, is that the the data of each feature of a particular player's repertoire was concatenated into a single set, such that all of player p's harmonic information is contained in one set, all of their rhythmic information in another, and all the amplitude information in a third.

1) Feature 1: Pitch: Extraction of pitch information from a MIDI file is fairly straightforward. Since each 'note' is comprised of two messages that give pitch, it is simple to extract the pitch information from a note-on event. MIDI pitches exist in the range between 0 and 127, each number representing a single pitch in 12-tone equal temperament with 60 being middle C. In other words, each MIDI pitch number corresponds to a single key on the piano but extends beyond the piano range in both directions.

For a given song, the entire track containing the desired instrument is parsed and pitches are saved in an array in the order in which they appear in the solo. Since a MIDI file also contains key information, each song was transposed to the key of C (harmonic changes throughout the song notwithstanding). The pitch sequences were then converted to sequences of semitone intervals to further refine the pitch information and render it harmony-agnostic resulting in an array of intervals that is the length of the number of notes in the song minus one. Extracting, combining, and normalizing the pitches in this way - that is, apart from harmony - rendered any notion of harmonic progression null. Even if an entire network were trained on one players repertoire on a single set of harmonic changes, it was not possible to keep track of these changes in the order they occur in the song when sampling the model.

This process was repeated for each song of a particular player's solo in the dataset. The resulting interval sequences were then appended together to give a long sequence of intervals that is the length of the sum of all the notes in all the solos in the dataset minus the number of songs in that particular players' set of songs.

2) Feature 2: Rhythm: Rhythmic information can be deduced in a MIDI file by examining the delta time values of sequential MIDI events. Each MIDI message contains a time value (better understood as delta time) which is the time in ticks per beat that elapsed between the previous event and this event. That is, if a MIDI message contains a time value of 100, we know that this event or message occurs 100 ticks after the previous message. Using this information, it is relatively trivial to deduce the duration of a note or rest when complementary note-on and note-off events can be found. To be sure, if a note-on event of pitch p occurs at time t and the complementary note-off event occurs with a delta time value of 100, the length of the note of pitch p is 100 ticks long.

In an important deviation from standard musical practice, the datasets did not include rests, or silences in a musical line. It is common knowledge that music is not a steady stream of sound; rather, it is comprised of both sound and silence. However, while it was fairly trivial to include rests in the pre-processed rhythmic data, the results from said data were unusable in a musically meaningful way. Therefore, the choice was made to not include rests in the data and thus, not include rests in the generated rhythmic data. This is an obstacle that would be needed to overcome in further iterations.

Once rhythms were reduced to ticks, the next step was to convert these values to fractions of a beat. The ticks per beat value in a MIDI file varies across different files depending on the time resolution needed for a particular song. Given that fact, the particular value of ticks per beat in a given song could not be relied upon to be consistent between and among different files. Thus, each delta time which represented rhythm was divided by the ticks per beat value in its respective file, giving a ratio of a beat that becomes tempo and ticks per beat agnostic. This allows the network to be training on musically more meaningful information as opposed to literal time as measured in milliseconds. In the same way pitch information was gathered, the rhythmic data of each song was concatenated to create a large set of rhythmic information of a particular player, independent of tempo, literal time, and ticks per beat.

3) Feature 3: Amplitude: In MIDI-space, amplitude is called velocity and is given as a value between 0 and 127. 0 is essentially silent while 127 is the maximum amplitude. Important to note is that unlike a real instrument, different amplitudes do not change the spectral or timbral characteristics of the sound. It is merely softer or louder without the sound quality being different in any way. In a real instrument, for instance, playing softer often reduces the presence of high frequency harmonics while the MIDI sound contains the same harmonic profile whether it is loud or soft.

Velocities, then, are also part of the note events. For a note-off event, they are always 127 while the velocity value of note-on event determines the actual amplitude of the note. In the same way pitch was extracted from note-on event, velocity too was extracted from the same events. Moreover, since the velocity value is independent of both harmony and rhythm, there was no need to normalize the data since it is, in a sense, already normalized. Like II-B1 and II-B2, the resulting velocities were concatenated into a single set comprising the entirety of the velocities of a particular player's repertoire.

III. Basic Architecture

Feed-forward neural networks consists of 2 or more layers of processing units, each with weighted connection to the next layer. Each unit passes the sum of its weighted inputs through a non linear sigmoid function. Each layer's outputs are fed forward through the network to the next layer, until the output layer is reached. Weights are initialized to small initial random values. Via the back-propagation algorithm, outputs are compared to targets, and the errors are propagated back through the connection weights. Weights are updated by gradient descent. Through an iterative training procedure, examples(inputs) and targets are presented repeatedly; the network learns nonlinear function of the inputs. We explore these networks for the purpose of processing musical features.[4]

A. Recurrent Neural Networks

Recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This allows it to exhibit dynamic temporal behavior. Unlike feed-forward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. These networks use shared parameters across time, thereby providing these networks with capability to remember time history.

A recurrent network uses feedback from one or more of its units as input in choosing the next output. This means that values generated by units at time step $t-1$, say $y(t-1)$, are part of the inputs $x(t)$ used in selecting the next set of outputs $y(t)$. A network maybe fully recurrent; that is all units are connected back to each other and to themselves. Or part of the network may be fed back in recurrent links. [4]

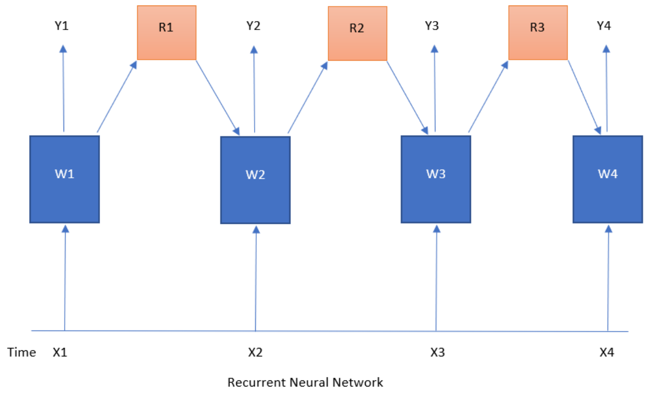

Fig. 1. Basic Recurrent Neural Networks

Figure 1 shows a typical RNN. Here if the sequence is stationary then the network simplifies where weights simplifies as $W = W1 = W2 = W3 = W4$ and recurrent connections simplifies as $R = R1 = R2 = R3$. In recurrent neural networks we need to back propagate through time. Due to a lot of correlated updates per iteration may cause the problem of exploding or vanishing gradients. The problem of vanishing gradient is solved using LSTM (long short term memory) cells instead of regular neurons as is described in next section.

B. LSTM

Long Short-Term Memory (LSTM) is a recurrent neural network (RNN) architecture that has been designed to address the vanishing and exploding gradient problems of conventional RNNs. Unlike feed-forward neural networks, RNNs have cyclic connections making them powerful for modeling sequences. They have been successfully used for sequence labeling and sequence prediction tasks, such as handwriting recognition, language modeling, phonetic labeling of acoustic frames. [1]

The LSTM network is a significant departure from other networks in that it uses hidden layer of memory blocks that can be thought of a complex processing units as shown in figure 4. Departing from typical notion of neuron units that sums its weighted inputs and passes them to a non linear sigmoid function, each LSTM memory block contains gating units. The Write gate learns to controls when inputs are allowed to pass in to the cell, the Read gate learns to controls when cell's outputs are passed out of the block, and the Forget gate learns to control when to reset the memory.

The weight updates for each block of the LSTM network are complex because of the use of the n memory cells and the three gates (described above) that control these n cells within each block. Over and above each output unit of the whole network has a set of weights used to multiply the values coming from the memory blocks. Each gate has a set of weights that it uses to multiply its inputs (recurrent inputs from all the memory blocks and also external inputs) and then pass through a sigmoid function. [3]

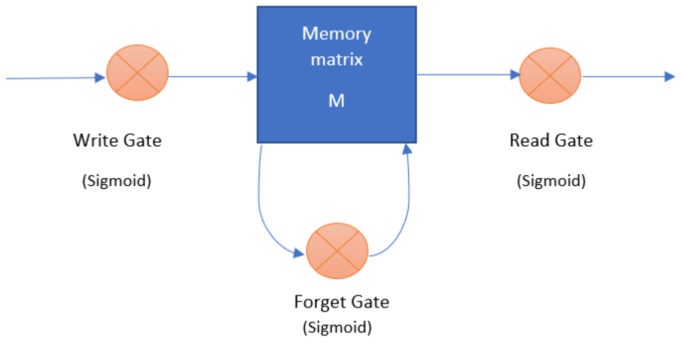

Fig. 2. Model Architecture

C. Model Description

We have designed our system such that a separate recurrent neural network is trained on each of the features extracted from a soloists repetoire. We then generate the feature using a starting seed and recombine them to create a new solo over an ambiguous harmonic structure. To achieve a fusion of different musical styles we have taken an approach wherein we generate the selected characteristics, pitch for instance, of a particular player A, and a different characteristic, rhythm, of of a particular player B and so on. We then combine these features using a Generation Engine to recover a solo as shown in figure 3

$$Song_A = feature_A 1 + feature_A 2 + feature_A 3$$

$$Song_B = feature_B 1 + feature_B 2 + feature_B 3$$

$$Song_{Fusion} = feature_A 1 + feature_B 2 + feature_A 3$$

We first pass the music files through a feature extraction engine as described in II-A. We then select the musical features that we are interested in combining from the music files. These features are then used to train the neural networks with a tunable memory parameter. Once trained these networks are used for generation of the learned features. These features are then passed through a Generation engine that works opposite to the Feature extraction, and combines the features to generate a musical file.

1) Model for Pitch: We train a LSTM recurrent neural network using the pitches extracted from musical composition as described in II-B1. Here we use a sliding memory window of size M. We create training set by sliding the window over the entire interval and can be represented as follows:

$$feature = pitch_t, pitch_{t+1}, ..., pitch_{t+(M-1)}$$

$$label = pitch_{t+M}$$

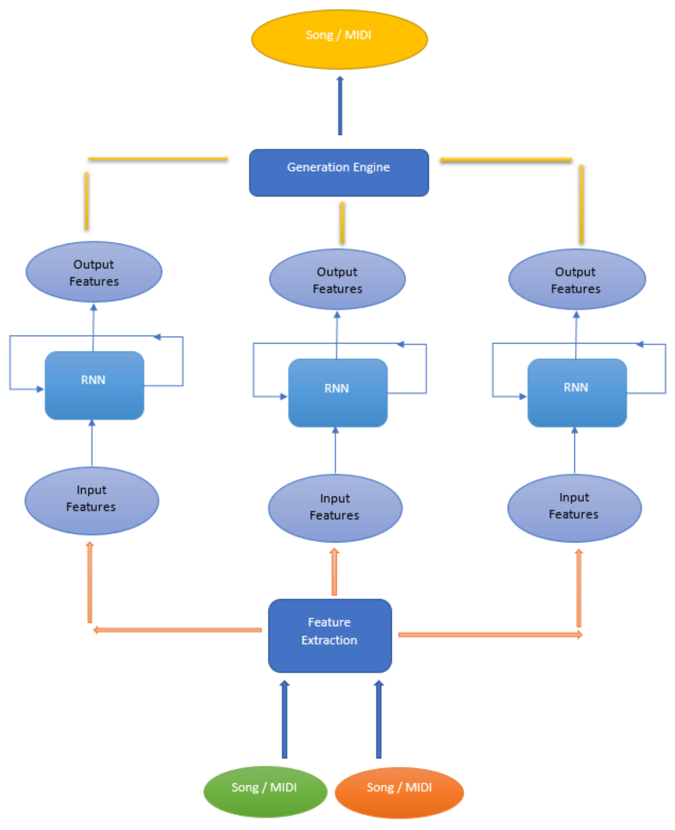

Fig. 3. System architecture

We generate similar feature-label pairs from the soloist feature repertoire. Then we use a deep recurrent LSTM network with 256 nodes, one hidden layer, a dropout of 0.2 coupled with a `softmax` activation function. We use `categorical cross-entropy` as our loss function.

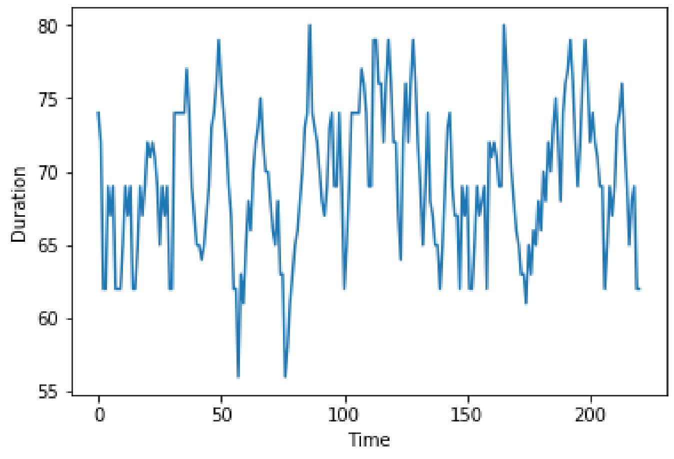

Fig. 4. Time Variation of Original Pitches

2) Model for Rhythm: We train a LSTM recurrent neural network using the rhythmic features extracted from a repertoire

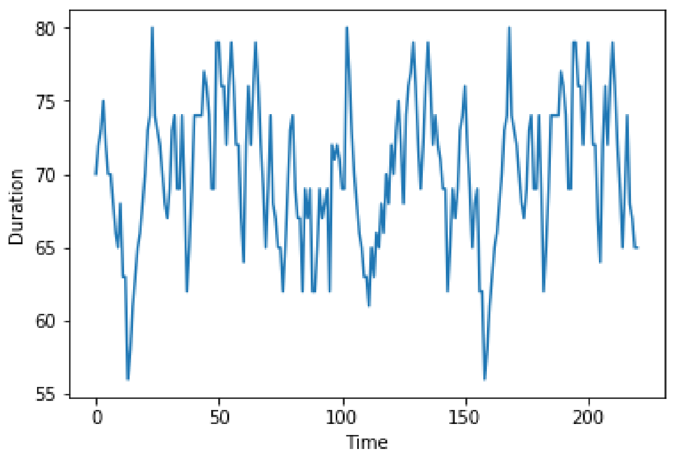

Fig. 5. Time Variation of Generated Pitches

of a player as described in II-B2. Here we use a sliding memory window of size M. We create training set by sliding the window over the entire interval and can be represented as follows:

$$feature = rhythm_t, rhythm_{t+1}, ..., rhythm_{t+(M-1)}$$

$$label = rhythm_{t+M}$$

We generate similar feature-label pairs from the soloists feature repertoire. Then we use a deep recurrent LSTM network with 256 nodes, one hidden layer, a dropout of 0.2 coupled with a `softmax` activation function. We use `categorical cross-entropy` as our loss function.

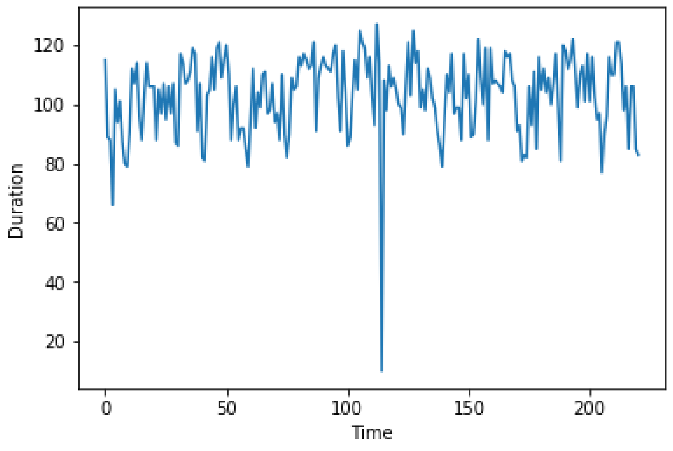

Fig. 6. Time Variation of Original Rhythms

3) Model for Amplitude: We train a LSTM recurrent neural network using the velocities extracted from musical solo as described in II-B3. Here we use a sliding memory window of size M. We create training set by sliding the window over the entire interval and can be represented as follows:

$$feature = amplitude_t, amplitude_{t+1}, ..., amplitude_{t+(M-1)}$$

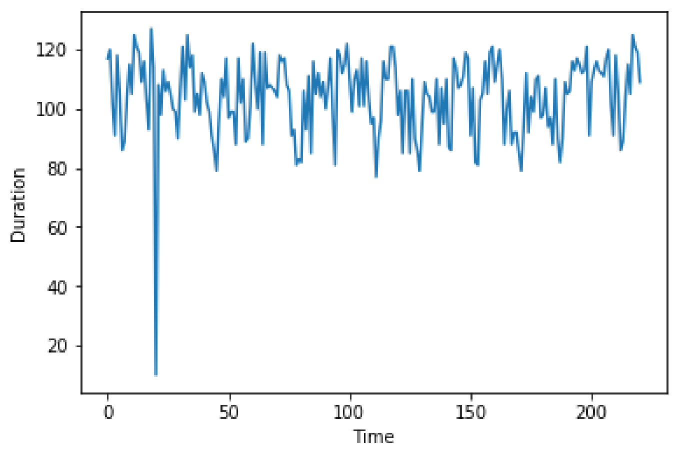

Fig. 7. Time Variation of Generated Rhythms

$$label = amplitude_{t+M}$$

We generate similar feature-label pairs from the musical composition. Then we use a recurrent LSTM network with 4 nodes. We use `mean squared error` as our loss function.

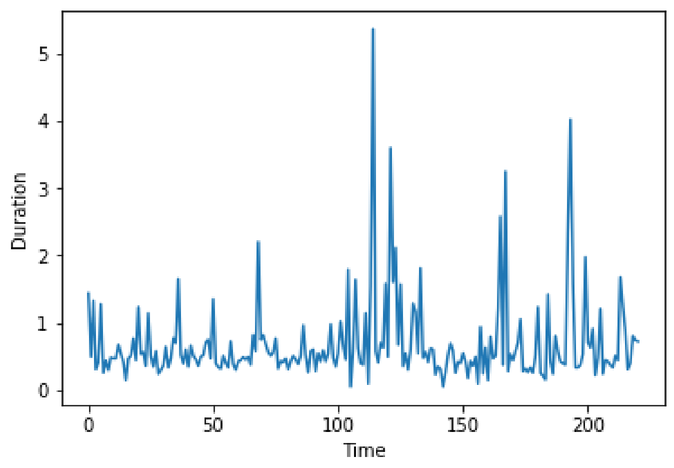

Fig. 8. Time Variation of Original Amplitude

IV. Experiments & Results

We experimented the with learning the features of a musical composition as described in II-B above. We have mainly utilized Keras [15] for training the neural networks.

After the data was processed, it was used to train models with 500 epochs, a batch size of 32, and a memory of 20. That is, a model was trained with the aforementioned parameters on a players pitch, rhtyhm, and amplitude; i.e. Miles Davis' pitch, Miles Davis' rhythm, and Miles Davis' amplitude. This was the case for each of the four players we processed: Miles Davis, John Coltrane, Ornette Coleman, and Charlie Parker. The reasons for choosing these particular

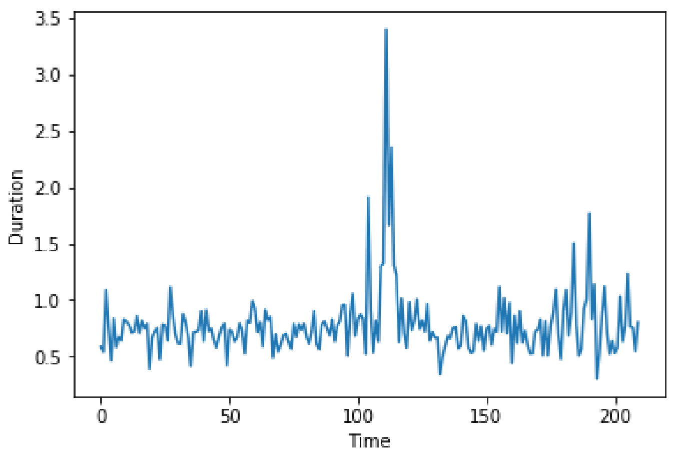

Fig. 9. Time Variation of Generated Amplitude

players are noted in II.

Perhaps the most valuable parameter available is the memory value. Most of our experiments were performed with a memory window M of size 10. However, this resulted in too quick a convergence. We experimented with values up to 50, as well as a dynamic memory of 0.05, 0.1, and 0.2 the size of the dataset. While higher memory values often gave more interesting results and no convergence, the training process took far too long even on a GPU. We settled on a value of 20 which gave us later convergence and a relatively fast training time. Moreover, a memory of 20 ensured that we do not look so far back in the stream of notes so as to get something unrelated, though musical phrases are often not nearly as long as 20 notes in a straight-ahead jazz context. Experiments were also carried out with a smaller number of epochs but this yielded unmusical results as the values, similar to small memory windows, converged too quickly to be useful.

Musically, a memory value is difficult to justify. In a real musical context, a player doesn't necessarily look back over the number of notes they have played; rather, they listen to the musical context and, given what they have played before, make choices on the fly. This backward looking can go even as far back as several minutes, depending on the length of their solo. In other words, a real player does not choose what to do next given a backward looking window of notes or time, they take their context and past, however long, into account.

After training the models, models representing different features were sampled. These were then combined to create a new solo over harmonically ambiguous terrain. The results were mixed: rhythmically, the individual players' styles stood out, as expected. Harmonically, given the ambiguity, the results were less than pleasing. What often resulted, since there were no rests, was a constant stream of notes, what my jazz guitar instructor once referred to as ”melodic diarrhea”. Phrases of any discernable length were impossible to hear and, while it could be discerned what feature came from what player, they were on the whole musically unsatisfying.

V. Future Scope

We have restricted our work to use of three features described in section II-B. In general, a musical work contains many features which can be used in feature extraction. These include spectral centroid, high frequency energy, high frequency content, spectral irregularity, spectral flux and running entropy, vector-based bandpass filter envelopes are Fourier transform with a sliding window.[13] These features can be used to enhance the learning and generation process. It is important to note, though, that these measurements and features are only meaningful if they are taken from an original sound recording rather than a MIDI reproduction.

Attempts were also made to produce a quasi-time series set such that the sample values were the MIDI pitches, silence being -1, and each sample representing a point in time of the file. This had the advantage of encoding both the rhythmic and pitch information in the same set. However, using an LSTM on this yielded unusable results. Nonetheless, it is perhaps an avenue for future experiments.

VI. Conclusion

The current work has demonstrated that recurrent neural networks (RNNs) in combination with long short term memory (LSTM) cells, has the capability to capture temporal dynamics of a musical composition. Although not precise reproduction but we achieved very similar reproduction of the certain features, when we seeded the network with beginning of time t_0 for a musical composition. Our motive in this paper was to focus on generative aspects of the musical composition so we seeded the network randomly which resulted in generation of different samples. The periodicity of feature patterns are preserved in such cases, but the the coordination information seems to be lost when we try recombination of the same.

This is in accordance with out expectations as we did not want an entire reproduction of the samples. We wanted the sensitivity of out network to be high, but also a good amount specificity so that the network could learn a generic style which could then be used in combination with counterpart features from fundamentally variant data points. Our experiments yielded good results when we restricted the features to being pitches, rhythms, and amplitude as defined in the section.II-B Although we would prefer keeping the model simple, we expect improved results by taking into account an increased number of uncorrelated features. On the basis of our experiments we can say that the technique presented here can definitely be used for fusion of different musical styles when the characteristics of the style can be reduced to features such as pitch, rhythm, and amplitude. We are eager to experiment with more complex architectures and variant styles to see how well out preliminary results evolve on tuning various parameters.

References

[1] Hasim Sak, Andrew Senior, Francoise Beaufays, *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling*. Google, USA

[2] Bob L. Sturm, Joao Felipe Santos, Iryna Korshunova, *Folk music style modelling by recurrent neural networks with long short term memory units*. Late-breaking demo at the 2015 Int. Symposium on Music Information Retrieval.

[3] Judy A. Franklin. *Recurrent neural networks for music computation*. ORSA journal on computing, 18(3):321338, 2006

[4] Judy A. Franklin, Krystal K. Locke, *Recurrent Neural Networks For Musical Pitch memory and classification*. International Journal on Artificial Intelligence Tools. July, 2004

[5] Judy A. Franklin. *Jazz Melody Generation from Recurrent Network Learning of Several Human Melodies*. American Association for Artificial Intelligence, 2005

[6] Bharucha, J. J. and Todd, P. M. *Modeling the perception of tonal structure with neural nets*. Computer Music Journal, 13(4):4453.(1989)

[7] Douglas Eck, Jŭergen Schmidhuber. *A First Look at Music Composition using LSTM Recurrent Neural Networks* Instituto Dalle Molle di studi sullíntelligenza artificiale Galleria 2 Technical Report No. IDSIA-07-02

[8] Stevens, C. and Wiles, J. (1994).*Representations of tonal music: A case study in the development of temporal relationship*. In Mozer, M., Smolensky, P., Touretsky, D., Elman, J., and Weigend, A. S., editors, Proceedings of the 1993 Connectionist Models Summer School,pages 228235.

[9] Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. In Kremer, S. C. and Kolen, J. F.,editors, A Field Guide to Dynamical Recurrent Neural Networks.IEEE Press

[10] B.Laden and D.H.Keefe. *The Representation of Pitch in a Neural Net Model of Chord Classification Music and Connectionism*, eds. Todd, P.M. and Loy, E.D.,(MIT Press, Cambridge, MA, 1991)

[11] Abeer, Jakob and Frieler, Klaus and Pfleiderer, Martin and Zaddach, Wolf-Georg (2013), *Introducing the Jazzomat project - Jazz solo analysis using Music Information Retrieval methods*. Conference: Proceedings of the 10th International Symposium on Computer Music Multidisciplinary Research (CMMR),

[12] The Jazzomat Research Project, *Database Content*. available at ://jazzomat.hfm-weimar.de/dbformat/dbcontent.html.

[13] K. Jensen and T.H. Andersen. *Real-time beat estimation using feature extraction*, Jensen2003RealTimeBE 2003

[14] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[15] Chollet François and others, *Keras* https://github.com/fchollet/keras 2015