JAZZ BASS ACCOMPANIMENT AND SOLO GENERATION

Machine Learning Final Project

Yu-Ting, Tsai
Department of Computer Science
National Tsing Hua University
Hsinchu City, Taiwan (ROC)
A38050787@gmail.com

Yi-Chieh, Chiu
Department of Computer Science
National Tsing Hua University
Hsinchu City, Taiwan (ROC)
shes202@gmail.com

Kevin Richardson Halim
Department of Computer Science
National Tsing Hua University
Hsinchu City, Taiwan (ROC)
k.richardsonhalim2002@gmail.com

***Abstract*—Machine Learning has been prevalent recently. However, instead of the well-known application, such as picture recognition and category prediction, we try to implement a model that can generate bass automatically for jazz music. Even though it may not seem to be the most lucrative application of Machine Learning compared to Some of the most prominent fields of Natural Language Processing (NLP,) Computer Vision, and Quantitative trading, we still see a bright future in the lot of introducing machine learning to music. In this paper, we use bidirectional LSTM to deal with this task. In addition, we also put some effort into data preprocessing and experiments.**

Keywords—jazz, bass, bidirectional, Recursive Neural Network (RNN)

I. INTRODUCTION

Jazz has become a trendy music genre after its birth in the late 19th and early 20th centuries in the African-American communities of New Orleans, Louisiana. Swing and blue notes, complex chords, call-and-response vocals, polyrhythms, and improvisation characterize jazz. In addition, jazz music is known for its intricate and improvisational bass lines, which add depth and complexity to the music. However, creating original and fitting bass lines for jazz can be challenging, even for experienced musicians.

We choose jazz music because it isn't too complex to understand, and its structure is simpler than classical music. Moreover, the chords of jazz are analyzable. By decomposing every note, finding the pattern the composer wants to construct is not too difficult.

There are a lot of jazz music resources available on the internet. Thanks to the midi (Musical Instrument Digital Interface) file, we can translate music to the desired format based on several python modules built for midi files.

In this project, we aim to use machine learning techniques to generate bass lines for jazz music automatically. By training a model on a dataset of existing jazz bass performances, we hope to create a tool that can generate new and original bass lines that fit the harmony and structure of jazz music. For instance, Let a piece of jazz music be A + B, B = music of double bass, and A = music played by other instruments (piano, guitar, drums, etc.). We want the model to learn the relationship between A and B. If we input A, the output will be B'. We want B' to be as similar to B as possible. We then combine B' and A to get the complete song. In our project, we tested several methodologies to achieve this goal. In the following sections, we'll introduce Recursive Neural Network (RNN) and some augmentation on the data set and model.

II. METHODS

A. Input format

Our input format contains the following elements:

- chromagram 12x3
- speed (integer)
- number of instruments playing simultaneously (integer)
- is bass playing (boolean value)
- time signature (two integers)
- beat position (int)
- bar position (int)

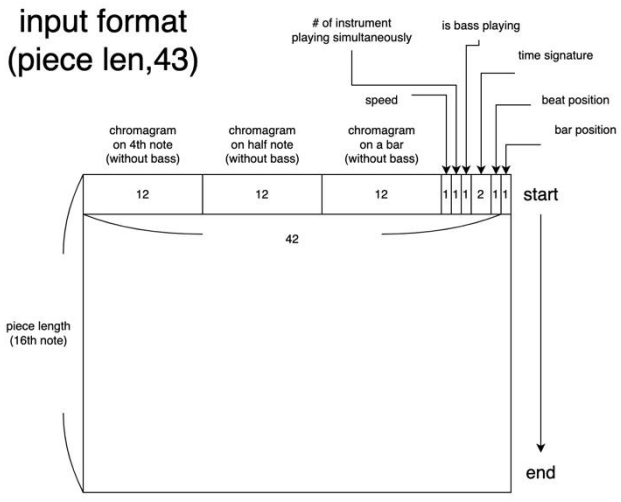

Fig. 1. Input format figure

From Fig. 1, we can see that each row contains elements representing a song's information. Chromagram can be seen as a feature that gives a unique index to every pitch. With the knowledge that a set of notes {C, C♯, D, D♯, E, F, F♯, G, G♯, A, A♯, B} can be viewed as twelve-pitch spelling attributes, we can assign chroma C with index 0, and chroma C# with index 1, for example. We record the chromogram on the 4th note, the half note, and the bar because this is usually the pattern and basic unit of a chord change. By recording these notes, we can catch the trend of the music.

We wish the model could use the rest of the features and improve the training result since they are all essential features representing a song, such as speed, time signature, beat, and bar position. For beat and bar position, beat position records the location of the music this row element represents. For example, for a time signature which is Four-Four Time, since each row element is recorded on the frequency of 16th notes, then the 16th row element will have beat position 15 (starting from 0), and the 17th row element will have beat position 0 since it is the start of another bar. For bar position, it's just which bar the row element resides in the music, so for the previous example, the 15th and 16th row elements both have bar position 0, while the 17th row element has bar position 1. Beat position and bar position can also help the model catch the music's trend. For example, at the beginning of most jazz music, the music is smooth, and it will become exciting or exaggerated in the middle since this is usually the theme of the music. Then, by recording the bar position, the model can try to learn the relationship between notes and bar position. Similarly, the beat position has the same feature but with a smaller scale.

As for the feature " number of instruments playing", it aims to catch the trend of bass affected by the number of instruments playing. For example, when multiple instruments are playing simultaneously, we can often notice that the bass has a relatively simple rhythm since it is accompaniment. On the contrary, the bass becomes the main character when no other instruments are playing. Great bass performers often improvise at this moment. However, it's not entirely improvised because they still need to follow specific patterns, such as tonality. Another feature is " is bass playing", and it is pretty straightforward. Nevertheless, it is worth noticing that we measure this feature bar by bar; this is to avoid that there will be a long section without bass in some arrangements (for example, another instrument is soloing), which is almost impossible to learn, so we marked it out.

Each column represents a certain period of music, and the sampling period is 16th note-long. For example, the leftmost chromogram will change every four rows since one 4th note is as long as four 16th notes. Similarly, the chromogram on the half note will change every eight rows since one half note is as long as eight 16th notes. Of course, we can sample the music with the unit 32nd note-long, but it's not practical. From the musical perspective, the 16th note is more often viewed as the basic unit rather than the 32nd note, and if we adopt the latter period, the input and output size of our data will be huge.

Another concern is why we only need 12 pitches to represent a song. The reason is that humans perceive two musical pitches as similar if they differ by octave. Hence, from the view of chords, they are not much different. Since the number of pitches of a song may be huge (piano has 88 keys that represent 88 pitches!), by reducing it to only 12 pitches, we can vastly reduce the size of the input format.

B. Output format

Since our task is to predict the bass part of jazz music, we use one-hot encoding to represent the pitches that may be played by the bass, as shown below.

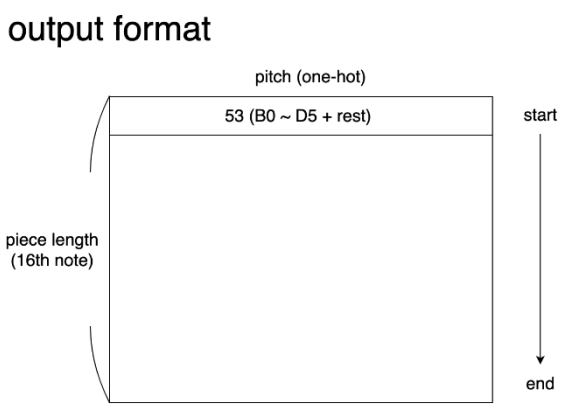

Fig. 2. Output format figure

C. Data Set

As mentioned before, we use the midi file as our raw input data. This is because the midi (Musical Instrument Digital Interface) file is a well-defined technical standard that can illustrate and record music perfectly. Not to mention there are a lot of resources and python packages that can support our implementation. We tried three data sets listed below, and I'll explain their advantages and disadvantages.

- minor9 jazz standards + Jazz ML ready MIDI

  These data sets are all midi files, and since there's no position for the rest note, we use the position of D5(the highest note) instead.

- midikar dataset

  For validation purposes.

- iReal Pro dataset

  Since the result trained by the original midi files was poor, we tried another midi dataset, the iReal Pro dataset. All the midi files of this dataset are computer generated, and their purpose is for music practice, so the structure is simpler than our original midi files. First, however, we must emphasize that even though the dataset is computer generated, it has nothing to do with machine learning. Instead, it follows a particular pattern to generate a large amount of canned music.

After the completion of our model, we perform transfer learning on the model with the iReal Pro dataset. The result is solid, so we trained the model with other midi files, and the model still works fine. Thus, we combine minor9 jazz standards, Jazz ML ready MIDI, and iReal Pro for the final dataset.

D. Data Augmentation

We also transpose these midi files to 12 keys to enlarge our dataset. The concept is that if we transpose a music piece to another key, it would still sound reasonable.

In practice, since it's not easy to recognize the key of a music piece from a MIDI file, and the jazz genre is known for its complex key changes, so we shift all notes (except drums) in a MIDI file up or down the same distance, and discard ones that exceed pitch range that bass can play.

We didn't do data augmentation on the validation set.

E. Data Preprocessing

We mainly use the python library "pretty_midi" to process midi data. We write a program to automatically process all midi data to ".npy" files which fit in input and output format.

We first find the bass instrument track and make its content as ground truth. Since the MIDI files we use use "general MIDI" specifications, it's not hard to find the bass track. Although some pieces have no bass instruments, we discard them.

After that, we generate the data, which will be inputted to the model using other content, involving notes on different tracks and some midi event, such as tempo_changes (to get the speed) and time_signature_changes.

F. Model

We use a bidirectional LSTM (long short-term memory) to implement our model. The detail of the model is as follows.

Layer	Output Shape	Number of Parameters
Input Layer	(None, 128, 42)	0
Masking	(None, 128, 42)	0
Bidirectional	(None, 128, 400)	388800
Time Distributed	(None, 128, 52)	20852

Total parameters: 409, 652

Trainable parameters: 409, 652

Non-trainable parameters: 0

Table 1. Model description

III. Results

Our model reaches an accuracy of 0.8873 with 500 epochs and 3 hours of training. Note that the predicted result generated by the model is only the bass part; we then combine the bass with other instruments to form a complete song by replacing the bass notes in the original MIDI file with notes that the model generated. However, there are some interesting things: unnatural short notes jump up and down and many long notes because we cannot make pitch repetition in the output.

Since there is still time before the deadline of this project, we think it's a good idea to raise the difficulty of our implementation. Recall that we use chromograms to represent jazz music. Jazz music is nothing special from other music; they are all composed of chords. A chord can be viewed as a structure, and the most critical component of the structure is called the root. In music theory, the root is the idea that a chord can be represented and named by one of its notes. It is linked to harmonic thinking—the idea that vertical aggregates of notes can form a single unit, a chord [2]. We suspect that the root may be a hint for the model to predict the bass part, so we want to explore what will happen if we remove the root. Will the model still perform perfectly? If not, what method can be done to enhance the model? This will be covered in the Experiment section.

IV. Experiment

After removing the root of the chords from all the chromograms, we have done some experiments listed below.

A. Experiment 01

We use Adam (adaptive moment estimation) as an optimization algorithm for our bidirectional LSTM model. The result is terrible, and we discover there might be overfitting and gradient exploding issues. We figured out two ways to improve this model. First, enhance the quality of our input dataset. Second, improve the capabilities of our LSTM model. We decided to focus on the second improvement, and the corresponding experiments are as follows.

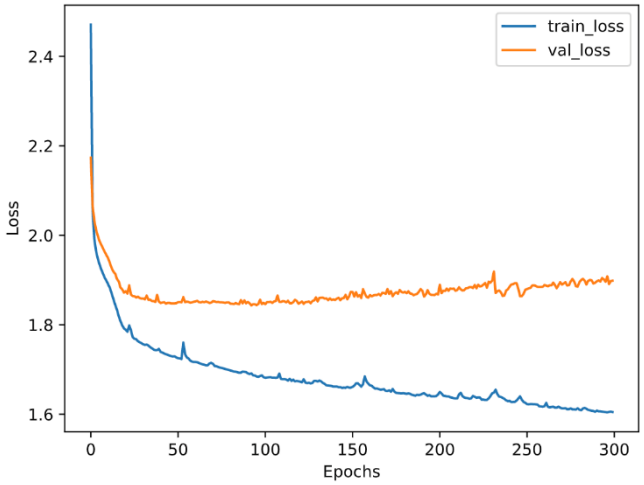

Fig. 4. The lost diagram of experiment 01

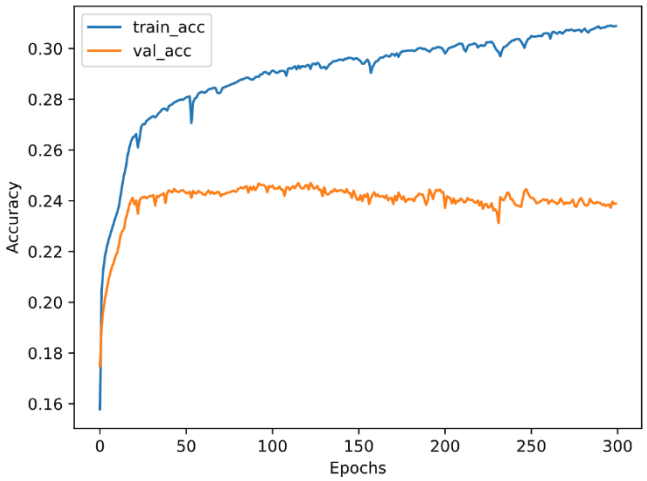

Fig. 5. The accuracy diagram of experiment 01

B. Experiment 02

In this experiment, there are two deep layers and one wide layer, and the units remain the same. The improvement and corresponding reasons are as follows.

- Add a dense layer to input

  The original chromogram lacks the chord root and is not complete, so it would be great if the input dense layer could play a complementary role

- Add a dense layer to output

  We hope LSTM can focus more on memorizing information

- Regularization

  The previous experiment had a serious overfitting issue, so l2 was added to the dense layers, and the LSTM kernel also has it, but it is relatively small (0.0001).

- Gradient clipping

  After running the previous experiment, the problem of gradient explosion is too severe to ignore.

- Nadam

  Because of the gradient explosion problem, we think Nadam, another optimizer that replaces momentum in Adam with Nesterov momentum, is better.

Note that all dense layers have batch normalization. The result of the above experiment shows that regularization did its job; however, gradient explosion is getting worse.

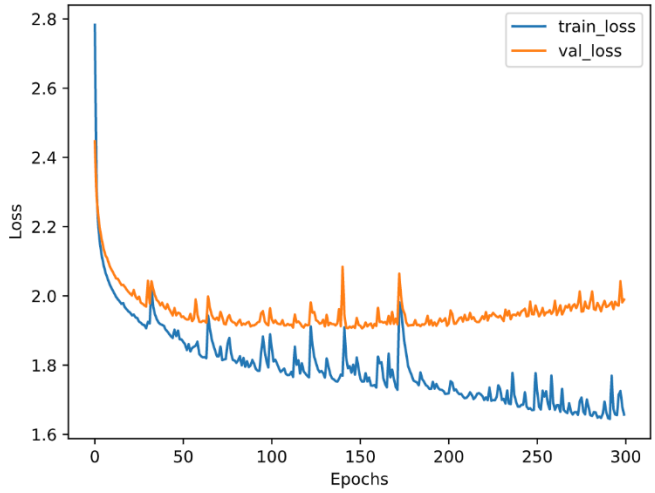

Fig. 6. The lost diagram of experiment 02

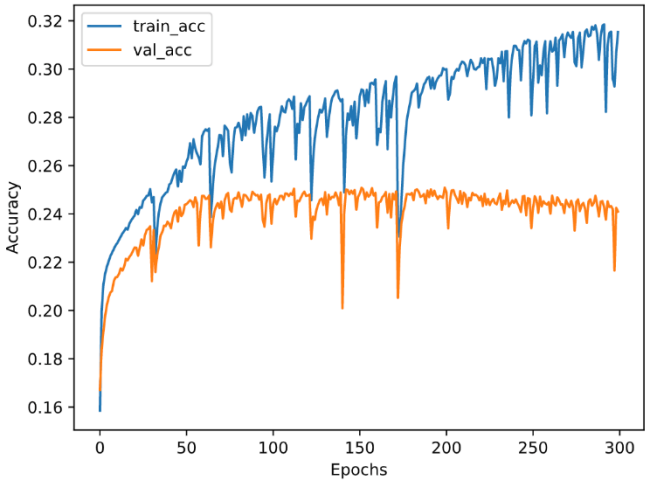

Fig. 7. The accuracy diagram of experiment 02

C. *Experiment 03, Experiment 04*

In experiment 03, we tried to add regularization to Nadam, and in experiment 04, we added a thicker FeedForward before and after LSTM and added regularization. Unfortunately, both experiments ended up a failure.

D. *Experiment 05*

In this experiment, we simplify experiment 04, leaving only one layer before LSTM and adding a bunch of regularization; the performance is ordinary.

E. *Experiment 06*

In this experiment, we try to find a better combination of two hyperparameters, piece_length and LSTM_unit.

When we train the model, the sequence length must be the same to use mini-batching. Therefore, we split and pad every training data to the same length, which is the piece_length. However, since the separated pieces are treated as an independent pieces in the training process, it will affect the result. If piece_length is too big, it would be difficult for LSTM to train. On the other hand, if piece_length is too tiny, LSTM cannot learn the long-term structure of music.

LSTM_unit is the output width of LSTM (half of the width of bidirectional LSTM), and it also affects the width of the dense layer before LSTM. A bigger LSTM_unit could make better model expressiveness, but it also takes more time to train and has a higher risk of overfitting.

We tried three different values of piece_length. In a Four-Four-time piece, 1024 is as long as 32 measures, 128 is as long as eight measures, and 16 is only a measure long. We tried two different values of LSTM_unit, 50 and 100. To save time, we only train 100 epochs in these experiments.

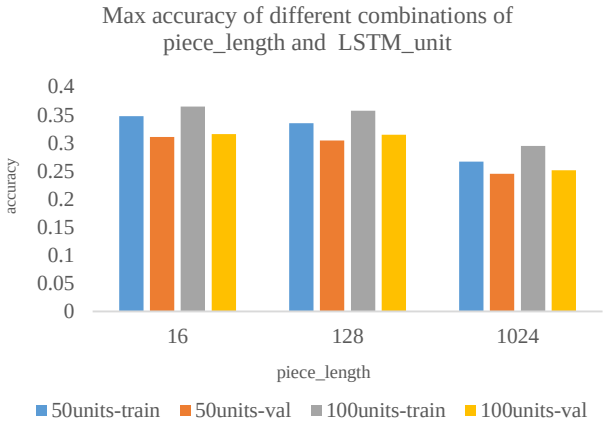

Fig. 8. Max accuracy of different combinations of piece_length and LSTM_unit.

In this experiment, we have several observations. First, in this scale, a larger LSTM_unit is better. Second, the smaller the piece_length, even if we train and predict one measure at a time, the better. This is out of our expectations, and we think the reason is that the long-term structure doesn't help much at the current accuracy.

However, even though piece_length 16 has slightly better accuracy than the 128 one, there is a problem that doesn't show on the accuracy score. When testing, the model would generate a whole piece. It will first split and pad the piece into many segments with the same length as piece_length, then inference them independently and concatenate them in the end. Since 16-piece_length one divides the piece into segments a measure long, the result of it is less smooth than the result of 16-piece_length one.

Therefore, though setting piece_length as 16 could obtain slightly higher accuracy, we set piece_length as 128 and LSTM_unit as 100 in the following experiments.

F. Experiment 07

We use the transfer learning technique in this experiment to obtain higher accuracy.

The data we use for previous experiments are real-world data. Since there are many "creative" arrangements, they may not be a good source for early-stage learning. Therefore, we train the model on the iReal Pro dataset first, then fine-tune it on real-world data.

As we explained before, the iReal Pro dataset consists of algorithm-generated accompaniment. As a result, they are less "creative" but more "reasonable." On the other hand, it only contains piano, bass, and drums, so the input is also cleaner. Therefore, we expect the model to learn basic music theory from this simpler model and then learn more complex rhythms or creativity on real-world data.

We first train the model on the iReal Pro dataset for 100 epochs and then train it on real-world data for 100 epochs.

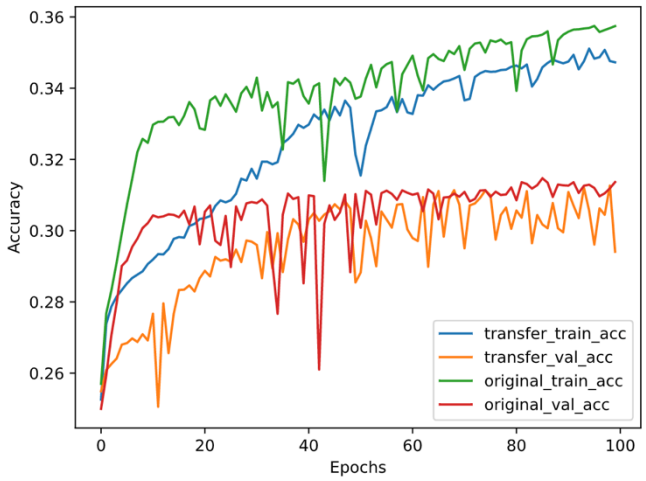

Fig. 9. The accuracy curve comparison of transfer learning one (after pre-training on the iReal Pro dataset) and the original one

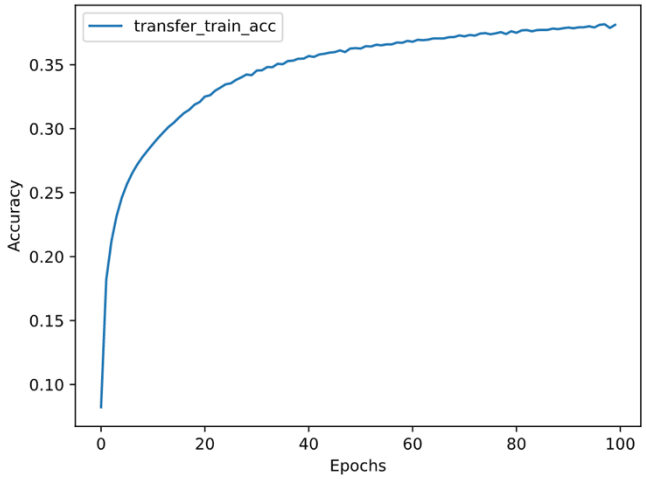

Fig. 10. The accuracy curve of the pre-training phase

However, the result isn't good as we expected. As Fig.9 shows, the accuracy is worse with transfer learning. It probably results from the difference between real-world data and the simpler one being too big. Another reason may be our pre-training is not that successful. As Fig.10 shows, though the learning curve is much smoother than training on real-world data, its final accuracy doesn't exceed training on real-world data. It may result from the fact that we didn't tune hyperparameters on the iReal dataset, or our model is still too weak to work on this task, even on a simpler dataset.

G. Experiment 08

At the end, we use the best setting, while we only enlarge the LSTM_unit from 100 to 200, to train our final model. We train it without pre-training on iReal Pro dataset for 300 epochs.

Layer	Output Shape	Number of Parameters
Input Layer	(None, 128, 43)	0
Masking	(None, 128, 43)	0
Time Distributed Dense	(None, 128, 200)	8800
Bidirectional	(None, 128, 400)	641600
Time Distributed	(None, 128, 53)	21253

Total parameters: 671, 653

Trainable parameters: 671, 653

Non-trainable parameters: 0

Table 2. Final model description

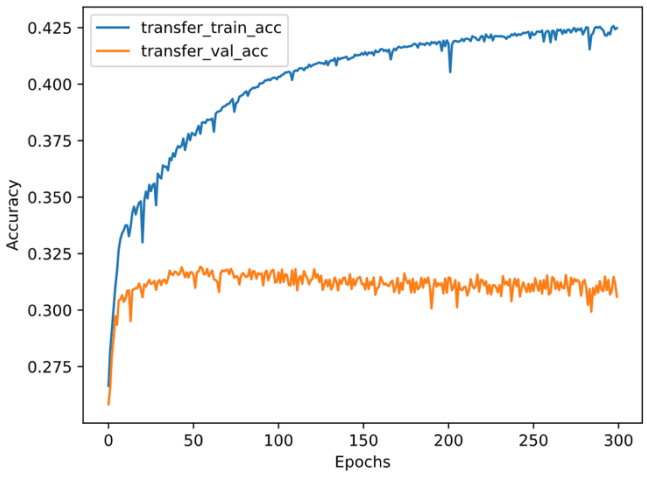

Fig. 11. The accuracy curve of the final model

The final training accuracy is 0.4248, which higher than all previous experiments, however, the final validation accuracy is 0.3059, which is similar to previous experiments.

An example of final model predict result and its ground truth is provided in the following link :

https://drive.google.com/drive/folders/1cU_rKtdnHp3UrXuEbzNb-AFqUl2PaPeR?usp=share_link.

H. Conclusion

This will be covered in the Discussion/Conclusion section below.

V. Discussion/Conclusion

In this paper, we introduced a bidirectional Recursive Neural Network (RNN) to build a model that automatically generates jazz bass accompaniment and solo. The model performs well with an accuracy of 0.8873, and we can conclude that we have achieved our goal. For further study in the Experiment section, we can figure that it's tough for a model to learn the relationship between the music of bass and other instruments without the root of the chords. Although we tried different tricks, such as adding regularization terms, adding a dense layer before and after the bidirectional LSTM layer, and pre-training it on a simpler dataset, the results were not as good as expected. This makes sense since, even for humans, this is also mission impossible. Given a pitch, it has no absolute answer for a person to decide which other pitch should be combined with this given pitch without the root of the chord. The reason is that there may be multiple answers. This can be introduced by another concept, "consonance and dissonance." In music, consonance and dissonance are categorizations of simultaneous or successful sounds. Within the Western tradition, some listeners associate consonance with sweetness, pleasure, and acceptability, and dissonance with harshness, unpleasantness, or unacceptability, although roads led to this on familiarity and musical expertise [2]. This concept may seem too theoretical, but anyone with no music training background can still recognize whether the chord is consonance or not since this is human instinct. Hence, for anyone to guess the corresponding pitch, the goal is to choose another pitch that would be consonant with the given pitch. Unfortunately, there may be multiple choices simultaneously because there is more than one consonance chord. The answer to this question depends on the composer's preference when composing the song. Nevertheless, this is precisely what we wish the model to do, a challenging task even for human beings.

We believe that machine learning is suitable for tasks that humans can do, but it's too costly and inefficient. For example, a human can recognize whether the food on the plate is pizza, but if there are 1000 plates, we train a model to let the computer do it instead of doing it ourselves. Furthermore, our experiment has shown that some tasks that are difficult for a human to accomplish may also be challenging for a machine.

However, in conclusion, the results of this project demonstrate the potential for using machine learning to generate high-quality bass lines for jazz music. Our model benefits humans, especially musicians and the music industry. The model can assist musicians with tedious work so that they can put their focus on high-value tasks. It can evaluate/score a piece of music before any further production and can be viewed as an inspirational and innovative tool for the musician. Overall, this project provides a proof of concept for using machine learning to generate jazz music and opens the door to future research in this area.

VI. Future work

A. Some Direction of Better Input/output Format

- Add drums information into the input since drums play an important role in jazz music.
- Add velocity as output since now the velocity is set by ourselves.
- add onset/offset timing correction into the output
- Incorporating more information about the context of the music, such as the melody or the style of the song, to generate bass lines that are more fitting and coherent with the music.

B. A Newly-designed Model

- Train encoder-decoder RNN with attention since it's a great idea to treat it as a kind of sequence to sequence translation problem
- A transformer is suitable for this project as well.

VII. Author Contribution Statements

- Yu-Ting, Tsai (33.33%)

  Data collecting, model training, programming of the model, final presentation, and writing the final report.

- Yi-Chieh, Chiu (33.33%)

  Data collecting, data preprocessing, model training, programming of the model, and writing the final report.

- Kevin Richardson Halim (33.33%)

  Model training and programming of the model.

References

[1] Root (Chord). (n.d.). Wikipedia. https://en.wikipedia.org/wiki/Root_(chord)

[2] Lahdelma, Imre, and Tuomas Eerola (2020). "Cultural Familiarity and Musical Expertise Impact the Pleasantness of Consonance/Dissonance but Not Its Perceived Tension." Scientific Reports 10, no. 8693 (26 May)