Machine Learning Quarter 2 Project: Accompaniment Generation Model

Ryan Ghimire and Aniketh Luchmapurkar
Thomas Jefferson High School for Science and Technology
Machine Learning 1
Dr. Yilmaz
February 1, 2025

Abstract:

As many people lack the musical knowledge or skill to compose music, we are creating a piano song generation model trained on a dataset of 909 pop songs. This transformer-based model will focus on identifying correlations between songs and their attributes, including keys, notes, and chords; the model will apply these correlations to generate new accompaniment music. We will objectively evaluate our model by observing the set of notes generated by the model and matching them to scales/keys to see how musically "correct" the songs are. Additionally, we will evaluate the quality of the songs themselves by having a sample population knowledgeable in music evaluate the songs.

Introduction:

Many don't know how to create good music to accompany a pre-existing melody. It is a very subtle thing to create effective accompaniments. To combat this issue, we intend on creating a machine learning algorithm that is capable of generating accompaniments to a melody. The input to our algorithm would be a melody sequence (a set of notes) and the previous four chords played. The output would be the predicted chords for that melody sequence. The melody sequence is passed into the model as a tuple, the previous four chords in a list, and the predicted chord is outputted as a string. The model is called repeatedly for a longer set of melody sequences and previous chords, and the final chord prediction is outputted as a list.

Related Work:

Initially, music generation began by using statistical modeling to generate melodies (Conklin, 2003). These models have now evolved to using Transformer and Recurrent Neural Network (RNN) models, one of the first of these models being DeepBach, a RNN model focused solely on generating classical music pieces from scratch (Hadjeres et al., 2017). These models have further evolved to generating varied music styles (Mao et al., 2018), and generating songs based on melodic feature input (Copet et al., 2024). The currently leading AI music generation model, Google Magenta, specializes in varied types of music generation, including chords. However, Magenta's Chord-RNN model is able to generate chord progressions, but not ones that align with a given melody. As mentioned by Briot and Pachet (2020), all of these models lack a major component: the ability to provide user interactivity and creativity in the music generation process. That's why Spotify's Creator Technology Research Lab (CTRL) is focusing on creating AI tools to help artists in composing music. However, there is currently no model released by

them since their inception in 2017 that achieves this. Because of this, there remains a need for tools that do not restrict user creativity in the field of AI music generation.

Dataset and Features:

The dataset being used is the POP909 dataset. This dataset has 909 pop songs, with a folder dedicated to each song. Each folder contains three files of use: beat_audio.txt, chord_audio.txt, key_audio.txt. The beat_audio.txt file has two columns (time and beat order). The chord_audio.txt file has three columns (start time, end time, chord name). Lastly, the key_audio.txt file also has three columns (start time, end time, key played). One should note that all times in these files are recorded in seconds.

All of these files are summarized into a csv file called key_dict.csv in the format indicated below (columns and sample instance).

active_key,melody_sequence,chord,*chord_count,*prev_chord_1,prev_chord_2,prev_chord_3,prev_chord_4
Gb:maj,"(61, 63, 66, 68)",Eb:min,*1,*None,F#:maj,B:maj7,Bb:min7

Subsequently - using keys_dict.csv - melody sequences, previous chords, and chord data were extracted for the model to process. The chord data was used to create chord labels, and that was encoded by use of a label encoder. The other two pieces of data that were extracted (melody sequences and previous chords) are to be used as the features/attributes of the model. Both the previous chords and melody sequences are stored as 2D numpy arrays. However, the melody sequences were first numericized due to keys_dict.csv storing the sequences as a stringified tuple. They were then padded due to varying sequence lengths (all padded with zeros to the maximum sequence length). Of the 909 songs in the main POP909, the first 5 songs were kept aside as test songs and of the remaining 904, 80% were training data and 20% were validation data.

Methods:

Our model is split into two sub-models: one to handle the melody data and one to handle the previous chord data (to put it simply, one model per feature). As such, we will talk about each model separately, and then discuss how they were merged to form one larger model.

The sub-model that uses the melody feature consists of four layers: an input layer, an embedding layer, a transformer layer, and a global pooling layer.

The input layer, as the name suggests, handles the input of the features. Its size reflects the dimensions of the data.

Next is the embedding layer. This layer's purpose is to lower the dimensionality of an input and allows for the ability to draw relationships between multiple parts of the input. It does this by converting a melody sequence that may be hard to decipher into a list of numbers showing the relation/meaning of each piece of the melody sequence.

Following the embedding layer is the transformer layer. This is the most important part of this sub-model. It utilizes the simplified representation of the melody sequences created in the previous layer to create correlations and relationships. The way this is done is through an encoder and decoder. The process of calculating "importance" of each part of an input is known as self-attention. The encoder uses multi-head attention (multiple parallel runs of self-attention that are averaged) to determine importances in the melody sequence, and the decoder will create chords with a similar set of importances.

Last in the melody sub-model is the global pooling layer. Up until now, all the data was represented using many vectors. This layer simplifies/averages it all into a single vector.

The sub-model that uses the previous chord feature consists of four layers as well: an input layer, an embedding layer, a flatten layer, and a dense hidden layer.

The first two layers are similar to those of the previous sub-model outside of the feature they are processing, so we will not discuss them in detail.

The flatten layer serves a similar purpose to the global pooling layer of the previous model in that it simplifies the data. However, instead of averaging multiple vectors into one, this layer is changing multidimensional data into 1D data.

The last dense hidden layer is meant to find patterns in the flattened data and determine correlations between previously played chords and chords to play. The activation function of choice is ReLU due to it being a hidden layer.

Both sub-models pass in their patterns they find within their feature and the label into the Concatenate layer, which is connected to yet another dense hidden layer (activation function also ReLU) meant to find patterns between the two features and the labels. Following that is an output layer that returns the optimal chord to play in the accompaniment. The output layer uses a softmax activation function. The model uses sparse categorical cross entropy due to inputs being previously encoded into integers.

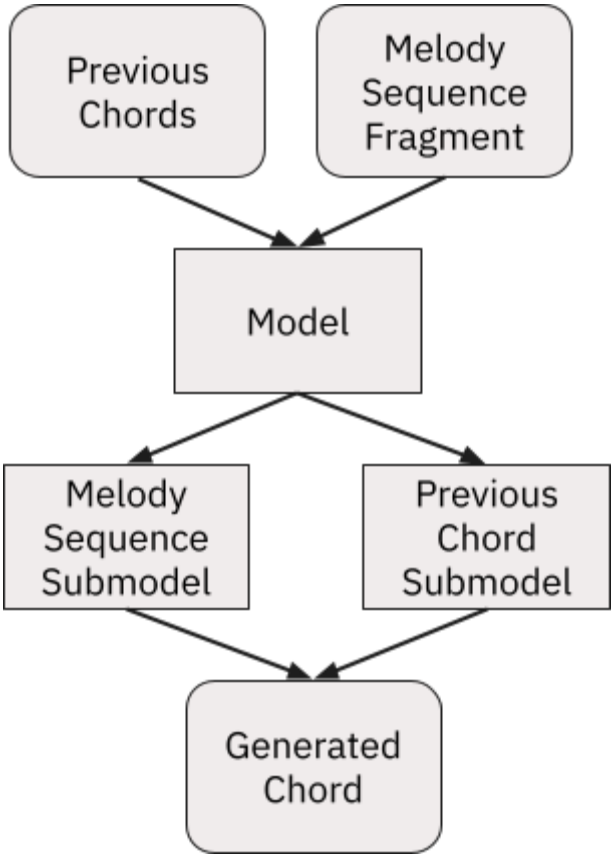

Figure 1: General Architecture of Models

Experiments/Results/Discussion:

Our experiments were done with our main metric being accuracy. We have several parameters that need to be adjusted. They are embed-dimension, embed layer output dimension, Transformer feed-forward node count, multi-head count, dense hidden layer counts, epoch number, and learning rate.

One of these quantities, the embed-dimension (for both sub-models), are fixed quantities not meant to be tested (128 pitches possible for melody model, number of possible chords for previous chord model). For some of the others, such as the transformer feed-forward node count and the dense hidden layer node counts, simply following industry standard is considered acceptable (i.e. 32 nodes for a small network, 64 nodes for a small-medium network).

For the remaining parameters (embed-layer output dimension, multi-head count, learning rate, and epoch number), some testing is required. In order to test these, only one variable can change at a time (all else must stay constant). Here are the tables for each of these parameters, where it is to be noted that the optimal value (bolded in table) for a parameter is used in the

following table testing another parameter. Refer to Table 1, Table 2, and Table 3 for embed-layer output dimension, multi-head count, and learning rate testing, respectively.

Embed-layer output dimension/Transformer-Input dimension	Validation Accuracy at Epoch 1
16	22.34%
32	**25.28%**
64	24.11%
128	19.25%

***Table 1*:** 128 is as high as is possible if we also regard runtime/computational complexity. 16, 32, and 64 are all very close to one another, but 32 is the best (just barely though).

Multi-head Count	Validation Accuracy at Epoch 1
2	25.28%
4	24.54%
6	**27.15%**
8	22.84%

***Table 2*:** Similar to before, any higher head counts would not result in a computationally simple model. There doesn't seem to be any trend like the last parameter (i.e. no clear relative max), but 6 did perform the best.

Learning Rate	Validation Accuracy at Epoch 1
0.001	**27.15%**
0.005	13.75%
0.01	4.19%

***Table 3*:** The algorithm seems to prefer a slow learning rate, hence 0.001 performed the best. As the learning rate increases, performance decreases.

As for testing the optimal number of epochs, we ran the model for 10 epochs with the optimal values from the tables above, monitoring the training and validation accuracies. We produced a graph for analysis of accuracies (refer to Figure 2 below). When the validation accuracy started decreasing but training accuracy kept on increasing, we realized that the model began to overfit. Thus, we should have our optimal epoch number right before then (i.e. when it peaks/at relative maximum). We found this epoch number to be 5.

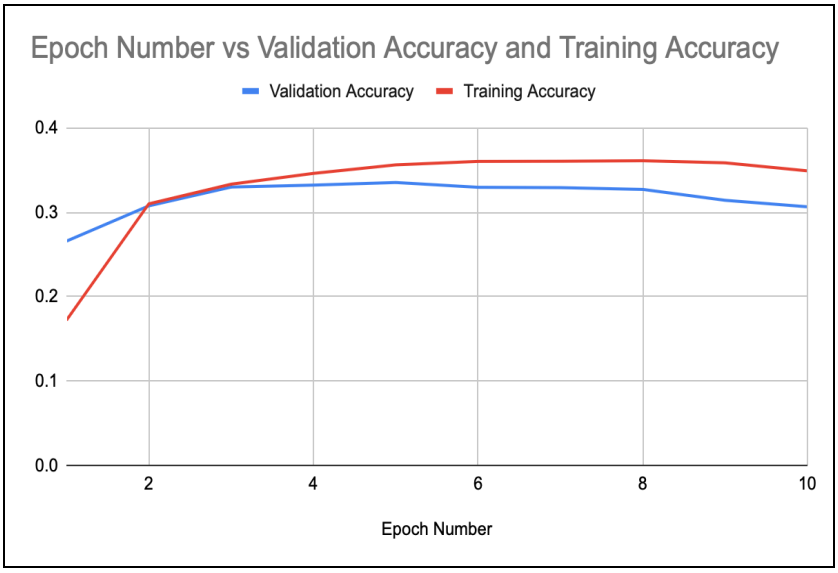

Figure 2: Effect of Epoch Number on Validation and Training Accuracies

One last check we did to test our model was a subjective-human evaluation of the generated accompaniment. It sounded somewhat pleasant, and thus passed this test.

Conclusion:

As a whole, our model succeeded in our goal of improving the quality of songs by generating chords for any melody. In terms of performance, the model worked best with lower learning rates, epochs, and other parameters as it was very prone to overfitting. Although our model did generate pleasant-sounding chords, the generated chords tended to repeat often and we had the restriction of chords being the same tempo.

Future Work:

For future improvements, we have to make our model focus more on chord progressions rather than mainly on harmonizing chords with the melody fragments, giving it more data like harmonic functions to do so. Along with this, we can branch out to make our model generate notes of different tempos and dynamics as well, something we already have the data to train. Our model can be improved, but as our model stands now, it is still a solid tool for musical accompaniment generation.

Contributions:

- Ryan Ghimire preprocessed the data, turned the generated chords into a listenable format, created prediction code, created midi player code, created the majority of the slideshow, and wrote some of the paper.
- Aniketh Luchmapurkar created the model, helped create some of the slideshow, and wrote the majority of the paper

References

Briot, J.-P., & Pachet, F. (2020). Deep learning for music generation: Challenges and directions. Neural Computing and Applications, 32(4), 981–993. https://doi.org/10.1007/s00521-018-3813-6

Conklin, D. (2003, April). Music generation from statistical models. In Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences (pp. 30–35). https://doi.org/10.1080/09298215.2016.1173708

Copet, J., Kreuk, F., Gat, I., Remez, T., Kant, D., Synnaeve, G., … & Défossez, A. (2024). Simple and controllable music generation. Advances in Neural Information Processing Systems, 36. https://doi.org/10.48550/arXiv.2306.05284

GarageBand. (n.d.). Apple Inc. Retrieved from https://www.apple.com/mac/garageband/

GeeksforGeeks. (n.d.). Transformer model from scratch using TensorFlow. Retrieved from https://www.geeksforgeeks.org/transformer-model-from-scratch-using-tensorflow/

GeeksforGeeks. (n.d.). Getting started with transformers. Retrieved from https://www.geeksforgeeks.org/getting-started-with-transformers/

GeeksforGeeks. (n.d.). What is embedding layer? Retrieved from https://www.geeksforgeeks.org/what-is-embedding-layer/

Hadjeres, G., Pachet, F., & Nielsen, F. (2017). DeepBach: A steerable model for Bach chorales generation. Proceedings of the 34th International Conference on Machine Learning, PMLR 70, 1362–1371. https://doi.org/10.48550/arXiv.1612.01010

Mao, H. H., Shin, T., & Cottrell, G. (2018, January). DeepJ: Style-specific music generation. In 2018 IEEE 12th International Conference on Semantic Computing (ICSC) (pp. 377–382). IEEE. https://doi.org/10.1109/ICSC.2018.00077

NumPy. (n.d.). NumPy library documentation. Retrieved from https://numpy.org

Pandas. (n.d.). Pandas library documentation. Retrieved from https://pandas.pydata.org

TensorFlow. (n.d.). TensorFlow library documentation. Retrieved from https://www.tensorflow.org

Pretty_MIDI. (n.d.). Pretty MIDI documentation. Retrieved from https://craffel.github.io/pretty-midi/

Scikit-learn. (n.d.). Scikit-learn library documentation. Retrieved from https://scikit-learn.org

OS (Python Standard Library). (n.d.). Python documentation. Retrieved from https://docs.python.org/3/library/os.html

Transformer Networks. (n.d.) Luchmapurkar, A. & Gagvani, M. (2024) Retrieved from https://sites.google.com/fcpsschools.net/transformer-networks/what-are-transformer-networks

Wang, Z., Chen, K., Jiang, J., Zhang, Y., Xu, M., Dai, S., … & Xia, G. (2020). Pop909: A pop-song dataset for music arrangement generation. arXiv preprint arXiv:2008.07142. https://doi.org/10.48550/arXiv.2008.07142