Generate Piano Instrumental Music by Using Deep Learning Models

Batuhan Ayvaz - Bachelor in Computer Science - Hacettepe University
Özge Kökyay - Bachelor in Computer Science - Hacettepe University
Batuhan Orhon - Bachelor in Computer Science - Hacettepe University

Abstract

In the last few years, deep learning technology that is the representative technology of AI, showed remarkable achievements in speech recognition, image recognition, natural language processing and music generation but generating music is still a challenging and interesting topic. In this paper, we present The Artificial Pianist that is a model trained for generating piano music and different methods like the models LSTM and GRU based RNN and self-attention based transformers that used throughout the project.

1. Introduction

Artificial intelligence and artificial creativity are closing to the human-level day by day and a growing area of application is the generation of creative artworks especially the generation of music for this project.

Similarly to the most of the art fields, music is also can not be analyzed qualitatively and quantitatively and even can't be described by simple words because of there is no clear indicator to assess and this is what makes music generation a challenging task.

We investigate if it is possible to build a model that automatically learn musical styles with using previous artworks and generate piano music that sounds similar to the real artworks of composers that respects the rules and classic patterns you expect to hear in The Artificial Pianist.

The Artificial Pianist is a deep learning project that is trained with MIDI formatted instrumental piano music from Maestro Dataset to generate different piano music. To train the model, we need to preprocess the MIDI files and feed our model with time series that consisting notes. After that, Artificial Pianist can generates its own series of notes.

The first problem that we came across while training the model the sequentiality of the input and the output the relation between a time point and the rest of the serial input, the difference between the length of inputs and outputs, etc.

2. Related Works

2.1. LSTM and GRU based RNN music generation projects

Realistic music generation has always remained a challenging problem as it may lack of structure or rationality when it comes to creating the melody and the harmony at the same time. In previos studies sequence models have been used for modeling music, from (Huang et al., 2018) Hidden Markov Models to RNNs and Long Short Term Memory networks, to bidirectional LSTMs because every neuron can learn from the previous input. (Hadjeres et al., 2017)

MelodyRNN from the Magenta project by Google Brain , tried to create compelling music after training the LSTM-based RNN on the NSynth data in wav form. LSTM networks have been shown to be effective because they can be trained to learn the structure and the characteristics of music.

In another paper (Keerti et al., 2020) study made for transfer the style of the music. This paper propose a deep learning based music generation method in order to produce old style music particularly jazz with rehashed melodic structures utilizing a Bi-directional Long Short Term Memory (Bi-LSTM) Neural Network with Attention.

In the first experiments of generating piano music, we choose Recurrent Neural Networks to train our models by using 'Generating Sequences With Recurrent Neural Networks (Graves, 2014) instead of Convolutional Neural Networks because CNNs only accept a fixed-sized vector as input and produce a fixed-sized vector while RNNs allow operating over sequences of vectors. The paper propose to train character-level language models based on multi-layer LSTMs to genarate text like it one character at one time from a given large chunk of text and this is why it seem a well-suited approach for the music generation task.

We have used Bi-Directional Layers with LSTM and GRU layers for solving short-term memory problems that we came across while using long sequences as input like 2D piano roll representation of MIDI formatted files that decomposed into a sequence. (Huang et al., 2018)

2.2. The Transformer Model

In recent years, the state of art approach in natural language processing and machine translation has changed drastically from using deep neural networks like CNN, RNN and LSTM to using the Transformer a sequence to sequence model based on self-attention mechanism. (Weng, 2018)

We decided that transformer model architecture will be a good match for solving this problem and creating the melody and the harmony at the same time because music generation tasks are similar to text generation tasks and music pieces also consisting of a series of specific notes like sentences are consisting words.

A transformer model handles variable-sized input using stacks of self-attention layers while RNNs and CNNs are usually process language by generating fixed length vectors thereby layer outputs can be calculated in parallel, instead of a series like an RNN and this boost the speed of training the model. Additionally all items can affect each other without using many convolution layers or RNN-steps and the model can learn the previous information as a whole.

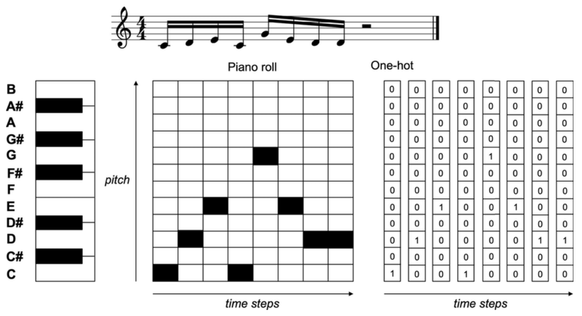
3. The Approach

3.1. Dataset

The Maestro Dataset have used in this project. The Maestro Dataset contains 200 hours of virtuosic piano performances captured with fine alignment between note labels and audio waveforms. The repertoire is mostly classical, including composers from the 17th to early 20th century. The dataset has several sections by year of performance and we prefer the 2013 year of performances section because it includes similar style piano music.

3.2. Preprocess

Preprocess is a phase that the dataset has been prepared for the training phase. There are some essential Python libraries in preprocess phase, one of them is PrettyMidi. We have extracted the required features from midi files by using the PrettyMidi library and we have trained our model using these features. First of all, we have read all midi files in our dataset. The midi files have starting time, ending time, played notes, and their velocity data for each instrument in order. Our dataset only includes piano music, so we have imported only the piano section in midi files. We have got rid of redundant data at this step and it is significant for speeding our training phase. After clearing the data, we get a (notes, time) dimension NumPy array.(Briot, 2020)

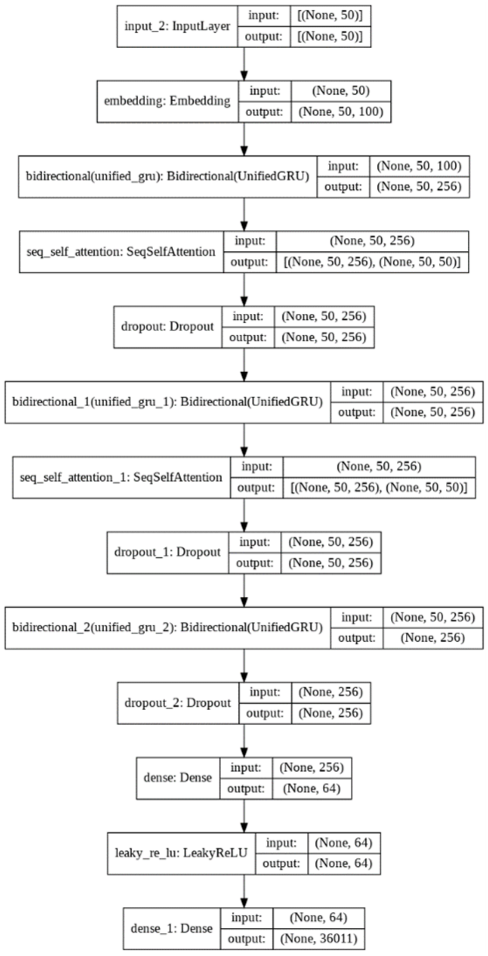


We have paid special attention to the using NumPy array because we have done lots of matrix operations in our code and NumPy arrays perform much better performance comparing to vanilla arrays. The (notes, time) dimension NumPy array have had still redundant data because only a few notes played in a specific time sequence. Therfore, we have converted the NumPy array to the dictionary. Dictionary keys start from a beginning time and their values are the played notes array in that time zone. After initializing the dictionary, we have done converting operation on keys that transforming the played notes array to string. After that operation, we have transformed the dictionary into a vector that included only played notes in sequential order. Finally, we have done token operation to that vector. The token operation has transformed each note to different integer values. After the Token operation, our dataset ready for the training phase.

3.3. Neural Networks Architecture

3.3.1. Overview

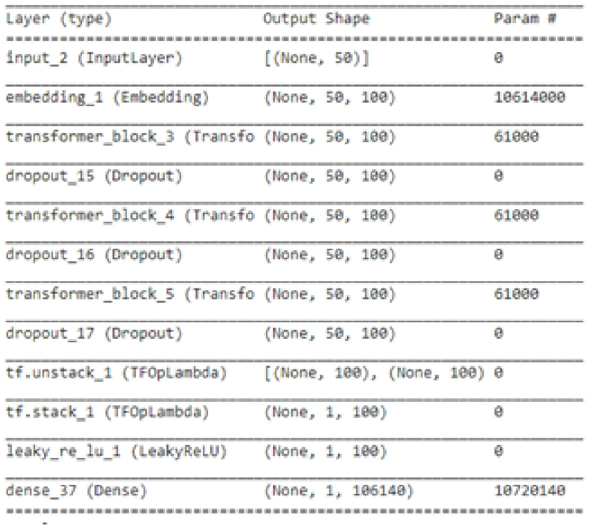
In the base model, bi-directional GRUs are used. Using GRU layers is reasonable because RNN structures are good to work on sequential data and we feed our network architecture with sequential data. Recurrent Neural Networks use hidden states at each step whose values are calculated using the previous step's output. GRU is a type of RNN that uses some gates to solve the vanishing gradient problem of a standard RNN. Bi-directional GRU calculates future outputs and uses outputs of both future and previous steps to train the model. Up to that point, we mentioned the good sides of bi-directional GRU. However, there are some weaknesses of that structures such as training the data sequentially. As mentioned, we feed our neural network architecture with a sequence of 50 notes and GRU calculates each token's output serially(more calculations for a bi-directional layer). That causes very low training speed. Also, those hidden states make backpropagation harder. Nevertheless, even after 5-10 hours of training the results are "not good". So, we decided to change some hyperparameters such as epoch, batch size, initializer note and changed our dataset to one with more similar songs to improve the results but we couldn't provide major changes. At that point, we knew we had to change the model to get better results.


input_2: InputLayer | input: [(None, 50)] | output: [(None, 50)]
embedding: Embedding | input: (None, 50) | output: (None, 50, 100)
bidirectional(unified_gru): Bidirectional(UnifiedGRU) | input: (None, 50, 100) | output: (None, 50, 256)
seq_self_attention: SeqSelfAttention | input: (None, 50, 256) | output: [(None, 50, 256), (None, 50, 50)]
dropout: Dropout | input: (None, 50, 256) | output: (None, 50, 256)
bidirectional_1(unified_gru_1): Bidirectional(UnifiedGRU) | input: (None, 50, 256) | output: (None, 50, 256)
seq_self_attention_1: SeqSelfAttention | input: (None, 50, 256) | output: [(None, 50, 256), (None, 50, 50)]
dropout_1: Dropout | input: (None, 50, 256) | output: (None, 50, 256)
bidirectional_2(unified_gru_2): Bidirectional(UnifiedGRU) | input: (None, 50, 256) | output: (None, 256)
dropout_2: Dropout | input: (None, 256) | output: (None, 256)
dense: Dense | input: (None, 256) | output: (None, 64)
leaky_re_lu: LeakyReLU | input: (None, 64) | output: (None, 64)
dense_1: Dense | input: (None, 64) | output: (None, 36011)

First, we tried to use a different attention mechanism, but multi-head attention was the best we could get. While searching about the attention mechanisms, we encountered the research “Attention Is All You Need”. In that work, transformer blocks were described. (Weng, 2018)

Transformers are widely used in natural language processing problems and most of the time they give better performance than RNNs. Similar to RNN, transformers have an encoder-decoder approach which is slightly different. In transformer blocks, there isn’t any RNN, but only attention with normalization and feed-forward layers. So, the absence of RNN makes the transformer approach much faster since attention layers don’t work sequentially and calculate the output of given 50 input values parallelly. In addition to those, a model with transformers gives better results even after a train time 10 times less than GRU based model. After we added the transformers, the improvement of the results was obvious. As the last step, we added dropout layers between the transformers and that provided some improvement too.

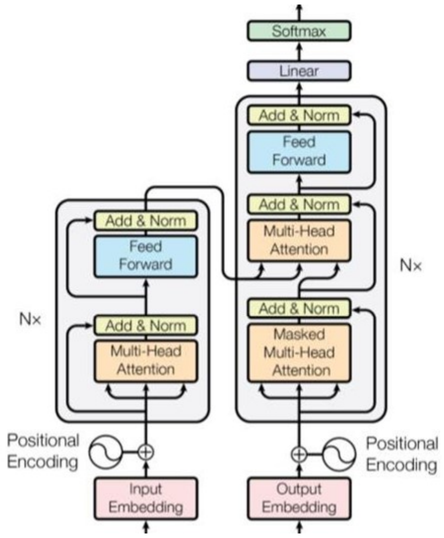
3.3.2. Model Details



```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 50)]              0
_________________________________________________________________
embedding_1 (Embedding)      (None, 50, 100)           10614000
_________________________________________________________________
transformer_block_3 (Transfo (None, 50, 100)           61000
_________________________________________________________________
dropout_15 (Dropout)         (None, 50, 100)           0
_________________________________________________________________
transformer_block_4 (Transfo (None, 50, 100)           61000
_________________________________________________________________
dropout_16 (Dropout)         (None, 50, 100)           0
_________________________________________________________________
transformer_block_5 (Transfo (None, 50, 100)           61000
_________________________________________________________________
dropout_17 (Dropout)         (None, 50, 100)           0
_________________________________________________________________
tf.unstack_1 (TFOpLambda)    [(None, 100), (None, 100) 0
_________________________________________________________________
tf.stack_1 (TFOpLambda)      (None, 1, 100)            0
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 1, 100)            0
_________________________________________________________________
dense_37 (Dense)             (None, 1, 106140)         10720140
=================================================================
```

Embedding Layer In NLP problems, most of the time embedding layers are used as the first layer to compress the input feature space into a smaller one. Using such a layer we represent the tokens using a dense vector representation.

Transformer This is the main layer that our model learns about the sequences whose instructions mentioned in the overview. Transformers basically uses a structure similar to the encoder-decoder approach in RNN.



The image above is a representation of transformer model architecture. The transformer consist an encoding component and a decoding component. The encoding component is a stack of encoders and the decoding component is a stack of decoders in same numbers with the encoding component. Encoders consist a self-attention layer and a feed-forward neural network. Decoders consist an additional layer between self-attention layer and feed-forward neural network that provides focusing the relevant parts of the input. Self-attention is described as mapping a query and a set of key-value pairs to an output as a weighted sum of the values where the weight of each value is computed by using multi-head attention layers functions with the query and its key. Self-attention provides a comparasion between the content of an input to all other inputs in the sequence and put the relationships into the embedding. Multi-head attention is a different form of self-attention layer that consisting a mechanism called multi-headed attention and each position in the encoder and decoder can attend to all positions in the previous layers, in this way performance of the attention layer is boosted and the model's ability to focus on different positions is expanded.

Dropout There are different neural network model configurations to solve the overfitting problem. However, most of these solutions come with an extra computational expense. Dropout model comes with the solutions for both these problems. It does randomly drop out nodes during training. This solution is computationally cheaper than the other solutions. Therefore, the dropout model is an effective regularization model for the overfitting problem.

Stack-Unstack As an input we take 50 sequential notes of a part of the song, and we want to predict 51th note. Using stack operations, we seperate the last element of the given sequence.

Dense Layer — LeakyRelu Layer LeakyRelu is a neural network activation function. It will output the input directly if it is positive, otherwise, it will output zero. Our model performs well with Relu layer and it is easy to train a model with it. Then we take the output of LeakrRelu and give it as input of a dense layer. Dense Layer is a neural network layer just like the other neural network layers and it has the same updating formula. However, Dense Layer is a non-linear layer and its output passed to the activation function. Our dense layer's output length is equal to the number of unique notes, and its activation function is softmax. After all these operations we choose one of the most probable notes randomly in generation part.

3.4. Generation

The generation phase is generating midi music files using the trained model. Our project can generate midi music files recurrently by using the same trained model. This section mentions how to create random pieces of music in the project.

First of all, initial notes must be determine. Our initial notes include forty-nine empty notes following by one initial note. The one initial note is hyperparameter and specifies by considering our dataset. Then, we predict the weights of each candidate note by using the trained model that parameters are initial notes. After that, we decide the eventual note randomly by considering each candidate note weight using NumPy random python library. In the second step, we shift the initial notes to the right by one. New initial notes are forty-eight empty notes following by two notes. Then, we perform the identical prediction and shifting operation to the end of the music. Before the creation of the midi file, we drop the initial fifty notes.

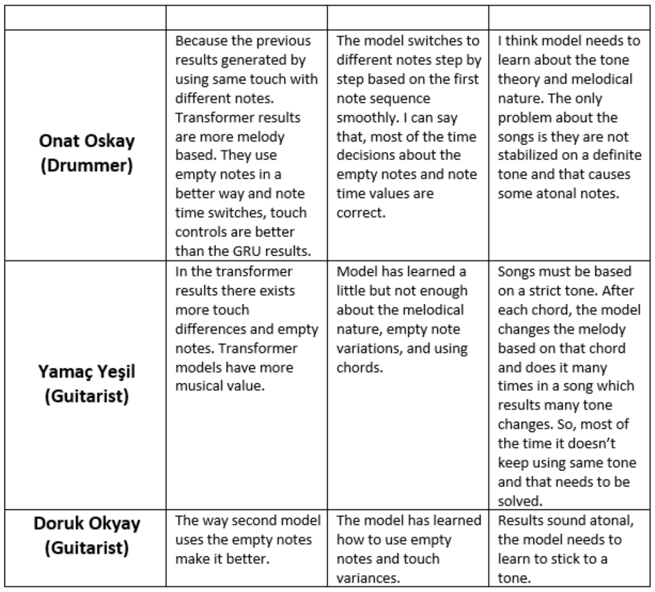
4. Experimental Results

4.1. Subjective Evaluation

Evaluation of musical results is a challenging problem, and it is not easy to implement some functions for evaluation of songs. In the training phase, we know the real values of the predicted outputs and by using these values we can calculate the loss . However, in the evaluation phase we need to find another way to calculate the performance of randomly generated songs. Throughout the progress we got some help from our musician friends. Evaluating the results one by one is time consuming but it is more accurate.

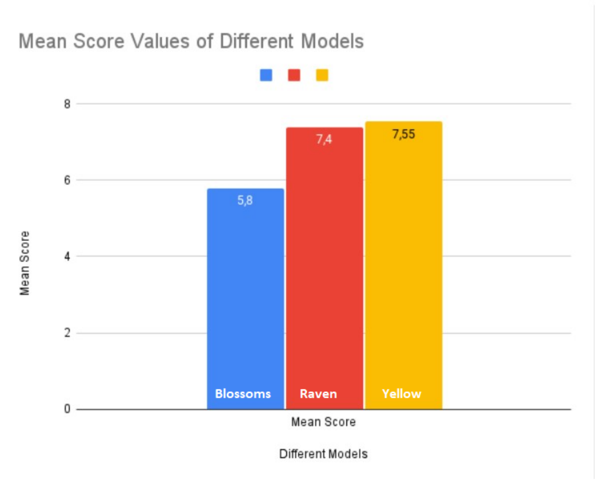
4.2. Some Subjective Evaluations

We asked three questions to three different musicians about the results, and got these answers:



Onat Oskay (Drummer)	Because the previous results generated by using same touch with different notes. Transformer results are more melody based. They use empty notes in a better way and note time switches, touch controls are better than the GRU results.	The model switches to different notes step by step based on the first note sequence smoothly. I can say that, most of the time decisions about the empty notes and note time values are correct.	I think model needs to learn about the tone theory and melodical nature. The only problem about the songs is they are not stabilized on a definite tone and that causes some atonal notes.
Yamaç Yeşil (Guitarist)	In the transformer results there exists more touch differences and empty notes. Transformer models have more musical value.	Model has learned a little but not enough about the melodical nature, empty note variations, and using chords.	Songs must be based on a strict tone. After each chord, the model changes the melody based on that chord and does it many times in a song which results many tone changes. So, most of the time it doesn't keep using same tone and that needs to be solved.
Doruk Okyay (Guitarist)	The way second model uses the empty notes make it better.	The model has learned how to use empty notes and touch variances.	Results sound atonal, the model needs to learn to stick to a tone.

4.3. User Study



We have made a user study for the 3 music pieces generated by different model architectures and collect points between 1 to 10 from 50 listeners and calculated mean opinion score for each piece as subjective evaluation.

Blossoms is one of the first music pieces that has generated by using LSTM layers and self-attention mechanism. Blossoms got 5.8 points from listeners in average and this is the lowest score. We received comments about Blossoms like it is atonal and most of the notes are just random, it has less harmony and not euphonic, some parts of the music are repeating often.

After the generation of Blossoms (and the other similar music pieces), we have decided that we need further improvements on the architecture of model and we have changed the model using Transformers. Results of this model were explicitly better than the first results.Raven also has been generated by using this model, and got 7.4 points and received nice comments from listeners like it is more harmonic, soft and less atonal.

Lastly, we have tried using dropout layers in the model. Yellow has been generated by using this model. Yellow got 7.55 points from listeners and nice comments too. There is not a big difference between the music pieces generated by the second and third model but we can say that the third model is slightly more euphonic.

5. Conclusions

In the evaluation results, almost all of the form attendants and our musician friends thought that transformers performs better than bi-directional GRU in our project. Transformer performed faster and provided better results than the bi-directional GRU. However, our model still has lots of thinks to learn such as stricting to a regular tone, and melody. My personal idea is that, our preprocess and generation functions may be improved and we may feed our network in a better way to reach the needs mentioned in evaluation part. Additionally, we have only trained 100-200 songs in our approach and that count is quite low. In the future, we may change AI pianist to AI orchestra, or we may train our networks to mimic specific musician's style. However, the first thing we would like to improve is our learning mechanism and generation function. After our musician friend said we could train tone theory, we thought that we might implement our generation function in such a way. To conclude, we learned a lot about the RNN and transformers and we also learned advantages of transformers over RNN.

References

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer, 2018.

Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation, 2017.

Gullapalli Keerti, A N Vaishnavi, Prerana Mukherjee, A Sree Vidya, Gattineni Sai Sreenithya, and Deeksha Nayab. Attentional networks for music generation, 2020.

Alex Graves. Generating sequences with recurrent neural networks, 2014.

Lilian Weng. Attention? attention! *lilianweng.github.io/lil-log*, 2018. URL `http://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html`.

Jean-Pierre Briot. From artificial neural networks to deep learning for music generation – history, concepts and trends, 2020.