Music Generation with Deep Learning

Yusuf Cem Kasap
Control and Automation Engineering Department
Istanbul Technical University
Istanbul, Türkiye
kasap19@itu.edu.tr - 040190764

Gülsüm Beyza Altundal
Electronics and Communication Engineering Department
Istanbul Technical University
Istanbul, Türkiye
altundal20@itu.edu.tr - 040200211

***Abstract*— With the advancement of artificial intelligence, deep learning techniques for generating has gained popularity in the recent years. In this project, we aim to propose a generative model using Recurrent Neural Networks (RNN), in other words a Long Short-Term Memory (LSTM) network. A melody can be represented using the Musical Instrument Digital Interface (MIDI) format to describe a collection of musical notes, which come together to form a musical piece. We aim to extract of onset time, duration, MIDI pitch and velocity features of notes of a musical piece from a dataset of 10,855 unique solo piano works.**

Keywords—Music, RNN, LSTM, Neural Networks, MIDI

I. INTRODUCTION

Music is the art of arranging tones or sounds in a sequence. It involves relationship of notes with continuity. Essentially, a musical note refers to any sound produced using musical instruments or the human voice.

AI models can be trained using either monodic sound, with a single melodic line, or polyphonic sound, involving multiple tones. Musical notes are defined by octaves or pitch intervals. Music details are essential for training the model and model's complexity and output depends upon the nature of the input. Music is generated by playing notes of different frequencies and the linkages between notes are preserved [1].

Deep learning, a branch of artificial intelligence and machine learning, is a fast-evolving field that has found widespread application in areas such as image analysis, voice recognition and computer vision. The popularity of deep learning and artificial intelligence are increasing due to factors such as availability of datasets, increase in computational power and emerging of new technologies. The formulation of music sequence generation as deep learning problem has been a widely explored topic in recent years.

A deep neural network consists three types of layers. Input layers, output layers and hidden layers. Input and output layers have the input data and the output data and the hidden layers are responsible for the mathematical calculations on the input data [2].

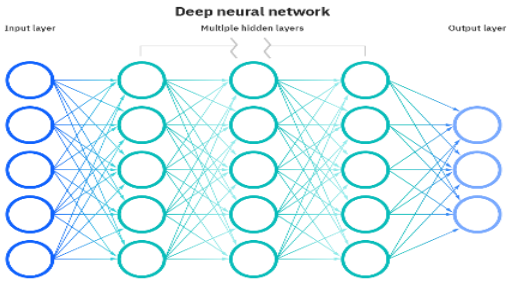

Fig. 1. Deep neural network structure.ar

This project uses Musical Instrument Digital Interface (MIDI) to maintain the symbolic domain generation of music. Symbolic domain generation of music was done using Recurrent neural network (RNN) before. But Hochreiter et. al stated that standard RNNs are not sufficient enough to store information from previous time steps for very long due to vanishing or exploding gradient problems [3]. However, as a solution to such problems, Long Short-Term Memory (LSTM) [4] Networks, a type of RNN, have emerged to store information over a long period.

Thus, in this project we aim to propose a solution using LSTM networks which can be used to generate music and melodies without human interaction. The goal is to develop a model which can learn from a set of data which is extracted from the MIDI files, which consists onset and duration times, velocity and MIDI pitch information of notes.

In the following sections, we have provided a literature review in Section II. Section III introduces the problem description. A solution strategy and explanation of the proposed model is provided in Section IV. Test results are discussed in Section V, followed by the conclusion in section VI.

II. BACKGROUND

A. Literature Review

Over the years, music generation has undergone remarkable advancements, with researchers exploring diverse methods to create music across various domains. One of the earliest works for music generation was using Markov chains to generate music [5], [6], which was a mathematical system that undergoes transitions between different states, following specific probabilistic rules.

With the advancements in AI, new models and methods were proposed for music generation. Chen et. al [7] and Todd et. al [8] attempted to generate music using RNNs but they were not efficient in caring long-term dependencies.

Dua et. al [9] tried to enhance the accuracy of sheet music generation achieved by earlier models by improving their source separation and chord estimation components. Their method used deep learning techniques, specifically employing RNNs with Gated Recurrent Units (GRUs), which are a type of RNN architecture designed to capture dependencies in sequential data, and LSTM. The source separation component was developed by multiple layers of GRU cells, while the chord estimation component was build using LSTM. Their work enabled isolation of a large number of sources which greatly boosted the precision of chord estimation

Oord et al. [10] developed WaveNet, a text-to-speech model utilizing Convolutional Neural Networks (CNNs) trained directly on raw audio data. They demonstrated that their model is also capable of generating music. Engel et al. [11] tried to improve the structure of Wavenet with autoencoders to remove the need for external conditioning.

Generative Adversial Nets (GANs) [12] have also been used for music generation. More than one multiple CNN based GAN models have proposed [13],[14], and Mogren [15] proposed a Continuous RNN-GAN (C-RNN-GAN) model specifically for continuous sequential data which has a better overall performance.

LSTM was firstly used by Eck et. al [16], followed by many different models with changes and improvements [1],[9],[17],[18],[19]. Johnson [20] proposed, a model which is inspired by the convolution, Bi-axial LSTM (BALSTM), which has two variants of RNNs that divides the melody into pitch sequence and duration sequence.

Our model is inspired by "LSTM Based Music Generation" [1]. In their works, they used a model inferred from "Generation of Music using LSTM" [17], which is model also uses BALSTM that uses two LSTMs. One to predict time and other to predict note sequence. Here we are using one LSTM with different datasets and features to predict both. We presented the model of the original paper, our changes and approach in this paper.

B. MIDI Representation

Musical Instrument Digital Interface (MIDI) is an advanced technology and communication protocol that exchanges musical data between digital instruments, software, and gadgets. MIDI makes it possible to extract significant amount of information when studying music. The symbolic representation of music in the MIDI format is discrete. MIDI files provide two major symbolic elements: notes and chords [18].

A note is the representation of a single sound and a chord is combination of sounds played together. A note has two main features which are pitch and octave. Pitch is the frequency of a sound, measured in hertz (Hz), that determines how high or low it sounds. In musical notation, pitches are represented by letters: A, B, C, D, E, F, and G, where A corresponds to the highest and G to the lowest frequency in a given scale.

For instruments like the piano, pitch is identified by a letter and a number. The number indicates the octave, and it shows the range of the note on the keyboard. An octave is the pitch range or distance between one note and the next note with the same letter name, either higher or lower. Higher octaves have higher pitches.

A chord is consisting three or more notes played together. This combination creates a harmonious effect. And this adds depth and richness to the music. Chords are important elements in creating the emotional tone and harmony of a piece.

This structure helps MIDI preserve the timing and harmony of music. It gives a simple format for music analysis and creation.

C. Recurrren Neural Networks (RNNs)

RNNs are a type of model that can generate sequences in different areas like music, text and motion capture. They are trained to generate sequences by looking at real data one step at a time and predicting what comes next [21]. This is done by taking the output of each hidden layer, and feeding it back to itself as an additional output. Each node in the hidden layer gets both inputs from the previous layer and the outputs of the current layer from the previous time step.

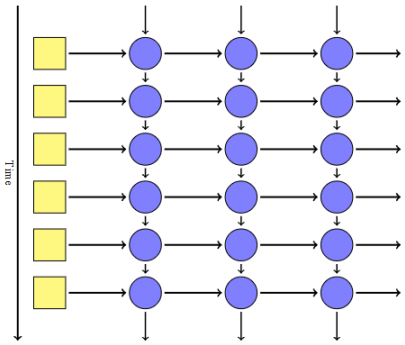

Fig. 2. RNN in unfolded form [22].

In this representation each horizontal line of layers is the running at a single time step. This enables the network to have a minimal, simple version of memory. This allows for inputs and outputs of variable lengths, we can provide inputs one at a time, and let the network combine the using the state carried over from each previous time step.

A typical neural network has the form:

$$X_t = Y_t \tag{1}$$

In case of RNN, this is modified to:

$$[X_t, X_{t-1}] = Y_t \tag{2}$$

Where the current input is X_t, input from the past is X_{t-1} are used to predict the output Y_t. The equation of the prediction is:

$$h_t = f(w_h h_{t-1} + w_X X_t + b_h) \tag{3}$$

Here, h_t is the new hidden state, h_{t-1} is the hidden state, w_h, w_x are weights, and b_h is the bias.

One problem with this is that the memory is very short-term due to vanishing or exploding gradient problem. To solve this, more advanced RNN architectures were

developed such LSTM networks which are better equipped to manage long-term dependencies [23].

D. Long Short-Term Memory (LSTM)

LSTM neural network is a special type of RNN capable of analyzing long time series data and learning persistent dependencies. For applications like music generation, this makes it a perfect solution as they demand the neural network to retain information from several previous steps to present [21].

LSTM processes information by using the previous hidden state and the current input as it's inputs. The first component of a LSTM cell is the forget gate (f_t), which determines whether specific information should be discarded or not. Next, the input gate identifies the relevant information to add from the current step. To compute the updated cell state (C_t), the previous cell state (C_{t-1}) is first multiplied by the forget vector. Then, the output from the input gate (i_t) multiplied by the candidate value ($\widetilde{C_t}$). Finally, the output gate (o_t) determines the next hidden state (h_t). The activation functions employed in LSTM cells are sigmoid (σ) and $tanh$ [24].

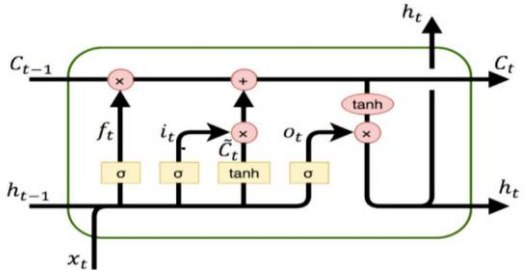

Fig. 3. LSTM cell

Equations for LSTM cell is as follows.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (4)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (5)$$

$$\widetilde{C_t} = tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (6)$$

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \quad (7)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_0 \quad (8)$$

$$h_t = o_t * \tanh(C_t) \quad (9)$$

E. Bi-Axial LSTM (BALSTM)

A network architecture proposed by Daniel D. Johnson [22] The BALSTM architecture creates polyphonic music by treating each note at a given time step as a probability, conditioned on all prior time steps and the notes already generated within the current time step. The structure is defined as:

$$P\left(X_{t,n} \middle| X_{t,n-1}, X_{t,n-2}, \dots, X_{t-1,N}, X_{t-1,N-1}, \dots, X_{1,2}, X_{1,1}\right) \quad (10)$$

The network architecture primarily consists of a time-axis and a note-axis. After the note data is processed and prepared as the input layer, it is connected to the time-axis layers. The time-axis, which includes two layers of stacked LSTMs, processes note across all time steps for each pitch recurrently. As a result, the output retains the temporal structure of the music, serving as high-level features that carry time-based information to the note-axis. The note-axis, also composed of two layers of stacked LSTMs, trains note across all pitches for each time step recurrently, capturing the vertical pitch relationships among notes. This is useful for identifying chord or counterpoint features in the music [19].

F. Bidirectional LSTM (BiLSTM)

Bidirectional Long Short-Term Memory (BiLSTM)[28] networks are an extension of traditional LSTM architectures designed to process sequential data in both forward and backward directions. This bidirectional approach enables BiLSTMs to capture context from both past and future states simultaneously, enhancing their performance in various sequence modeling tasks.

In a BiLSTM, two separate LSTM layers are employed: one processes the input sequence from start to end (forward direction), and the other processes it from end to start (backward direction). The outputs from both layers are then combined, typically by concatenation, to form the final output. This structure allows the model to have a more comprehensive understanding of the input data, as it considers information from both preceding and succeeding elements in the sequence.

III. Methodology

A. Problem Description

Generative music AI models have a great potential for transforming the way music is generated, but they encounter some challenges to generate outputs that meet both artistic and practical requirements.

One of the first issues encountered while designing a generative music AI model is selecting an appropriate data representation for the music. Possible formats include signal, modified signal, MIDI, and text. The final destination of music content is an important topic. For instance, the format destination could be a human user, and in this case the output needs to be in a human-readable form. Such as an audio file (e.g., MP3 or WAV) that can be played and listened to. However, in our case the target is a machine, therefore, the output should be in a machine-readable format, such as a MIDI file, which encodes musical data in a structured and accessible form for computational processing.

Another challenge is processing the MIDI data into a format that is compatible with the model for training and generation. While MIDI is a rich symbolic representation of music, its raw format is not directly compatible with machine learning models. In this project, which is coded with MATLAB, some functions from MIDI toolbox will be utilized, to process this data.

Supervision during output generation is also an important issue to consider. At one extreme, there is complete autonomy and automation, requiring no human oversight. The approach might be more interactive, allowing for early halting to monitor the music creation process. This paper's neural network technique is not designed to be interactive. The MIDI file format is intended

for this dimension as it provides a machine-readable output without human interaction. Autonomy is a promising feature for musicians who may need to pause content creation.

B. Dataset

While training our model, GiantMIDI-Piano [25] will be used. The dataset contains 38,700,838 transcribed notes and 10,855 unique solo piano works composed by 2,786 composers as MIDI files. Approximately 30 of these files were broken but this wouldn't be a problem for our study.

MIDI files do not contain actual sounds but consist of a series of commands such as "note on," "note off," and "change tempo." These MIDI messages are interpreted by hardware or software MIDI instruments, which then generate the corresponding sounds. The messages can be transmitted on different channels, each assigned to a specific instrument. For instance, channel 0 might represent a piano, while channel 1 could correspond to a guitar. Since MIDI files only store the musical score (similar to sheet music) rather than audio, they typically require significantly less storage space compared to other audio formats like WAV or MP3. To extract the information MIDI Toolbox [26] will be used in MATLAB. With the functions in toolbox, we can extract the onset time, duration, pitch, and velocity information of MIDI files as a matrix which is called a "note matrix"

```
nmat =
0         0.9000    1.0000   64.0000   82.0000         0    0.5510
1.0000    0.9000    1.0000   71.0000   89.0000    0.6122    0.5510
2.0000    0.4500    1.0000   71.0000   82.0000    1.2245    0.2755
2.5000    0.4500    1.0000   69.0000   70.0000    1.5306    0.2755
3.0000    0.4528    1.0000   67.0000   72.0000    1.8367    0.2772
3.5000    0.4528    1.0000   66.0000   72.0000    2.1429    0.2772
4.0000    0.9000    1.0000   64.0000   70.0000    2.4490    0.5510
5.0000    0.9000    1.0000   66.0000   79.0000    3.0612    0.5510
6.0000    0.9000    1.0000   67.0000   85.0000    3.6735    0.5510
7.0000    1.7500    1.0000   66.0000   72.0000    4.2857    1.0714
```

Fig. 4. An example note matrix

Columns in the note matrix represents "Onset (Beats)", "Duration (Beats)", "MIDI channel", "MIDI pitch", "Velocity", "Onset (Seconds)", "Duration (Seconds)", from left to right. All the MIDI files in the dataset will be represented with note matrices and will be preprocessed to suit our model. The songs on the dataset have different durations so it can be useful normalize them to have same duration.

C. Solution Strategy

As stated in Section II, half our models are using two LSTM or BiLSTM layers, other half is one LSTM or BiLSTM layers to keep the model simple as it can.

Firstly, the data will be preprocessed to suit the model structure and will be divided into training, testing datasets. Then the proposed model will be implemented using MATLAB and MIDI Toolbox. After training and evaluating our model, it will be compared with other models with the evaluation criterions mentioned in [27]. Lastly, the output will be converted to MIDI files using MIDI Toolbox.

IV. Implementation

A. Dataset Preprocessing

Firstly, the midi files in the dataset have been added to MATLAB environment using the MIDI toolbox. It was seen that the dataset is too large to use it with the computational power we have so we had to trim it down. First, the songs that has 500 or less timesteps have been extracted. Another problem was the songs in the datasets have too many outliers with different starting and ending times and they are also extracted from the dataset. Lastly, we cropped the MIDI files to only use the first 500 timesteps of all of them. In the end, we had 945 songs that has similar durations and starting beats.

Before normalizing the dataset, the songs are quantized according to 1/32 of the beats, which divide each beat into 8 equal parts and rounds the beat data of the songs. Also, all of the songs in the dataset are tuned to C chord for better estimation.

We normalized the pitch, onset and duration values according to their maximum values on the entire dataset. Thus, mapping them between 0 and 1. Onset, duration and pitch values are on 1, 2 and 4th column respectively.

$$\max(pitch) = 103 \tag{11}$$

$$\max(onset) = 100 \tag{12}$$

$$\max(duration) = 12 \tag{13}$$

If we inspect the data representation, we can see that to generate a sequence we only need these values, so we extracted 3 important features from the songs and created a new dataset with feature dimension 3. Features are onset, duration and pitch. MIDI channel is set to 0 for all songs as they are piano songs. Velocity can be set manually as it has no effect on the sequence. Onsets and durations on seconds can be calculated using the tempo of the seed sequence. Almost all of the songs in the dataset have 120 BPM tempo which means there are 2 beats in a second. So, if we divide the onsets and durations by 2, we get the corresponding time in seconds. An example note matrix with extracted and normalized features are given below.

```
0.0163    0.0104    0.3689
0.0163    0.0104    0.4854
0.0238    0.0104    0.5049
0.0238    0.0104    0.3883
0.0262    0.0104    0.5146
0.0262    0.0104    0.3981
0.0350    0.0104    0.5340
0.0362    0.0104    0.6505
0.0425    0.0104    0.6505
0.0425    0.0104    0.5340
```

Fig. 5. An example note representation with extracted features.

In this work, two generation strategies are followed. A sequence-to-sequence model with a fixed sequence length which outputs another sequence with the same length and a sequence-to-one model which generates one time step at each iteration. On the trained models, sequence lengths are set as 50. Training, testing and validation is split as 90-7.5-2.5 respectively.

B. Network Design

Total of 8 models trained for both sequence-to-sequence and sequence-to-one approaches. In general, 4 different model structures are trained. First one using a single layer of LSTM with 512 cells. Second one using a

BiLSTM also with 512 cells. Other 2 models are the same with one more additional LSTM and BiLSTM layers.

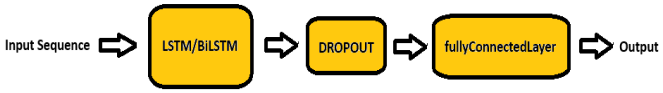

Fig. 6. Network structure with one RNN layer

Fig. 7. Network structure with two RNN layers.

A sequential input layer is added to the beginning with feature dimension of 3. A dropout layer with factor 0.3 is added after each RNN layer to generalize the training portion done by the LSTM layer. During training, it randomly "drops out" a fraction of the neuron's outputs at each forward pass to prevent overfitting. At the end a connected layer with feature dimension 3 is added. While training the network loss function is selected as **L2 Loss**. Initial learning rate is selected as 0.0001. Structures of the networks are given below.

Only difference between the trained networks labeled with 1 and 2 are the datasets used for training. First label is for sequence-to-sequence and second one is for sequence-to-one.

Network structures are given below.

1	'sequenceinput'	Sequence Input	Sequence input with 3 dimensions
2	'biLSTM'	BiLSTM	BiLSTM with 512 hidden units
3	'dropout'	Dropout	30% dropout
4	'fc'	Fully Connected	3 fully connected layer

Fig. 8. BiNet50 structure

1	'sequenceinput'	Sequence Input	Sequence input with 3 dimensions
2	'lstm'	LSTM	LSTM with 512 hidden units
3	'dropout'	Dropout	30% dropout
4	'fc'	Fully Connected	3 fully connected layer

Fig. 9. Net50 structure

1	'sequenceinput'	Sequence Input	Sequence input with 3 dimensions
2	'biLSTM_1'	BiLSTM	BiLSTM with 512 hidden units
3	'dropout_1'	Dropout	30% dropout
4	'biLSTM_2'	BiLSTM	BiLSTM with 512 hidden units
5	'dropout_2'	Dropout	30% dropout
6	'fc'	Fully Connected	3 fully connected layer

Fig. 10. duBiNet50 structure

1	'sequenceinput'	Sequence Input	Sequence input with 3 dimensions
2	'lstm_1'	LSTM	LSTM with 512 hidden units
3	'dropout_1'	Dropout	30% dropout
4	'lstm_2'	LSTM	LSTM with 512 hidden units
5	'dropout_2'	Dropout	30% dropout
6	'fc'	Fully Connected	3 fully connected layer

Fig. 11. duNet50 structure

V. RESULTS

A. *Progress Reports*

Models are trained using **"adam"** optimizer with 10 epochs. Mini batch sizes are chosen as 200.Figures below shows the training progresses of each model.

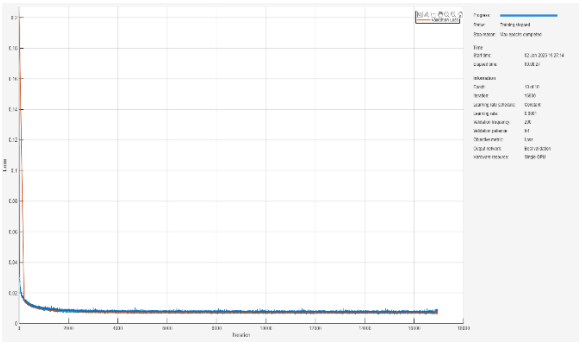

Fig 12. Training progress of Net50_1

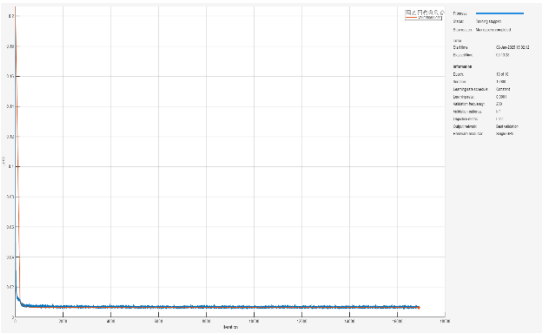

Fig. 13 Training progress of BiNet50_1

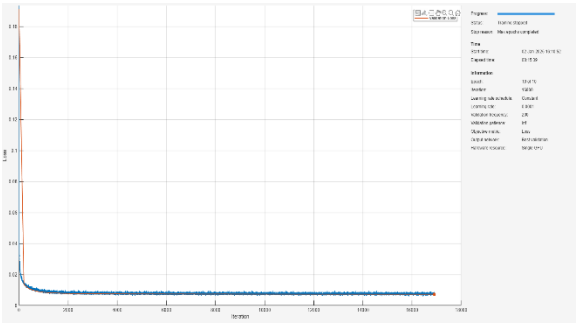

Fig. 14. Training progress of duNet50_1

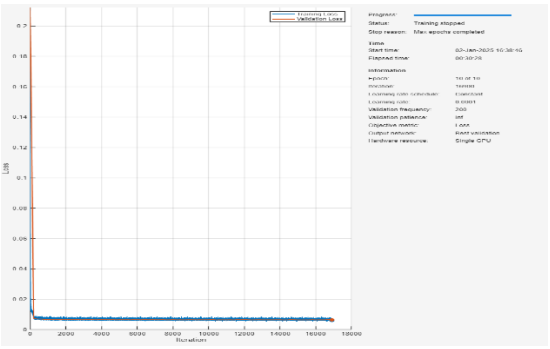

Fig. 15. Training progress of duBiNet_1

From these figures it can be seen that the model with sequence-to-sequence approach quickly converge to loss value of 0.006-0.007. Label 1 after the underscores refers to models with sequence-to-sequence approach.

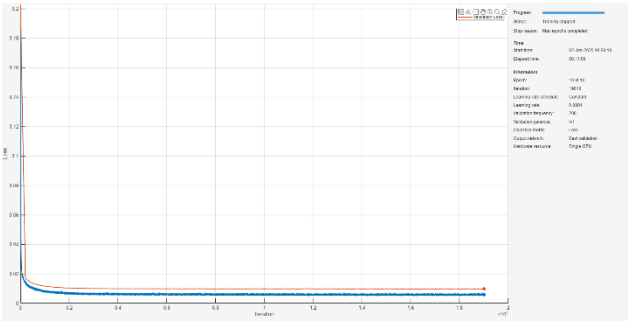

Fig. 16. Training progress of Net50_2

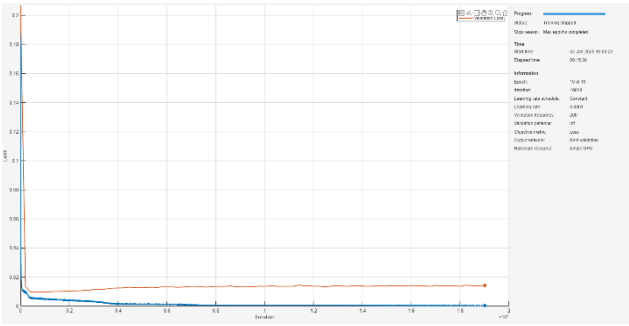

Fig. 17. Training progress of BiNet50_2

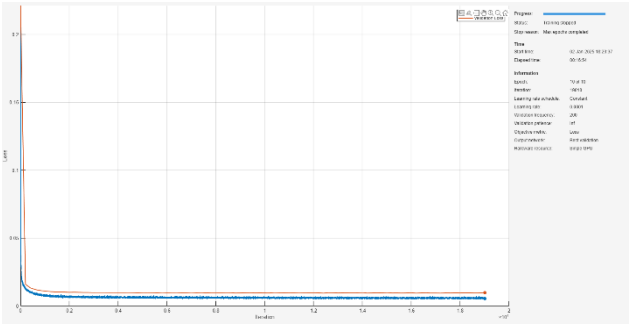

Fig. 18 Training progress of duNet50_2

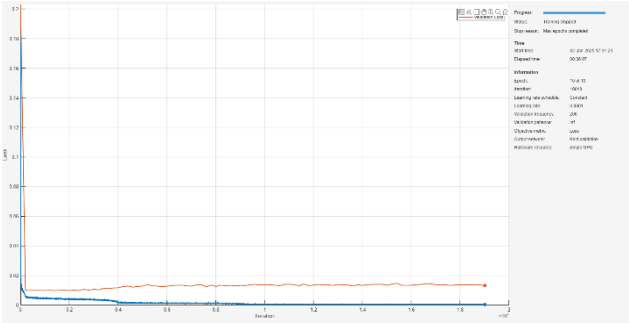

Fig. 19. Training progress of duBiNet50_2

In the sequence-to-one approach, the orange line indication the validation loss seems to be diverging, indicating an overfitting. BiLSTM models seems to start overfitting in the Epoch 3

Table below shows the loss values of all trained networks.

TABLE I. RESULT ANALYSIS OF DIFFERENT MODELS

Network Results		
Network Name	***train_loss***	***val_loss***
Net50_1	0.0071076	0.0071313
BiNet50_1	0.0065434	0.006447
duNet_1	0.0070836	0.0072106
duBiNet_1	0.0062719	0.006406
Net50_2	0.0055775	0.0095978
BiNet_2	0.0004044	0.013894
duNet_2	0.0055766	0.0095131
duBiNet_2	0.0004277	0.0097761

B. Post Processing

After testing our models with the sequences from the test dataset, we give the model a seed sequence with 50 time steps and generated a new sequence with 200 time steps. But we need to reformat them before converting the note matrix back to midi format.

First, we denormalize the note matrices and add back the extracted columns of midi channel, velocity (fixed at 60), duration and onset in seconds.

Lastly, we convert the note matrices using MIDI toolbox.

VI. CONCLUSION

A. Conclusion

Although this work didn't achieve the goal of generating complex compositions that sound "human-like", it achieves to train a model that is capable of predicting the notes after a sequence and catching patterns in a classical composition. Music is one of the most complex type of arts and it is hard to copy the nature and creativity of humans with such a simple design. But even with the limited computational power and time a meaningful composition can be achieved. Most of the networks proposed today are trained for hours even days. These models are only trained for 30 minutes.

We can see that using a LSTM instead of a BiLSTM can be a better approach as BiLSTM converges the predictions too much. Also using a sequence-to-sequence approach can be better because in sequence-to-one regression approach, model also converges the predicted note values as the sequences are shifted by 1, in other words, overfits a lot.

We believe that follow up research along with better computational power can optimize the models for better results.

B. Future Work

To improve the performance of the model, one of the best solutions can be adding an Attention layer [29].

An attention layer is a mechanism used in deep learning to focus on the most relevant parts of the input data when making predictions. It is commonly used in sequence-based tasks, such as natural language processing,

time series analysis, and music generation. The attention mechanism allows the model to weigh different parts of the input sequence dynamically, based on their importance for the current prediction task. In music generation it can be used to highlight specific notes or chords that are influential for predicting the next sequence.

Adding convolution layers can also improve the performance [24]. CNN has the function of feature recognition and extraction, which inspired us to use the convolution layer to extract the features of musical melody for our future work. Using our approach with BALSTM proposed by Johnson can also result in a better performance.

Piano roll and frequency domain approaches can also be researched for using them as a dataset for the proposed models.

REFERENCES

[1] S. Mangal, R. Modak, and P. Joshi, "LSTM based music generation system,"2019,arXiv:1908.01080.[Online]. Available:https://arxiv.org/abs/1908.01080

[2] U. Rawat and S. Singh, "Challenges in Music Generation Using Deep Learning," *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, Greater Noida, India, 2022, pp. 553-558

[3] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: The difficulty of learning long-term dependencies," 2001.

[4] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural Computation*, vol. 9, 1997, pp. 1735–1780.

[5] A. Van Der Merwe and W. Schulze, "Music Generation with Markov Models," in *IEEE MultiMedia*, vol. 18, no. 3, pp. 78-85, March 2011.

[6] I. Xenakis, Formalized Music: Thought and Mathematics in Composition, vol. 6. Stuyvesant, NY: Pendragon Press, 1992.

[7] C. . -C. J. Chen and R. Miikkulainen, "Creating melodies with evolving recurrent neural networks," *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, Washington, DC, USA, 2001, pp. 2241-2246 vol.3

[8] P. M. Todd and G. Loy, "A Connectionist Approach to Algorithmic Composition," in *Music and Connectionism*, MIT Press, 2003, pp. 173–194.

[9] M. Dua, R. Yadav, D. Mamgai, and S. Brodiya, ‘‘An improved RNN-LSTM based novel approach for sheet music generation,’’ Proc. Comput. Sci., vol. 171, pp. 465–474, Jul. 2020.

[10] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” CoRR, vol. abs/1609.03499,2016.

[Online]. Available: http://arxiv.org/abs/1609.03499

[11] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi, "Neural audio synthesis of musical notes with WaveNet autoencoders,"*arXiv:1704.01279*, 2017. [Online]. Available:https://arxiv.org/abs/1908.01080.

[12] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27 (NIPS'14)*, Cambridge, MA, USA: MIT Press, 2014, pp. 2672–2680.

[13] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, ‘‘MidiNet: A convolutional generative adversarial network for symbolic-domain music generation,’’ 2017, arXiv:1703.10847.

[14] H. W. Dong, W. Y. Hsiao, L. C. Yang, and Y. H. Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018.

[15] O. Mogren, ‘‘C-RNN-GAN: Continuous recurrent neural networks with adversarial training,’’ 2016, arXiv:1611.09904.

[16] D. Eck and J. Schmidhuber, "Finding temporal structure in music: blues improvisation with LSTM recurrent networks," *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, Martigny, Switzerland, 2002, pp. 747-756.

[17] P. Y. Nikhil Kotecha, "Generating Music using an LSTM Network," arXiv.org, vol. arXiv:1804.07300, 2018.

[18] A. Remesh, A. P. K and M. S. Sinith, "Symbolic Domain Music Generation System Based on LSTM Architecture," *2022 Second International Conference on Next Generation Intelligent Systems (ICNGIS)*, Kottayam, India, 2022, pp. 1-4.

[19] K. Zhao, S. Li, J. Cai, H. Wang and J. Wang, "An Emotional Symbolic Music Generation System based on LSTM Networks," *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, Chengdu, China, 2019, pp. 2039-2043.

[20] D. D. Johnson, "Generating polyphonic music using tied parallel networks," in *International Conference on Evolutionary and Biologically Inspired Music and Art*, Springer, Cham, 2017, pp. 128–143.

[21] A. Graves, "Generating sequences with recurrent neural networks," *arXiv:1308.0850*, 2013.

[22] D. Johnson, "Composing music with recurrent neural networks," [Online].Available: https://www.danieldjohnson.com/2015/08/03/composing-music-with-recurrent-neural-networks/. [Accessed: Dec. 18, 2024].

[23] A. Kumar Bairwa, S. Bhat, T. Sawant and R. Manoj, "MGU-V: A Deep Learning Approach for Lo-Fi Music Generation Using Variational Autoencoders With State-of-the-Art Performance on Combined MIDI Datasets," in *IEEE Access*, vol. 12, pp. 143237-143251, 2024

[24] Y. Huang, X. Huang, and Q. Cai, "Music Generation Based on Convolution-LSTM," *Comput. Inf. Sci.*, vol. 11, no. 3, pp. 50–56, 2018.

[25] Q. Kong, B. Li, J. Chen and Y. Wang, “GiantMIDI-Piano: A large-scale MIDI dataset for classical piano music”, *arXiv:2010.07061*, 2020.

[26] P. Toiviainen and T. Eerola, "MIDI Toolbox 1.1," GitHub, 2016. [Online]. Available: https://github.com/miditoolbox/1.1. [Accessed: Dec. 18, 2024].

[27] L. C. Yang and A. Lerch, "On the evaluation of generative models in music," *Neural Comput. & Applic.*, vol. 32, pp. 4773–4784, 2020

[28] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," in *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673-2681, Nov. 1997

[29] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All You Need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, arXiv: 1706.03762