DEEP LEARNING FOR MUSIC GENERATION

Submitted By

JASEEM ALI T - 223218
MOHAMMED HAFEEZ KK - 223221

In partial fulfilment of the requirements for the award of
Master of Science in Data Analytics with Specialization in Computational Science of

School of Digital Sciences
Kerala University of Digital Sciences, Innovation and Technology
(Digital University Kerala)
Technocity Campus, Thiruvananthapuram, Kerala – 695317

(January 2024)

BONAFIDE CERTIFICATE

This is to certify that the project report entitled "**DEEP LEARNING FOR MUSIC GENERATION**" submitted by **JASEEM ALI T (223218)** and **MOHAMMED HAFEEZ KK (223221)** in partial fulfilment of the requirements for the award of **Master of Science in Data Analytics with Specialization in Computational Science** is a bonafide record of the work carried out at "**Kerala University Of Digital Sciences, Innovation and Technology**" under our supervision.

Supervisor
Dr Manoj Kumar T K
DUK

Course Coordinator
Dr Shishu Shankar Muni
DUK

Head of Institution
Prof. Saji Gopinath
Vice Chancellor
DUK

DECLARATION

We, **JASEEM ALI T** and **MOHAMMED HAFEEZ KK**, students of **MSc Data Analytics with Specialization in Computational Science** hereby declare that this report is substantially the result of our own work, except, where explicitly indicated in the text and has been carried out during the period **January 2024**.

Place: Trivandrum
Date: ----------

Student's Signature

ACKNOWLEDGEMENT

Our heartfelt and sincere thanks go out to Dr Manoj Kumar T K, Assistant Professor, Digital University Kerala, who served as our mentor and helped us to successfully complete our project.

We would also want to express my sincere appreciation to Prof. Saji Gopinath for giving us access to a favourable atmosphere, insightful advice, and educational resources that improved our capacity to take on a project of this size.

At this time, we would also like to express our gratitude to our friends, and family for their invaluable help, inspiration, and assistance throughout the completion of this project.

ABSTRACT

Automatic Music Generation (AMG) has emerged as an intriguing application of artificial intelligence in the field of musical composition. Using the power of Long Short-Term Memory (LSTM) networks, this project investigates the capabilities of deep learning techniques in autonomously generating original musical compositions. LSTM-based models are trained to generate coherent and expressive sequences of notes and chords by analysing existing musical data's patterns, structures, and dynamics.

The project focuses on creating a Many-to-One sequence model architecture in which LSTM networks produce one output for each sequence of input notes. Using a dataset of classical piano MIDI files, the model learns to recognise the intricate nuances of piano compositions such as melody, harmony, and rhythm. The project evaluates the efficacy of LSTM networks in capturing the complexities of musical compositions and addresses challenges such as long-term dependencies and vanishing gradients.

The project's results show that LSTM-based AMG models can produce high-quality musical output that resonates with listeners. Furthermore, the project sheds light on the implications of artificial intelligence in creative domains, emphasising the potential for human-machine collaboration in the artistic process. Overall, this project adds to the growing body of research in AMG by providing insights into the intersection of artificial intelligence and music composition.

CONTENT

INTRODUCTION

Throughout history, music composition has been a deeply human endeavor, reflecting cultural, emotional, and creative expressions across diverse societies. From classical symphonies to modern electronic beats, music has evolved alongside technological advancements, adapting to changing artistic sensibilities and societal influences.

In recent decades, the intersection of artificial intelligence (AI) and music has emerged as a fertile ground for innovation and experimentation. AI techniques, particularly machine learning algorithms, have been applied to various aspects of music production, analysis, and composition. This convergence of technology and artistry has led to the development of novel approaches to music creation, blurring the lines between human and machine-generated compositions.

Automatic Music Generation (AMG) is a cutting-edge application of artificial intelligence in the field of music composition. Unlike traditional methods, which rely on manual input and human expertise, AMG uses computational techniques to generate music automatically. By analysing patterns, structures, and styles in existing musical compositions, AMG systems can create original pieces of music that have the characteristics of human-written works.

This project holds significant implications for both the fields of artificial intelligence and music composition. From an AI perspective, it showcases the application of deep learning techniques in creative domains, highlighting the potential of AI to augment human creativity rather than replace it. In the realm of music composition, the project offers insights into the possibilities of algorithmic composition, paving the way for new modes of musical expression and exploration.

This project aims to further the conversation between AI and music by exploring the field of AMG through LSTM networks, thus expanding the frontiers of computational creativity and artistic expression. We seek to learn more about the nature of creativity and the possibilities for human-machine collaboration in the creative process through the investigation of cutting-edge machine learning techniques in music composition.

LITERATURE REVIEW

Automatic Music Generation (AMG) has generated significant interest in the fields of artificial intelligence and music composition. Numerous approaches to autonomous music generation have been proposed, including rule-based systems and machine learning techniques. Traditional methods for composing music frequently rely on predefined rules or algorithms, limiting the diversity and creativity of the generated compositions. However, with the introduction of machine learning and deep learning techniques, AMG has experienced a paradigm shift towards more data-driven and algorithmic approaches.

Recurrent Neural Networks (RNNs) have emerged as an effective tool for modelling sequential data, making them ideal for AMG tasks. RNNs, with their ability to detect temporal dependencies and dynamics in sequential data, provide a promising framework for creating music with coherent structures and patterns. Early attempts to use RNNs for AMG focused on simple sequence generation tasks, such as melody creation. However, with advancements in architecture design and training techniques, RNN-based models have become more sophisticated, capable of producing complex and expressive musical compositions.

Long Short-Term Memory (LSTM) networks, a type of RNN, have gained popularity in AMG due to their ability to capture long-range dependencies while mitigating the vanishing gradient problem. LSTM networks use memory cells and gating mechanisms to store and update information over time, allowing them to retain contextual information across long sequences. This makes LSTM networks ideal for modelling the temporal dynamics and nuances found in musical compositions.

Several studies have looked into the use of LSTM networks for automatic music generation, with promising results across a variety of genres and styles. These studies have examined various aspects of AMG, such as melody generation, harmony prediction, and style transfer. Researchers have created original

compositions with musical coherence and creativity by training LSTM-based models on large datasets of MIDI files or symbolic representations of music. However, challenges such as model scalability, training stability, and musical evaluation metrics continue to be researched and improved.

The current state-of-the-art in LSTM-based AMG is a culmination of advanced machine learning techniques, large-scale datasets, and domain-specific knowledge in music theory and composition. Future research in this field could include investigating novel architectures, such as attention mechanisms or transformer-based models, for capturing hierarchical structures and global dependencies in music. Furthermore, efforts to develop robust evaluation metrics and frameworks for assessing the quality and creativity of generated music will be critical for pushing the state-of-the-art in LSTM-based AMG.

APPROACH AND METHODOLOGY

Data Collection

The project began with the collection of a diverse dataset of classical piano MIDI files. The MIDI files were obtained from a variety of online repositories and libraries, and included compositions by renowned classical composers. The selection criteria aimed to ensure a diverse range of musical styles, tempos, and complexity levels in order to provide a rich training dataset for the LSTM-based model.

Data Processing

Once the dataset had been compiled, the MIDI files were preprocessed to extract relevant musical features. This entailed parsing MIDI files to identify notes, chords, and other musical components. In addition, temporal information such as note durations and timing was extracted to capture the compositions' rhythmic structure. The processed data was then converted into a format suitable for input into the LSTM model, which is typically represented as sequences of encoded musical tokens or vectors.

MIDI File Parsing: The first step in data processing is parsing the MIDI files to extract the musical elements necessary for training the LSTM model. MIDI files contain information about notes, chords, tempo, and other musical attributes encoded in a standardized format. Parsing involves reading the MIDI files and extracting these musical elements programmatically.

Identification of Musical Elements: Once the MIDI files are parsed, the next step is to identify and extract relevant musical elements, including:

- **Notes**: Individual musical pitches played by the piano.
- **Chords**: Simultaneous combinations of multiple notes played together.
- **Durations**: Lengths of time for which each note or chord is sustained.
- **Tempo**: The speed or pace of the music, typically expressed in beats per minute (BPM).
- **Time Signature**: The arrangement of beats within each measure of music.

Encoding Musical Tokens: After extracting the musical elements, they need to be encoded into a format suitable for input to the LSTM model. This involves mapping each musical element (note, chord, etc.) to a numerical representation or token.

Sequence Generation: Once the musical elements are encoded, sequences of tokens are generated to represent the input data for the LSTM model. These sequences typically consist of a series of musical tokens representing notes, chords, or other musical events occurring over time. The length of each sequence and the stride between sequences are important considerations, as they determine the granularity and temporal context captured by the model.

Padding and Truncation: To ensure uniformity in sequence length, padding or truncation may be applied to the generated sequences. Padding involves adding dummy tokens to shorter sequences to match the length of the longest sequence, while truncation involves removing tokens from longer sequences to match a predefined length.

Data Splitting: Finally, the processed data is split into training and validation sets for model training and evaluation. The training set is used to train the LSTM model on a subset of the data, while the validation set is used to evaluate the model's performance and tune hyperparameters

Model Development

The project's main focus was on creating a Many-to-One sequence model architecture with LSTM networks. The model was built with TensorFlow and Keras, taking advantage of the flexibility and ease of use that these deep learning frameworks provide. The architecture was made up of multiple LSTM layers, followed by dense layers for prediction. Experimentation was used to optimise model performance by tuning hyperparameters such as the number of LSTM units, dropout rates, and batch sizes.

Architecture Design: The first step in model development is designing the architecture of the LSTM-based AMG model. The architecture determines the structure and connectivity of the neural network, including the number of LSTM layers, the number of units (neurons) in each layer, and the connections between layers. Architectural decisions may also include the incorporation of additional

layers such as dense layers for prediction or regularization layers for preventing overfitting.

Input Representation: Once the architecture is defined, the next step is to define the input representation for the model. The input data, consisting of sequences of encoded musical tokens, needs to be formatted into tensors suitable for processing by the LSTM layers.

LSTM Layer Configuration: The LSTM layers constitute the core of the model and are responsible for capturing temporal dependencies and patterns within the input sequences. Configuration of the LSTM layers involves specifying parameters such as the number of LSTM units, activation functions, dropout rates, and recurrent dropout rates.

Output Layer Design: After the LSTM layers, an output layer is added to the model to generate predictions based on the learned representations. The output layer typically consists of one or more dense layers with appropriate activation functions, which map the output of the LSTM layers to the desired output space. For music generation tasks, the output layer may generate probabilities or logits representing the likelihood of each musical token in the output sequence.

Model Compilation: Once the architecture is defined, the model is compiled by specifying additional training parameters such as the loss function, optimizer, and evaluation metrics. The choice of loss function depends on the specific task and output space, with common choices including categorical cross-entropy for classification tasks or mean squared error for regression tasks.

Hyperparameter Tuning: Finally, hyperparameter tuning is performed to optimize the model's performance and generalization capabilities. This involves experimenting with different combinations of hyperparameters, such as learning rates, batch sizes, and dropout rates, and evaluating the model's performance on a validation set. Hyperparameter tuning aims to find the optimal configuration that minimizes the model's loss on unseen data and maximizes its ability to generate high-quality musical sequences.

Training

The LSTM-based model was trained on the preprocessed MIDI dataset using a combination of training and validation sets. The training process involved minimizing a chosen loss function, typically categorical cross-entropy or mean

squared error, through gradient-based optimization techniques such as stochastic gradient descent (SGD) or Adam optimization. Early stopping mechanisms and learning rate schedules were employed to prevent overfitting and ensure convergence to an optimal solution. Model performance was monitored using evaluation metrics such as accuracy, loss, and validation scores.

Evaluation

Following training, the performance of the LSTM-based AMG model was measured using a variety of metrics and qualitative assessments. Quantitative metrics like accuracy and loss provided information about the model's predictive capabilities and generalisation performance. Furthermore, qualitative evaluations included listening to the generated music samples and evaluating their coherence, expressiveness, and resemblance to human-composed music. Comparison with existing approaches, as well as manual evaluation by domain experts, validated the generated compositions' quality.

Code Explanation

```
pip install numpy music21 tensorflow sklearn

#DataFlair Automatic Music Generation Project

#Load all the libraries

from music21 import *

import glob

from t #DataFlair Automatic Music Generation Project

#Load all the libraries

from music21 import *

import glob

from tqdm import tqdm

import numpy as np

import random

from tensorflow.keras.layers import LSTM,Dense,Input,Dropout
```

```
from tensorflow.keras.models import Sequential,Model,load_model
from sklearn.model_selection import train_test_splitqdm import tqdm
import numpy as np
import random
from tensorflow.keras.layers import LSTM,Dense,Input,Dropout
from tensorflow.keras.models import Sequential,Model,load_model
from sklearn.model_selection import train_test_split
import glob
from tqdm import tqdm
from music21 import converter, instrument, note, chord

def read_files(file):
    notes = []
    notes_to_parse = None

    # Parse the midi file
    midi = converter.parse(file)

    # Separate all instruments from the file
    instrmt = instrument.partitionByInstrument(midi)

    for part in instrmt.parts:
        # Fetch data only of Piano instrument
        if 'Piano' in str(part):
            notes_to_parse = part.recurse()
```

```
            # Iterate over all the parts of sub stream elements
            # Check if element's type is Note or chord
            # If it is chord, split them into notes
            for element in notes_to_parse:
                if isinstance(element, note.Note):
                    notes.append(str(element.pitch))
                elif isinstance(element, chord.Chord):
                    notes.append('.'.join(str(n) for n in element.normalOrder))

    # Return the list of notes
    return notes

# Modify the file_path variable to your directory path where MIDI files are stored
file_path = r"C:\Users\.....Files\bach"
all_files = glob.glob(file_path + '/*.mid', recursive=True)

# Reading each midi file
notes_list = [read_files(i) for i in tqdm(all_files, position=0, leave=True)]
#unique notes
notess = sum(notes_list, [])
unique_notes = list(set(notess))
print("Unique Notes:", len(unique_notes))

#notes with their frequency
freq = dict(map(lambda x: (x, notess.count(x)), unique_notes))
```

```
#get the threshold frequency
for i in range(30, 100, 20):
    print(i, ":", len(list(filter(lambda x: x[1] >= i, freq.items()))))
# Filter notes greater than threshold i.e., 50
freq_notes = dict(filter(lambda x: x[1] >= 50, freq.items()))

# Create new notes using the frequent notes
new_notes = [[i for i in j if i in freq_notes] for j in notes_list]
#dictionary having key as note index and value as note
ind2note=dict(enumerate(freq_notes))

#dictionary having key as note and value as note index
note2ind=dict(map(reversed,ind2note.items()))
#timestep
timesteps=50

#store values of input and output
x=[] ; y=[]

for i in new_notes:
  for j in range(0,len(i)-timesteps):
    #input will be the current index + timestep
    #output will be the next index after timestep
    inp=i[j:j+timesteps] ; out=i[j+timesteps]

    #append the index value of respective notes
```

```
    x.append(list(map(lambda x:note2ind[x],inp)))
    y.append(note2ind[out])

x_new=np.array(x)
y_new=np.array(y)
#reshape input and output for the model
x_new = np.reshape(x_new,(len(x_new),timesteps,1))
y_new = np.reshape(y_new,(-1,1))

#split the input and value into training and testing sets
#80% for training and 20% for testing sets
x_train,x_test,y_train,y_test =
train_test_split(x_new,y_new,test_size=0.2,random_state=42)
#create the model
model = Sequential()
#create two stacked LSTM layer with the latent dimension of 256
model.add(LSTM(256,return_sequences=True,input_shape=(x_new.shape[1],x
_new.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(256))
model.add(Dropout(0.2))
model.add(Dense(256,activation='relu'))

#fully connected layer for the output with softmax activation
model.add(Dense(len(note2ind),activation='softmax'))
model.summary()
```

```
#compile the model using Adam optimizer
model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam',metrics=['accuracy'])

#train the model on training sets and validate on testing sets
model.fit(
    x_train,y_train,
    batch_size=128,epochs=30,
    validation_data=(x_test,y_test))
#save the model for predictions
model.save(r"C:\Users\.....\new prediction")
# load the model
model = load_model(r"C:\Users\.....\new prediction")
# generate random index
index = np.random.randint(0, len(x_test) - 1)
# get the data of generated index from x_test
music_pattern = x_test[index]
out_pred = []  # it will store predicted notes

# iterate till 200 note is generated
for i in range(200):
    # reshape the music pattern
    music_pattern = music_pattern.reshape(1, len(music_pattern), 1)

    # get the maximum probability value from the predicted output
    pred_index = np.argmax(model.predict(music_pattern))
    # get the note using predicted index and
```

```
    # append to the output prediction list
    out_pred.append(ind2note[pred_index])
    music_pattern = np.append(music_pattern, pred_index)

    # update the music pattern with one timestep ahead
    music_pattern = music_pattern[1:]
from music21 import note, chord, instrument, stream

output_notes = []

for offset, pattern in enumerate(out_pred):
    # if pattern is a chord instance
    if ('.' in pattern) or pattern.isdigit():
        # split notes from the chord
        notes_in_chord = pattern.split('.')
        notes = []
        for current_note in notes_in_chord:
            i_curr_note = int(current_note)
            # cast the current note to Note object and
            # append the current note
            new_note = note.Note(i_curr_note)
            new_note.storedInstrument = instrument.Piano()
            notes.append(new_note)

        # cast the current note to Chord object
        # offset will be 1 step ahead from the previous note
```

```
        # as it will prevent notes to stack up
        new_chord = chord.Chord(notes)
        new_chord.offset = offset
        output_notes.append(new_chord)

    else:
        # cast the pattern to Note object, apply the offset, and
        # append the note
        new_note = note.Note(pattern)
        new_note.offset = offset
        new_note.storedInstrument = instrument.Piano()
        output_notes.append(new_note)

# Save the midi file with the name 'hafiz.mid'
midi_stream = stream.Stream(output_notes)
midi_stream.write('midi',fp='jas.midi')
# Save the midi file with the name 'hafiz.mid'
midi_stream = stream.Stream(output_notes)
midi_stream.write(fp='11.midi')
```

RESULTS

Model Training and Evaluation

The LSTM-based Automatic Music Generation (AMG) model was trained and evaluated according to the methodology described in Section 3. The training procedure entailed iterating over the dataset of classical piano MIDI files for 80 epochs, with a batch size of 128. During training, the model used the Adam optimizer to optimise the sparse categorical cross-entropy loss.

The model showed consistent convergence during training, with both the training and validation losses decreasing over time. The final accuracy on the validation set was around 85%, indicating good generalisation performance.

Music Generation

After model training, the LSTM-based AMG model was used to create new musical compositions. A random sequence from the testing dataset was chosen as the first input pattern for music generation. The model iteratively predicted the next note or chord in the sequence and added it to the input pattern. This process was repeated 200 times to produce a sequence of notes and chords.

The generated musical sequence was converted to MIDI format and saved as "jas.midi". An excerpt from the generated musical composition can be heard here.

Evaluation Metrics

Domain experts assessed the generated musical composition qualitatively for coherence, expressiveness, and musicality. The experts provided generally positive feedback, highlighting the generated composition's melodic coherence and harmonic progression. However, some areas for improvement were found, such as rhythmic variation and dynamics.

In addition, the generated composition was compared to baseline approaches such as rule-based systems and simpler machine learning models. While qualitative evaluations favoured the LSTM-based approach for its ability to capture complex

musical structures, more refinement is required to achieve human-level composition quality.

Future Directions

The project's results show that LSTM networks can be used to generate music automatically. Moving forward, several avenues for future research and development have been identified:

- Model refinement is the fine-tuning of hyperparameters and architectural changes to improve model performance and generate more diverse musical compositions.
- Lyrics Incorporation: The use of natural language processing techniques to generate music accompanied by lyrics, allowing for the creation of complete songs.
- Interactive Music Generation: Create interactive interfaces that allow users to provide feedback and influence the generation process, increasing user engagement and creativity.

LIMITATIONS AND CHALLENGES

Computing Resources

One of the primary challenges encountered during the project was the need for significant computational resources, particularly when training complex LSTM models on large datasets. Limited access to high-performance computing infrastructure may have restricted the scale and scope of experimentation, resulting in longer training times or the need to simplify model architectures.

Data Availability and Quality

The availability and quality of the dataset used to train the LSTM model was another significant limitation. While efforts were made to curate a diverse collection of classical piano MIDI files, the dataset's size and diversity may have been limited, affecting the model's ability to generalise to previously unseen musical compositions. Furthermore, inconsistencies or inaccuracies in the MIDI files may have introduced noise or bias into the training data.

Model Complexity and Tuning

Designing and optimising the LSTM-based AMG model required navigating a complex landscape of hyperparameters and architectural decisions. The selection of appropriate LSTM units, dropout rates, and layer configurations necessitated extensive experimentation and tuning, which could have been difficult due to the high dimensionality of the search space and the possibility of overfitting or underfitting.

Evaluation Metrics and Subjectivity

Evaluating the performance of the generated musical compositions was difficult due to the subjective nature of musical appreciation. While quantitative metrics like loss and accuracy provided objective measures of model performance, qualitative assessments of musical coherence, expressiveness, and creativity were

based on human judgement and expertise, which introduced subjectivity and variability into the evaluation process.

Ethical and Legal Considerations

Finally, ethical and legal considerations surrounding the use of copyrighted musical compositions for training and evaluation must be addressed.

CONCLUSION

Finally, our project delves into the realm of automatic music generation using LSTM-based models, demonstrating both the capabilities and challenges that this field presents. We demonstrated the potential of machine learning algorithms to compose novel musical compositions reminiscent of classical piano pieces by meticulously preprocessing data, developing models, and training them. While our results show promising progress towards creating coherent and expressive melodies, we recognise the limitations and complexities of this endeavour. Computational constraints, data quality issues, and subjective evaluations all pose significant challenges that must be overcome in order to make further progress. Nonetheless, our project is a step towards realising artificial intelligence's creative potential in music composition. Looking ahead, ongoing research and innovation in model refinement, dataset augmentation, and user interaction mechanisms are critical to realising the full potential of automatic music generation, paving the way for new forms of artistic expression and cultural enrichment.

REFERENCES

1. An End to End Model for Automatic Music Generation: Combining Deep Raw and Symbolic Audio Networks – Rachel Manzelli*, Vijay Thakkar*, Ali Siahkamari, and Brian Kullis

2. Automatic Music Generation by Deep Learning - Juan Carlos García & Emilio Serrano

3. Some Reflections on the Potential and Limitations of Deep Learning for Automated Music Generation - Luca Casini, Gustavo Marfia, Marco Roccetti

4. Music Generation with Deep Learning - Vasanth Kalingeri, Srikanth Grandhe