MUSIC NOTES GENERATION

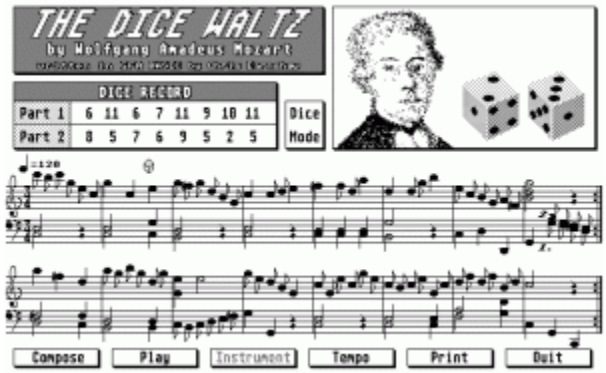
OVERVIEW

Automatic music generation refers to the process of using computer algorithms and artificial intelligence techniques to create music without direct human intervention. It involves the generation of musical compositions, melodies, harmonies, and rhythms using software or AI models. This technology can range from simple algorithms that create basic tunes to advanced AI systems that produce complex, highly expressive music.

Music is defined as a collection of tones of different frequencies. So, the Automatic Music Generation is a process of composing a short piece of music with minimum human intervention. Automatic music generation is a creative process that involves the generation of musical pieces, considering various musical parameters such as pitch intervals, notes, chords, and tempo. This automated approach to music creation has the potential to revolutionize how we compose and produce music. It not only provides new avenues for musicians and composers but also offers opportunities for AI-driven music generation in various applications, from entertainment to advertising.

HISTORY

It all started by randomly selecting sounds and combining them to form a piece of music. In 1787, Mozart proposed a Dice Game for these random sound selections. He composed nearly 272 tones manually. Then, he selected a tone based on the sum of 2 dice.



Musical Grammar comprehends the knowledge necessary to the just arrangement and combination of musical sounds and to the proper performance of musical compositions

– Foundations of Musical Grammar

In the early 1950s, Iannis Xenakis used the concepts of Statistics and Probability to compose music – popularly known as Stochastic Music. He defined music as a sequence of elements (or sounds) that occurs

by chance. Hence, he formulated it using stochastic theory. His random selection of elements was strictly dependent on mathematical concepts.

The origins of this endeavour trace back to the late 18th century when Mozart, one of the most prolific composers in history, proposed a "Dice Game" to create music by randomly selecting musical elements. Mozart's manual composition of nearly 272 tones serves as a testament to the early creative approaches in music generation. Additionally, the article discusses the use of musical grammar in generating music, emphasizing the significance of proper musical structure in compositions.

OBJECTIVE

To create an LSTM that generates notes for music. For such projects, we feed the network a series of strings and the network predicts the next string in the series based on the information it is trained on. We decided to use the same principle on the music notes.

The concept of automatic music generation, where a system composes music with minimal human intervention, has garnered significant attention in the field of music composition. The focus here is on generating music using a Piano Instrument and understanding the underlying technical aspects of LSTM in the context of music generation.

INTRODUCTION

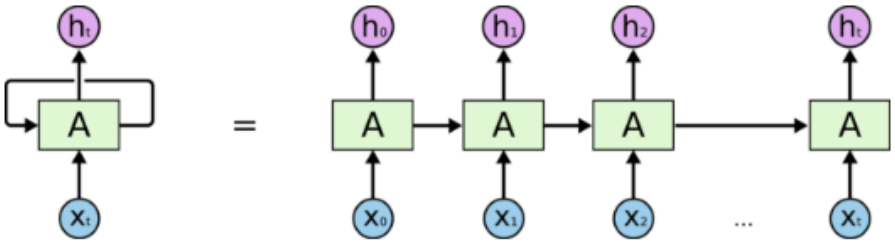
MUSIC: Music is essentially composed of Notes and Chords.

Let me explain these terms from the perspective of the piano instrument:

- Note: The sound produced by a single key is called a note
- Chords: The sound produced by 2 or more keys simultaneously is called a chord. Most chords contain at least 3 key sounds
- Octave: A repeated pattern is called an octave. Each octave contains 7 white and 5 black keys

LSTM

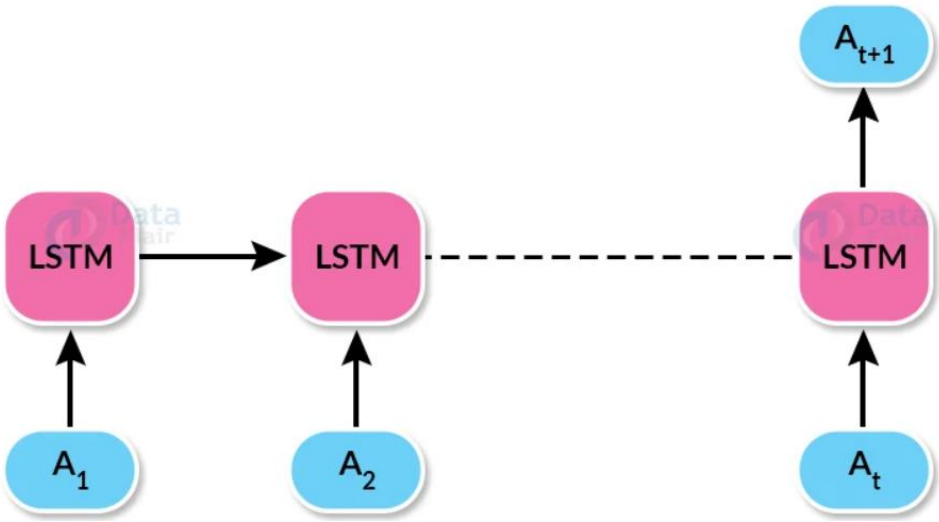
Long Short-Term Memory Model, popularly known as LSTM, is a variant of Recurrent Neural Networks (RNNs) that is capable of capturing the long-term dependencies in the input sequence. LSTM has a wide range of applications in Sequence-to-Sequence modeling tasks like Speech Recognition, Text Summarization, Video Classification, and so on.



An unrolled recurrent neural network.

Long Short-Term Memory (LSTM) is a type of RNN (Recurrent Neural Network) that solves some scenarios where RNN failed. LSTM solves Long-Term dependency problem of RNN i.e., RNN networks store data of previous output in a memory for a truly brief period.

LSTM also solves the problem of Vanishing Gradient i.e., to get the best result, the model tries to minimize the loss after every time step by calculating loss with respect to some weights. After reaching a certain period, this weight becomes so less that it approximates to zero or vanishes and the model stops training.



At each timestep, an amplitude value is fed into the Long Short Term Memory cell – it then computes the hidden vector and passes it on to the next timesteps. The current hidden vector at timestep ht is computed based on the current input at and previously hidden vector ht-1. This is how the sequential information is captured in any Recurrent Neural Network:

DRAWBACK OF LSTM

LSTM requires lots of resources and time to get trained for real world applications. Randomly initializing different weights makes LSTM networks behave like feed forward neural networks. Therefore, they require small weight initialization instead.

Our model will be a Many-to-one sequence model as there will be one output for every sequence of input notes after each timesteps.

Input to the LSTM model will be the amplitude(A) of these notes which are recorded at different intervals of time which computes hidden vectors and passes to the next layer for the next timestep(t).

WHY ONLY DEEP LEARNING?

Deep Learning is a field of Machine Learning which is inspired by a neural structure. These networks extract the features automatically from the dataset and can learn any non-linear function. That is why Neural Networks are called Universal Functional Approximators.

The article asserts that deep learning architectures have emerged as a state-of-the-art solution for automatic music generation. The reasoning is rooted in the premise that deep learning models can automatically extract features from datasets and learn complex, non-linear functions. Drawing parallels with the use of neural networks in other fields like Natural Language Processing and Computer Vision, the article sets the stage for the exploration of deep learning in music composition.

CHALLENGES IN MUSIC GENERATION WITH LSTM

Using LSTM for music generation introduces specific challenges. The demands of training such models can be computationally intensive and time demanding. Additionally, the risk of vanishing gradients remains a concern. The article provides insights into how LSTM handles these hurdles, ensuring a comprehensive understanding of the technology's practical applications.

PROJECT SCOPE

1. DATA COLLECTION

   MIDI (Musical Instrument Digital Interface) is a widely used file format and protocol for representing and communicating music data. It is a versatile and efficient way to store and exchange musical information between different devices, software, and musical instruments. Here are some key points about MIDI files:

- Data Format: MIDI files do not contain actual audio recordings. Instead, they store musical instructions in a digital format. These instructions can include information about notes, pitch, duration, tempo, dynamics, and more.

- Instrument-agnostic: MIDI is instrument-agnostic, meaning it does not specify the sound source. It can control any musical instrument, synthesizer, or software that understands MIDI instructions. This flexibility allows a MIDI file to be used to play back music using different instruments.

- Tracks and Channels: MIDI files can have multiple tracks, and each track can contain musical data for different instruments or voices. Within each track, there are 16 MIDI channels, which allow for further differentiation and control.

- Polyphony: MIDI supports polyphony, meaning it can handle multiple notes played simultaneously. This is crucial for representing chords and complex musical passages.

- Editing and Manipulation: MIDI files can be easily edited and manipulated using various software tools. Musicians and composers often use MIDI sequencers to create, edit, and arrange music.

- Size Efficiency: MIDI files are minor compared to audio files like WAV or MP3. This makes them ideal for applications where file size is a concern, such as video games or web applications.

MIDI files are an essential tool for musicians, composers, and producers. They provide a means to create, share, and reproduce music in a standardized, digital format. MIDI's versatility and efficiency have made it a fundamental component of modern music production and performance.

2. DATA PREPROCESSING

Once the MIDI files are loaded, a function is created to extract musical notes from these MIDI files. If a chord is detected, it is separated into its individual notes.

The Python function is designed to extract musical notes and chords from a collection of MIDI files. It processes each file, separates instruments, iterates through the music's parts, and, for each element encountered (which can be a note or a chord), appends a string representation of the element to the `notes` list.

In more detail, the function begins by initializing an empty list called `notes` to store the extracted musical notes. It also initializes a variable named `pick` to `None`.

The function then iterates through each MIDI file in the `file` collection. Within each file, it partitions the music by instrument, aiming to separate the music based on the instruments used. Next, it iterates through each part of the music, collecting all musical elements that make up the music within that part.

Within each part, the `pick` variable is set to contain all the elements in a recursive manner. The recursive approach is essential because musical compositions can be hierarchical, containing elements nested within other elements.

For each element encountered in the `pick`, the function checks whether it is a single note or a chord. If it is a single note, the pitch of the note is extracted and converted to a string, which is then appended to the `notes` list. If it is a chord, the function creates a string representation of the chord by joining its component notes, and this representation is appended to the `notes` list.

The function repeats this process for each MIDI file in the collection, effectively collecting all the notes and chords from the entire dataset. After processing all the MIDI files, the function returns the `notes` list.

The result is a list containing string representations of the notes and chords found in the MIDI dataset, which can be used for further analysis or processing. The function is a valuable tool for

extracting musical information from MIDI files, which is often a necessary step in music analysis and generation tasks.

3. DISTRIBUTION OF NOTES AND THEIR OCCURENCE

The resulting bar chart visually shows the frequency of each note in the dataset, which can be helpful for gaining insights into the musical composition, identifying patterns, and making informed decisions for further analysis or music generation. The x-axis represents notes, the y-axis represents frequency, and each bar represents a specific note's frequency in the corpus.

Then we move on to calculate the number of total unique notes in the corpus.

We further go on to explore the notes dictionary finding the count of notes & recurrence, and then define a function to calculate average recurrence for a note in Corpus. We print out the most frequent & least frequent notes that appear in the corpus.

Next, we obtain the list for rare notes that have occurred less than 5000 times in the corpus and then print them out. Using this list of rare notes, we then filter out rare notes from the corpus. Optionally, you can print the length of the filtered corpus to see how many rare notes were removed.

Next, we store all the unique characters present in my corpus to build a mapping dictionary, and then find the length of corpus and length of total unique characters.

Then we build a dictionary to access the vocabulary from indices and vice versa. The dictionary has key as note and value as note index. We also built a reverse mapping dictionary with the same key value pair. Then we print the total number of characters & number of unique characters. We split the corpus in equal length of strings and output targets and find toal number of sequences in the corpus.

4. METHOD

- The model used in the implementation is based on a single-layered LSTM network, which is trained to learn the sequences of polyphonic musical notes. MIDI file format was chosen as the representation of data due to its characteristics and availability of datasets. The metadata of MIDI files provides information about the song, and it is commonly used in the field of music generation.
- The implementation addresses the vanishing or exploding gradient problem of recurrent neural
- Networks by utilizing LSTM cells, which can remember long-term dependencies.
- The paper provides a detailed analysis of the model's performance and efficiency, including the representation of weights, biases, losses, and accuracy at each step and batch.
- The implementation includes the preprocessing of MIDI files, such as reading, processing, and preparing them for input into the model.

The proposed algorithm for generating musical notes using LSTM networks. We describe the layered architecture used in the LSTM model and its intertwining connections to develop a neural network. The model is designed to learn the sequences of polyphonic musical notes over a single-layered LSTM network. The LSTM network is used to predict both the time and the note in sequence. The paper also discusses the technical challenges involved in generating music automatically and provides a detailed description of the preprocessing steps involved in preparing MIDI files for input into the model.

5. MODEL

We start with reshaping the input and output for the model. We split the data into the train test with 20% of the data in the test.

We define an LSTM model using Keres. We initialize a sequential model. A sequential model is appropriate for a plain stack of layers, where you have one layer after another.

The code snippet provided represents the architecture and configuration of a neural network model, used for a sequence-to-sequence or time series prediction task. Here is an approach in paragraph form suitable for a report:

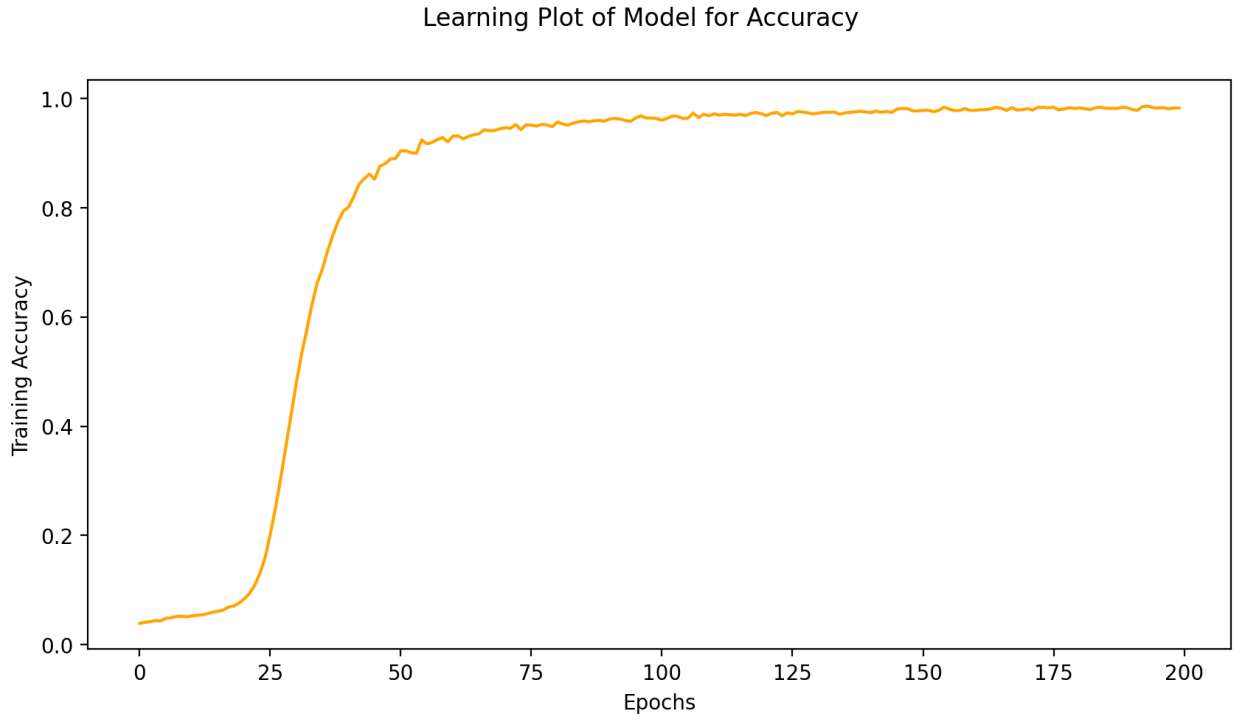
The model is constructed as a sequential neural network. It begins with an LSTM (Long Short-Term Memory) layer with 512 units, serving as the primary input layer. This LSTM layer takes input data with a shape defined by `network input`, which typically corresponds to the dimensions of the input data. Additionally, it is configured to return sequences, which is crucial for sequence prediction tasks. To prevent overfitting, a dropout layer with a specified dropout rate is inserted after the first LSTM layer.

The network's architecture continues with a second LSTM layer consisting of 256 units, followed by a dense layer with 256 units. These layers are designed to capture deeper patterns in the data and represent the neural network's hidden layers. Another dropout layer is included after the second LSTM layer to further enhance generalization by randomly deactivating a fraction of neurons during training. Finally, a dense layer with as many units as there are classes or output features in the dataset is used. The activation function "SoftMax" is applied to this layer, which is commonly used for multiclass classification tasks, to produce probability distributions over the possible outputs.

The model is compiled using specific optimization, loss, and metric configurations. The `optimizer` was chosen as the optimization algorithm to minimize the loss during training. The `loss` represents the objective function to be minimized, typically related to the task's specific goals (e.g., mean squared error for regression or categorical cross-entropy for classification). The `metrics` parameter defines the evaluation metrics used to assess the model's performance during training and validation. After configuration, the model's summary, including layer information and the number of parameters, is printed to provide an overview of the network's structure and complexity. This code structure provides a foundation for training and evaluating neural networks on a wide range of sequence prediction tasks. After this we fit the model and train it for 150 epochs with a fixed batch size.

RESULT

To visualize the performance of the model, we plot the learnings using loss curve as a metric.



The LSTM model is learning to generate music accurately. The model can improve its accuracy over time, suggesting it can learn the complex relationships between the different musical elements.

The learning curve is also relatively smooth, suggesting that the model is not overfitting the training data. This is important because it means that the model will be able to generalize new data and generate music that is like the music in the training set.

Overall, the learning curve is a positive sign that the LSTM model is learning to generate music accurately. However, the model has not yet reached an accuracy of 1.0. This suggests that there is still room for improvement, and that the model may benefit from further training or additional regularization techniques.

Here are some specific inferences that can be drawn from the learning curve:

The model can learn to generate music accurately, as evidenced by the increasing accuracy over time.
The model is not overfitting the training data, as evidenced by the smooth learning curve.
The model may benefit from further training or additional regularization techniques to improve its accuracy.
It is also important to note that the accuracy metric used to generate the learning curve may not be the best way to measure the quality of generated music. Other metrics, such as perceptual loss or human evaluation, may be more informative.

The loss curve with a starting value of 4 that decreases with increasing number of epochs to a value of 0 indicates that the LSTM model for music generation is learning effectively. The model can reduce its loss over time, suggesting that it can learn the complex relationships between the different musical elements and generate music that is more like the training data.

The loss curve reaching a value of 0 suggests that the model can perfectly generate music identical to the training data. However, this is not realistic to expect in practice, as there is always some noise and variability in real-world data. Therefore, a more realistic goal is to achieve a loss value that is sufficiently low, such as below 0.1.

Overall, the loss curve is a positive sign that the LSTM model is learning to generate music effectively. However, it is important to note that the model has not yet reached a loss value of 0. This suggests that there is still room for improvement, and that the model may benefit from further training or additional regularization techniques.

Here are some specific inferences that can be drawn from the loss curve:

The model is learning effectively, as evidenced by the decreasing loss over time.
The model may benefit from further training or additional regularization techniques to reduce its loss to a more realistic value.
The model is still under development, but it has the potential to generate music that is of high quality.
It is also important to note that the loss function used to generate the loss curve may not be the best way to measure the quality of generated music. Other metrics, such as perceptual loss or human evaluation, may be more informative.

We provide a code snippet. The "generate notes" function is a key component in creative music generation using a trained neural network model. It initiates the process by selecting a random starting point from the input dataset. The input dataset, represented as `network input`, contains musical patterns or sequences. Additionally, the function relies on two vital inputs: "pitchnames," which is a list of note names or pitch classes, and "n_vocab," representing the number of unique notes in the dataset. These inputs play a crucial role in translating numerical predictions back into human-readable note names.

To facilitate this translation, the function employs a dictionary called `int_to_note`. This dictionary serves as a mapping between numerical indices and their corresponding note names. This mapping is used when converting the model's numerical output into a recognizable musical note.

The process unfolds through a loop designed to generate 500 notes. Within each iteration, the current "pattern" is reshaped to match the expected input shape for the neural network model. This pattern is also normalized by dividing each value by "n_vocab." Normalization ensures that the input data falls within the range expected by the model.

The neural network model is then utilized to predict the next note based on the current pattern. The prediction results in a probability distribution over note indices. The index with the highest probability is selected, and the corresponding note name is retrieved from the "int_to_note" dictionary. This generated note is added to the "prediction output" list.

After each prediction, the "pattern" is updated by removing the first element and appending the newly generated note's index to the end. This process ensures that the model considers the most recent notes when generating the next note, maintaining the sequence's musical coherence.

The "generate notes" function returns the "prediction output," a sequence of 500 generated notes. This function plays a fundamental role in the creative application of neural networks for music generation, enabling the generation of music that follows the learned patterns and styles from the training dataset. We convert this output back into a midi file format.

FUTURE WORK

Enhanced Model Architectures: Experiment with more complex neural network architectures, such as stacked LSTMs, bidirectional LSTMs, or even more advanced models like Transformer-based architectures. These can potentially capture longer-term dependencies and intricate patterns in music more effectively.

Hyperparameter Tuning: Systematically optimize hyperparameters, including the network architecture, learning rates, batch sizes, and the length of input sequences, to improve the quality of generated music.

Incorporate Lyrics: Extend the system to generate music with lyrics. This could involve creating a separate model for lyric generation and fusing it with the existing model for music notes.

Real-time Music Generation: Develop a system capable of real-time music generation, where the model can generate music in response to user input or coordinated with live performances.

Genre and Style Control: Implement controls that allow users to specify the genre or style of music they want to generate. This could involve conditioning the model on genre-related data during training.

Music Accompaniment: Expand the system to generate full musical compositions with different instrumental accompaniments. This could be achieved by training separate models for various instruments and combining their outputs.

User Interface: Create a user-friendly interface that enables musicians and composers to interact with the model more intuitively. This could include options for modifying generated music on-the-fly and providing feedback to the model.

Multi-modal Generation: Combine the music generation with other modalities, such as visual art or text, to create multimedia experiences.

Transfer Learning: Investigate the use of transfer learning techniques. You could pre-train on a large dataset of existing music and then fine-tune the model on a smaller dataset of a specific artist or style.

Interactivity and Collaboration: Develop collaborative features, allowing multiple users to interact and co-create music in real-time.

CONCLUSION

This project marks a significant step into the realm of automatic music generation through LSTM neural networks. LSTM's ability to capture sequential patterns has enabled the creation of unique musical compositions. Data preprocessing, model training, and hyperparameter tuning have all played pivotal roles in achieving our musical results. While there remain challenges, such as occasional incoherence in generated music, this project opens doors to future possibilities. Ethical considerations surrounding AI-generated music, the nature of artificial creativity, and the potential for real-time music generation are key areas for exploration. This project signifies the fusion of art and technology, demonstrating the vast potential of AI in the world of creative expression, particularly in the universal language of music.