Music Generation using LSTM

Pedro Ortego

February 14, 2023

Abstract

This paper presents a novel approach for improving music generation using a long short-term memory (LSTM) algorithm, and making this algorithm as efficient as possible. By training the LSTM on a dataset of musical sequences, we were able to generate new, original pieces of music that demonstrate the ability of the model to capture the structure and style of the input data. Our experiments provide new insights into the capabilities and limitations of LSTM-based music generation algorithms, and pave the way for further research in this exciting and rapidly-developing field.

1 Introduction

The use of artificial intelligence for music generation has the potential to revolutionize the way music is created and experienced. In this project We are trying to explore methods to generate musical notes such that all the notes are in key, flow logically from one note to the next, and synthesize them all together to generate a pleasant listening experience. Currently, research in this field is fairly undeveloped, and current applications have not yet broken ground to gain the attention of mainstream layman audiences.

This is in contrast to some other forms of artistic creation generated by AI. The most notable of these is visual artwork, where models such as DALL-E, Stable Diffusion, and Midjourney have now obtained widespread attention. Or even more recently, a new chatbot based on NLP called chatGPT.

In regards to music, there are two main ways that AI can be used to create something new: Raw audio synthesis and manipulation of MIDI data. Raw audio synthesis is the primitive generation of audio by training a model on various musical tracks. MIDI manipulation, however, focuses more on the process of creating notes and melodies, rather than raw audio itself. Our project will initially focus on the latter of the two and details of our methods are described below.

2 Background

Overall, there is not a lot of background knowledge required to understand the work, besides some basic music theory and the algorithm that we will be using.

2.1 LSTM applied to music generation

Long short-term memory (LSTM) is a type of recurrent neural network that is well-suited for modeling sequence data, such as musical compositions. LSTM networks are composed of LSTM cells, which have the ability to remember past information and use it to make predictions about future events. This is achieved through the use of gates within the cells, which control the flow of information and the updating of the cell's internal state.

In the context of music generation, LSTM networks can be trained on a dataset of musical sequences, such as with the use of MIDI files. During training, the network learns to predict the next event in a sequence, based on the events that have come before it. Once trained, the network can be used to generate novel musical compositions by starting with a seed sequence and using the learned model to predict subsequent events in the sequence. In this way, LSTM networks are able to generate music that exhibits the structural characteristics of the training data, while also introducing some degree of novelty and variation.

2.2 Music theory

We didn't dive too deep into the music theory during our project. In fact the only thing required to understand how our project works, is knowing the difference between a note and a chord. One key difference between notes and chords is that notes are typically played one at a time, while chords are played together. This means that notes are typically heard as individual pitches, while chords are heard as a combination of pitches. Additionally, chords often have a specific function in harmony, such as serving as a tonic chord or a dominant chord, whereas individual notes do not have a specific harmonic role.

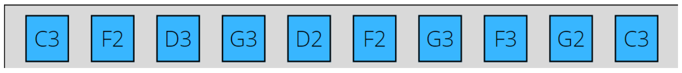
3 Method

3.1 The dataset

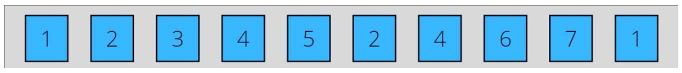
For this project, instead of using a pre-existing dataset, we decided to create our own. In this case we wanted to take multiple songs from the same composer, so that the songs have some coherence among them. For this project, we created a dataset taking a bunch of songs from Hans Zimmer (composer), and we imported these songs as .mid files. This dataset will only contain the sequence of notes and chords of all the songs. The total length of the dataset is approximately 15.000 chords and notes.

3.2 Data pre-processing

The first thing we will do is to get a list of strings containing the notes of the song in order. To make this task easier, we used the library Music21. After getting all the notes, we will create a funciton that maps each note to an integer, as we cannot train our model using strings. After mapping these notes to integers, we will create our X_{train} and Y_{train}. To see how this works we will show it with a simple example. Let's say we have the following list of notes:



After we map these notes, the result would be something like the following:



Then we will check the frequency of appearance of each note, and we will get rid of the ones that appear the least times, as we don't want these notes make the train harder or even show in the resulting song we will generate, as these might not sound as good.

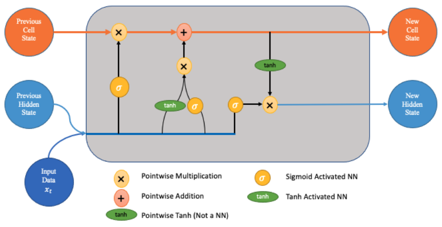
Once we have our data in this form, we can build the X_{train} and Y_{train}. For this part we will call the X: features and Y: target. For our model, the features will be the sequence of notes that we already generated, and the target will be the prediction for the next note. We created an animation on the jupyter notebook to show how it works. In the animation we are just taking 3 notes for the prediction, but in our pre-processing we will take 75.

Finally, we will reshape the features to get X, and normalize it. We will also apply OHE to the target using the toCategorical function, and we will get the y, so instead of having an integer as a target, we will have an array of zeros and a one in the position of the value of the target. With these we will train our model.

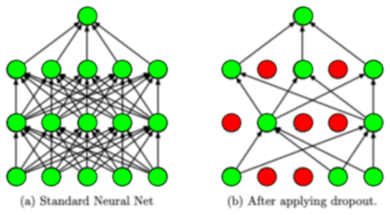
3.3 Building the model

We will create our model using a combination of the following layers.

- **LSTM layers:** The purpose of LSTM layers in a music generation model is to capture the structure and dependencies in the training data, allowing the model to generate novel and coherent musical sequences. LSTM layers are trained on a dataset of musical sequences and learn to predict the next event in a sequence based on the events that have come before it. Once trained, the LSTM layers can be used to generate novel music by starting with a seed sequence and using the learned model to predict subsequent events.



- **Dropout layers:** This is a regularization technique that prevents overfitting. In our case, in case of overfitting, we would be generating the same songs as the input. To prevent this to happen, this layer sets some of the values of the input to 0 at each update. The number of elements that are set to 0 depends on the parameter that we pass into the funciton to generate this layer.



- **Dense layers:** These are the layers of a fully connected neural network, where each of the input nodes are connected to the output nodes.
- **Activation layer:** This layer determines the activation function for our neural network, which will be used to calculate the output of a node. There are multiple activation functions, so we will try multiple ones to test which one works better.

3.4 Generate the song and display it

Using our trained model, now we can generate a song. For that, we will define a function that will take some random section of one of the input songs, and predict an specified number of notes. In our case, we decided to generate 250 notes, which is approximately 2 minutes of content.

In order to be able to display the music sheet of the generated song, as well as being able to generate a file with the mid file that plays the song, we used MuseScore.

4 Experimental analysis

4.1 Metrics Evaluation

To test if our model is working, we will use Categorical Cross-Entropy loss (also known as Softmax loss) to evaluate our model. Here is the formula for this loss:

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i$$

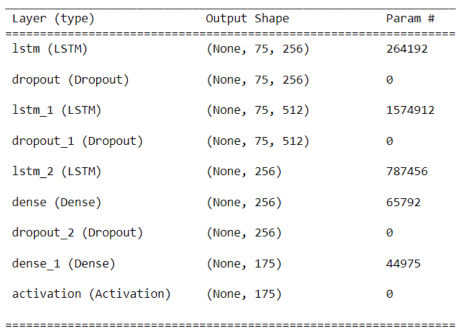
Where y is the true probability distribution over the classes, and $\hat{y}$ is the predicted probability distribution over the classes. We are using these loss function because in this case, we have a multi-class classification where the labels are one-hot encoded, and there is only one element in the target vector that is non-zero. This loss is calculated using the tensorflow library.

Additionally, we will also take into consideration some subjective metrics, which are going to be the following:

- **Coherence:** This measure evaluates how well the generated music follows the structural conventions of the training data.
- **Novelty:** This measure evaluates the degree to which the generated music is original and does not simply repeat the training data.
- **Listenability:** This measure evaluates how pleasant and enjoyable the generated music is to listen to. Listenability could be assessed through subjective evaluations.

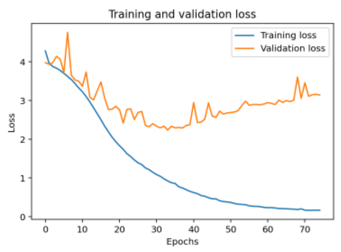
4.2 Hyperparameter tuning and analysis

For our milestone, we achieved some decent results using the following layers:



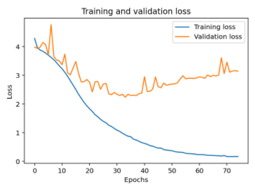
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 75, 256)           264192

 dropout (Dropout)           (None, 75, 256)           0

 lstm_1 (LSTM)               (None, 75, 512)           1574912

 dropout_1 (Dropout)         (None, 75, 512)           0

 lstm_2 (LSTM)               (None, 256)               787456

 dense (Dense)               (None, 256)               65792

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 175)               44975

 activation (Activation)     (None, 175)               0

=================================================================
```

So, for this project, we won't change the architecture of this model, but we will tune the different parameters to obtain better results. First, we start checking the results from the milestone. In this experiment we got a good loss function. However, the generated song was too similar to some parts of the original songs. To check if we were overfitting, we split the data, and check the validation loss. This is the loss function we got (using a value of 0.1 in the dropout layers).

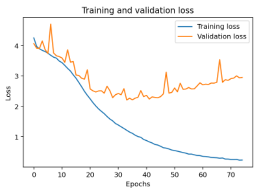


As we can see in the figure, we are clearly overfitting. Another additional problem we had during our milestone is the time it takes to train the model, which took over 8 hours. So to solve both problems, we will try changing the **value of the dropout layer**, and the **batch size** (as incrementing this last one will make the training faster, but less accurate). Here are the results we got:

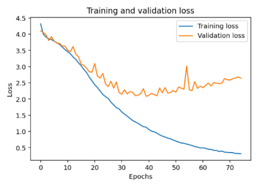
For **batch size = 128** the training took approximately 2 hours, and for **batch size = 256** it took about an hour and a half, which is a huge improvement from our previous result. We also see that the larger the value we use for the **Dropout layer**, the better results we get for validation.



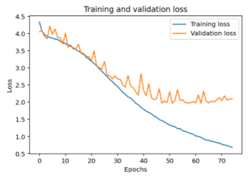
Batch size = 128, Dropout = 0.1



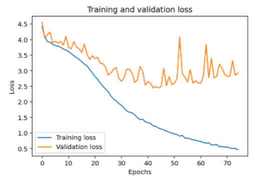
Batch size = 128, Dropout = 0.3



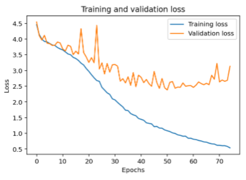
Batch size = 128, Dropout = 0.5



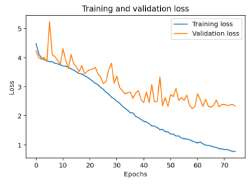
Batch size = 128, Dropout = 0.7



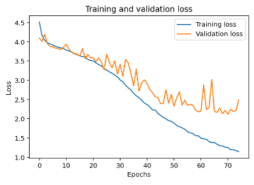
Batch size = 256, Dropout = 0.1



Batch size = 256, Dropout = 0.3



Batch size = 256, Dropout = 0.5



Batch size = 256, Dropout = 0.7

4.3 Analysis of results

Based on the previous results, we decided to use the **batch size of 256**, as it's the fastest one, and there is not a huge impact on accuracy.

For the dropout, we first decided to use 0.7, as it's the best reducing overfitting. However, when we generated the song with that model, we realized that it wasn't as good as before, and the "listenability" metric we defined earlier, was worse. We found that the best value for the **dropout layer was 0.5**. With these values we got the best results and in a much faster way.

We also tried reducing the overfitting using other techniques, such as early stop, and adding l1 or l2 regularization to the dense layers. For the first one, we got rid of the overfitting, but the music generated was pretty bad. For the second one, we couldn't get the loss to go below 3.75. So, we decided not to use these techniques.

5 Discussion and prior work

The experiments presented in this paper have shown that it is possible to generate musical sequences using a long short-term memory (LSTM) algorithm. By training the LSTM on a dataset of musical sequences, we were able to generate new, original pieces of music that demonstrate the ability of the model to capture the structure and style of the input data.

These results are in line with previous work on music generation using LSTM and other machine learning techniques. But we added upon these previous works in the importance of the values of batch size and dropout layers, and how these impact the performance and efficiency of the model.

6 Conclusion

In conclusion, this paper has presented an approach for generating music using a long short-term memory (LSTM) algorithm. By training the LSTM on a dataset of musical sequences, we were able to generate new, original pieces of music that demonstrate the ability of the model to capture the structure and style of the input data. While the generated music is still somewhat limited in terms of its diversity and complexity, this work represents an important step towards the goal of developing more sophisticated music generation algorithms based on LSTM and other machine learning techniques. In terms of future work, one promising direction would be to combine LSTM with other machine learning techniques, such as generative adversarial networks (GANs) or variational autoencoders (VAEs), to improve the model's ability to generate more diverse and complex music.

References

[1] Long Short-Term Memory by Sepp Hochreiter and Jurgen Schmidhuber. http://www.bioinf.jku.at/publications/older/2604.pdf

[2] Music generation with LSTMs by Bharath K. https://blog.paperspace.com/music-generation-with-lstms/

[3] Understanding Categorical Cross-Entropy Loss by Raul Gomez. https://gombru.github.io/2018/05/23/crossentropyloss/

[4] How to Diagnose Overfitting and Underfitting of LSTM Models by Jason Brownlee. https://machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models/

A Appendix

Jupyter notebook link, with more in depth explanation and all the code, as well as the animation mentioned on point 3.2:

`https://colab.research.google.com/drive/1THLRGsQQI4nF3aWHdn7etBdZRwzL5YZW?usp=sharing`

Link to the generated song:

`https://drive.google.com/file/d/1R9Ngian7nlQzimLE_CPMc_RYK-shDSRM/view?usp=sharing`

Image of the loss graph using early stop, batch size = 256, and Dropout = 0.5:

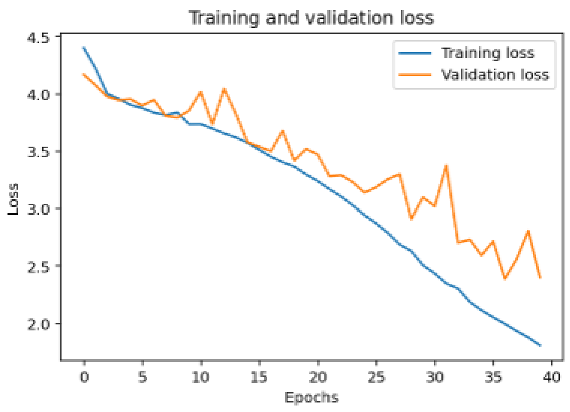


Image of a section of the music sheet from the generated song: