EDM Melody Generator

Deliver

Christian Pratellesi (12118532)

2023-01-14

Project Description and Approach

With the increasing availability of powerful computers on the private market and the diffusion of Digital Audio Workstations (DAWs), it has become straightforward to produce a song from the comfort of our beds. More and more people are trying their luck by producing and releasing songs, dreaming of becoming famous music producers. However, making a song is a complex process, and producing a hit song that will "catch the ears" of millions of listeners is challenging. One of the most critical elements of a memorable song, especially in Electronic Dance Music (EDM), is a catchy melody, and frequently artists struggle with coming up with one.

This project aims to create an application that facilitates the melody creation process by using deep learning methodologies to automatically generate melodies that can be used as they are or serve as a starting point to spark the imagination and creativity of artists. In particular, this project type is "**Bring your own data**", and the main focus is on collecting a dataset of EDM melodies and using them to train an already existing deep learning model to generate similar melodies.

The approach that this project uses are Recurrent Neural Networks (RNN) which have proven to be efficient in generating melodies, as we can see in various publications presented at the International Society for Music Information Retrieval (ISMIR) conference. Among the publications, there are melody generation models like VirtuosoNet [1], StructureNet [2], and other architectures based on RNNs [5]. RNNs are suited because they can learn the relationship each note has to the other notes being played and can use that information to generate notes based on notes that have been played previously in time. Specifically, for this project, we use the collected dataset to train MelodyRNN [3], a recurrent neural network designed by Google's Magenta project to generate monophonic melodies.

Dataset

Having an excellent and ample dataset is of really high importance. In virtually any deep learning application, the amount and quality of data can be the difference between a great and a poor result.

For genres like classical and pop music, big datasets already exist; an example could be the POP909 dataset [4], a collection of 909 pop piano performances by various professional pianists. By using this dataset, it is possible to train an RNN to generate pop melodies

that are to the human ear as pleasing as melodies handcrafted by professional musicians. However, when it comes to EDM, a suitable dataset still needs to be created.

Therefore, the main focus of this project is on collecting a suitable dataset to train a deep-learning model for EDM melody generation. The dataset contains monophonic melodies (not more than one note played at the same time) created by EDM artists (e.g., Avicii, Kygo, ...) saved in MIDI format (.mid), which is the standard format when it comes to storing musical information on electronic devices. In particular, MIDI does not store audio or any information about the sound being reproduced. Instead, it stores the pitch, start time, stop time, and other properties of each note being played and is used as a musical data format by many deep-learning frameworks.

Dataset Collection

For this project, mainly three sources of training data generation are used:

- Generating my melodies from scratch
- Using existing MIDI files
- Recreating famous melodies from scratch

After an initial assessment, it emerged that recreating famous melodies from scratch would take about 15m per melody, so I expected to create about 160 training samples in 40h of work. I added sample melodies from songs I created myself to augment the dataset. Additionally, I used existing MIDI files from the internet to speed up the process and collect even more data. The existing MIDI files had to be cleaned up and processed, but they turned out to be a more efficient way of creating data which helped me collect more than 200 samples in 25h, almost 2x faster than the original expectation. Finally, it is worth noting that all the melodies in the dataset are monophonic (not more than one note playing at the same time). They are named according to the convention "Artist - Title.mid". They are all labeled with original Beats Per Minute (BPM) and Key. All notes are set to a velocity value of 80.

Model Training

Two metrics have been used to evaluate this project: accuracy and perplexity. Both metrics perform a quantitative evaluation of the trained model's performance. On the one hand, accuracy measures how well a model predicts samples. On the other hand, perplexity measures the probability model's uncertainty in predicting a sample. The higher the accuracy and the lower the perplexity of the model, the better it performs.

After running the first training, the goal for accuracy was set to values above 0.9, and the goal for perplexity was set to values below 1.3. In total, five training runs have been executed, each for 20.000 steps, and they are summarized in Table 1 below.

Run	Training Time	Batch Size	Learning Rate	Model Size	Accuracy	Perplexity
1	1h24m	64	10e-3	64	0.9034	1.329
2	1h25m	32	10e-3	64	0.9055	1.313
3	1h24m	32	10e-4	128	0.8254	1.729
4	1h26m	32	10e-3	128	0.9862	1.045
5	1h27m	64	10e-3	128	0.9918	1.044

Table 1: Training Runs

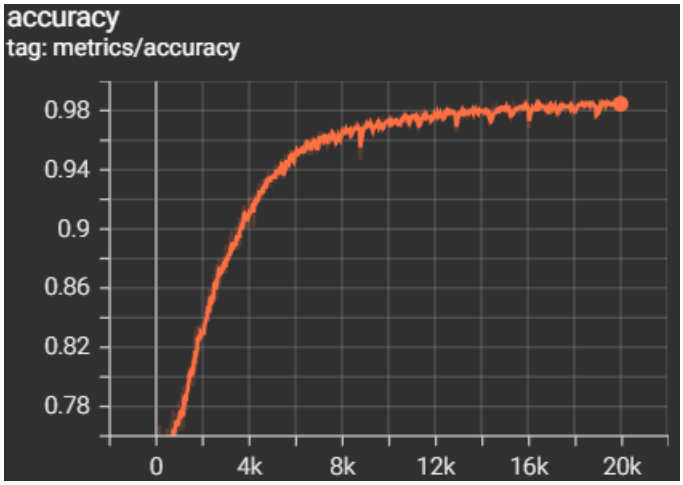

Figure 1: This figure shows the plot of accuracy over the training steps for the fifth and last run (adapted from TensorBoard).

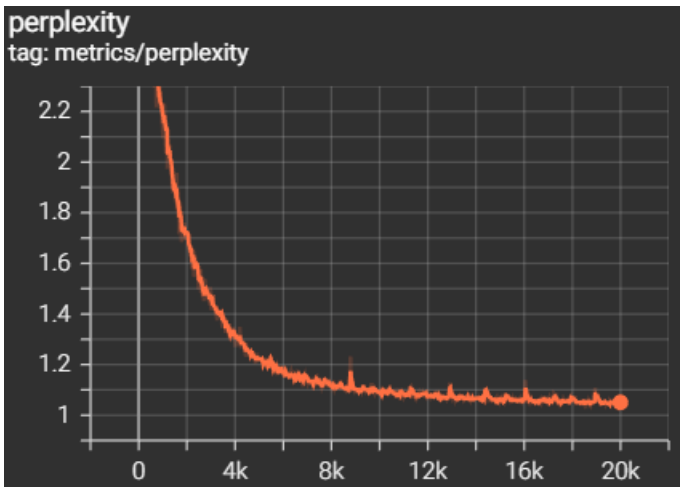

Figure 2: This figure shows the plot of perplexity over the training steps for the fifth and last run (adapted from TensorBoard).

As shown in Table 1, all the runs are equal in training times; they only differ by an insignificant amount of time. So, changing the model and batch sizes does not affect the overall training time of the network. It is also worth noting that the run with the lowest learning rate is the one that performed the worst regarding both accuracy and perplexity. Finally, with equal learning rate and model size, the model that performed best was the one with the bigger batch size, corresponding to the fifth run. Figure 1 and Figure 2 show the plots of the evaluation metrics during the training procedure for the fifth run (best-performing one).

So, a higher learning rate is preferred to a lower one since it yields higher accuracy values and lower perplexity scores. As for the model size, having a bigger model size is preferred since it yields better results than models having a smaller size, but if we do not have enough computing power, it is necessary to reduce the size of our model. Regarding batch size, considering the same learning rate and model size, a bigger batch size yields slightly better results.

Insight and Takeaways

One of the main takeaways I gained from this project was that it is essential to set up the whole project in the beginning before working on the data collection. Researching the best methodology and finding a great implementation, together with setting up the entire environment, took approximately 8 hours of work. It was a challenging task, but doing it all in the beginning, made it possible to focus solely on data collection in the following stage of the project. Instead of having to figure out how to run Google's MelodyRNN in the middle of my project, I was able to collect new data along the way and plug it into my project, which made it easier and less time-consuming to work on it.

Another insight I gained from working on this project is that data collection is time-consuming. Therefore, finding a fast and easy way to collect the data in the shortest time possible is essential. At first, I thought I could recreate all the melodies I wanted to use by hand, but I quickly realized that that was not doable, especially if I wanted to create a reasonably extensive dataset. Therefore, I relied on other methods, such as including melodies I made for previous musical projects or using MIDI files present on the internet. Regarding the latter approach, all the MIDI files were polyphonic and contained harmonization along with the melody of a song. Therefore, I had to clean up the files so that they matched the input requirements of the model I was using. However, this resulted in being faster than recreating all the melodies myself from scratch.

Musical data generation and annotation work very well with Logic Pro X, a DAW developed by Apple. When exporting a MIDI file, information about the original BPM and the melody's key are stored automatically, making it easy to annotate additional information within the MIDI files. Since I was already used to working with Logic Pro X, generating, importing, and cleaning up data was straightforward, as well as exporting it. However, it remains a time-consuming task to collect a large number of samples.

While training Google Magenta's MelodyRNN for this project, I tried various hyperparameter configurations before reaching the goal that I set. From all the training runs that I made, it emerged that, on my data, using a larger learning rate works worse than using a smaller one. In particular, when training the model for 20.000 runs, using a learning rate of 10^{-3} yielded an accuracy of 82%, while using a learning rate of 10^{-4} yielded accuracies bigger than 90%. Furthermore, using a larger batch size and a larger model size improved the result without affecting the training time of the model.

Lastly, the results that this project produced were very satisfactory. However, if I had to do the same task again, one thing that I would do differently is data collection. Initially, I thought I could recreate all the melodies I wanted to use for training by hand in a DAW. However, this proved to be time-consuming, and collecting an extensive dataset in a short time was not possible with this approach. Therefore, I switched my approach in the middle of the project, which helped speed up the data collection process. So, if I could do this project again, I would directly start the data collection process by crawling the internet and collecting existing MIDI files, even if they need to be cleaned up. Doing so would allow me to collect a more considerable amount of data in a shorter time.

Timeline

It took me to complete this project approximately 79 hours, without considering the lectures. Compared to the expectations that I first set at the beginning of the project, there are some parts that I underestimated and some that I overestimated. One of the

parts that I overestimated was the time it would take me to train a well-performing model. I expected it would take me about 4 to 5 hours to train a decent model, but the time I needed was much less. In about 8 hours of training, I could experiment with various hyperparameters configurations and train five different models. On the other hand, one of the parts that I underestimated was building an intuitive web application. To build my web application, I used Streamlit, which made it straightforward, so developing the application took little time. However, I encountered many problems with figuring out how to include the trained model within my application and also problems when deploying the application on the web with all the python packages/environment dependencies. Considering all these issues, developing the final web application took about 28 hours, almost double the time I initially estimated.

Table 2 shows the approximate expected timeline of all the project stages, the expected workload in hours, and the actual workload:

Stage	Expected Workload	Actual Workload
Dataset Collection	40h	25h
Network Building	5h	5h
Network Training	20h	8h
Application Development	15h	28h
Final Written Report	5h	2.5h
Final Presentation	2h	2h

Table 2: Stages

References

[1] Dasaem Jeong, Taegyun Kwon, Yoojin Kim, Kyogu Lee, and Juhan Nam. Virtuosonet: A hierarchical rnn-based system for modeling expressive piano performance. In *ISMIR*, 2019.

[2] Gabriele Medeot, Srikanth Cherla, Katerina Kosta, Matt McVicar, Samer Abdallah, Marco Selvi, Ed Newton-Rex, and Kevin Webster. Structurenet: Inducing structure in generated melodies. In *ISMIR*, pages 725–731, 2018.

[3] Elliot Waite. Generating long-term structure in songs and stories. https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn, 2016.

[4] Ziyu Wang, Ke Chen, Junyan Jiang, Yiyi Zhang, Maoran Xu, Shuqi Dai, Xianbin Gu, and Gus Xia. Pop909: A pop-song dataset for music arrangement generation. *arXiv preprint arXiv:2008.07142*, 2020.

[5] Jian Wu, Changran Hu, Yulong Wang, Xiaolin Hu, and Jun Zhu. A hierarchical recurrent neural network for symbolic melody generation. *IEEE Transactions on Cybernetics*, 50(6):2749–2757, 2020.