AAA project: Music Generation using different Deep Learning (DL) methods

Jonathan Gehmayr 57267

May 2022

1 Introduction

AI is revolutionizing a lot of different industries. One of them is the music industry. Producing music is strongly supported by diverse Deep Learing (DL) models that help musicians mixing and mastering music tracks. Not only musicians make us of AI, aslo streaming platforms like Spotify leverage the flexibility of AI, by predicting properties of songs like their happiness or sadness, but also to recommend their users songs that are similar to those they already liked [1]. In this project the focus is on the generation of music. Concretely, the project is focused on (1.) extending music tracks using recurrent networks (RNN) with long short term memory cells (LSTM) and (2.) to generate complete new tracks by applying variational autoencoders (VAE). The data the models receive as input is extracted from MIDI files containing classical piano music.
Other approaches like the one of Tham and Kim [2] use LSTMs, VAEs, generative adversarial networks (GAN) and convolutional neural networks (CNN) to produce not only single instrument music, but also multi-track music and gave big inspiration for this project. Skuli [3] applied LSTMs in the same fashion as to produce language models and generated single instrument piano music from one of his video games. Using LSTMs for music generation was already leveraged multiple times like also by Nelson [4], who produced lo-fi music. Yang [5] applied deep convolutional generative adversarial networks (DCGAN) and launched MidiNet, which is capable of producing multi instrument music. Dong [6] uses GANs as well and outputs synthetic multi instrument tracks, that are taking into account dependencies between different instruments. All of these approaches have in common to represent a music track as a sequence of notes that form a melody, but neglect the duration of a single note, but also the varying pauses between notes. As these properties are essential to music, these shortcomings were especially addressed in this project. The formed approach is presented in the next section.

2 Approach

Goal of the project is to firstly extend a song by giving a start sequence and secondly to generate whole new songs. Music is a complex datastructure with a lot of inter dependencies. Therefore, as a starting point only single instrument music was considered to reduce the data complexity. As previous approaches utilized datasets of MIDI files with piano music, this was adopted to this project as well. MIDI files have the advantage of being very light-weight and standardize the representation of music to be used by different programs [7]. Some of the more popular datasets containing single instrument piano music in MIDI format are among others the MAESTRO dataset [8] and the Lakh Pianoroll dataset [9]. Further research lead to the Classical Music MIDI dataset [10], which had the advantage of being really small (only $\sim$ 7MB) and

seemed therefore as a good starting point. Supplying the dataset as input for deep learning models, requires the transformation from the MIDI file format into an array with samples and extracted features. Other approaches neglected in this step the timing information of the MIDI files. Other than those, in this project a MIDI file is not mapped to a sequence of notes but to an 2D array, while rows represent the timeline and columns the notes and the value of a cell is the power of the note. Stacking those 2D arrays forms the dataset, which is used to feed the models. By using a LSTM model the goal is to learn the time dependent multivariate signal of piano music and to extend given music from a certain starting point. VAEs on the other side are able to generate new data by learning a multivariate Gaussian distribution of the input data. Feeding a random latent vector into the decoder of the VAE results in the generation of the new data. This is leveraged by training a VAE with before mentioned data.

3 Implementation

3.1 Feature extraction

Training neural networks requires input in array format. Therefore a way needed to be found to convert a MIDI file into a suitable format. The conversion from MIDI File format into 2D Numpy [11] arrays was inspired by an article [12] and works like following. Loading and working with MIDI files in Python is supported already though some libraries like Music21 [13] or Mido [14], whereas Mido was used in this project. Loading a MIDI file with Mido results in a midi-object, that contains different channels, whereas a channel usually represents a single instrument. A channel contains some meta information like author, or speed of the track, timing information in *control change* messages or actual notes with their duration. Message objects can be easily converted into *dictionaries*, which makes the transition to pure Python. The duration of a note is defined by ticks and beats, while a beat consists of several ticks. The timing of notes are defined as delta time, which tells how many ticks have passed since the last message [14]. Therefore to create an array from the loaded MIDI file, the rows represent the time and the columns the notes (0-127). One row has the value of one tick. Thus if a note is on for 5 ticks, 5 rows within a note column will have the same value. Mido messages contain a *velocity* property, which seems to be the the velocity of the song at first glance, but does actually encode the power of a note. This value is stored as cell value of the MIDI array and has a range of (0-127). Tracks do not have a standardized length, which implies that the generated array needs to be resized. For that OpenCV [15] is used, as with this module control over the interpolation is given. At that point the arrays have a fixed number of columns (128), but no fixed number of rows. The challenge here is to not loose to much information in the rescaling, but also to keep the data compact. Therefore, different datasets were produced, which differ from each other through the array size of a single song, resulting in four datasets with array sizes (256,256), (512,512), (1024,1024) and (1024,128). Neural networks need value ranges 0-1, so the datasets were additionally rescaled; and as to restrict bias during training they were shuffeld as well.

3.2 Song extension with LSTM

First challenge of this project is to learn the time dependent structure of songs and to further extend a given song. LSTMs are used for different forecasting problems and seem to be a good fit for tackling that challenge. As library for building the model Tensorflow [16] was used.

3.2.1 Model architecture

As it was unknown, which architecture to use for the model it was inferred by a hyperparamter optimization. Searching for optimal hyperparamter requires defining an input space which is listed in Table 1. The method

for hyperparameter optimization was Bayesian optimization, while all different combinations were tried. The models wered trained for 100 epochs on a single track to reduce time consumption.

Table 1: Hyperparameter search space for LSTM model

Parameter	Input range	Step	Best
number of layers (n_{layers})	1-3	1	2
number of units (n_neurons)	128-1024	128	768

The outcome of this search was that two layers with 768 neurons resulted in the smallest loss. As the model was only trained for 100 epochs it was assumed that for higher numbers of units the number of epochs could have been to small. So, as the complexity of music data is also high the actual number of neurons was set to 1024 in the final model with 2 layers. Additionally a drop out layer with a drop-out-rate of 0.2 was added to the architecture to reduce over-fitting. As the output should be of size 128 the last layer is dense with shape (1,128). The whole architecture is listed in the Appendix A.1

3.2.2 Model training

Training the model required generation of input sequences and output sequences. For this the dataset with array size (1024,128) was used. As the first axis represents the timeline of a song it was used to generate subsequences with a lag of 256, having therefore a size of (256,128) plus their dependent output which is the following timestep with size of (1,128). A single track produces trainings data of size (768,256,128), meaning we can split a track into 768 subsequence each having 256 timesteps and 128 features. From that the batch size for training is defined to be equal to the size of a track, so it was set to 768. Applying this procedure over the whole dataset with 295 songs, would result thereby in an array with memory allocation of roughly 50GB. To avoid this, a generator function was implemented that yields a single batch when it is called. Generators decrease the memory utilization drastically, but trade off training time by that, which is a common issue [17]. Experimenting with generators resulted in the conclusion that the trainings process would take too long, so it was switched back to standard training, but with limited dataset. Hence, the LSTM model was only trained using a single track for 500 epochs.

3.2.3 Model evaluation

Evaluating the model needs to determine how well the model adapts to the music input. This is not a trivial process as the quality of music is not quantified and depends strongly on the consumer. Nevertheless, in this approach it is only measured how well the model adapts to the given trainings data, but not how well it generalized its forecast abilities on an unseen music track. So, to validate that the training was successful the model is fed with a training sequence to prolong it. To measure the error of the predicted output the mean absolute error is used, by comparing the true sequence with the predicted one. This trainings validation is visually supported by plotting the actual song overlayed with the predictions.

3.3 New music generation with VAE

The second challenge of this project is to generate new music from the dataset used as input, entailing that a generative model must be applied. VAEs have proved themself to be successful in augmenting image data, which inspired the development of this task. As a starting point tutorials [18, 19, 20] considering the MNSIT dataset [21], were used to replicate their results.

3.3.1 Model architecture

After successful implementing first VAE architectures that process (32,32) images, the models were adapted to handle bigger formats needed in this project. The final architecture used in this project consists for the encoder of three blocks, each having a 2D convolutional layer, a max pooling layer and a batchnormalization layer.These three blocks are followed by a flattening layer and a dense layer. The last dense layer serves as input for the *mean dense* and the *log variance dense* layer, which represent the learnt Gaussian distribution $\mathcal{N}(z_\mu, z_\sigma)$. The dimensionlity of the output of these two layers namely vector z_μ and z_σ^2 was set to be $imagesize/2$, so if the model is trained with the (512,512) dataset the dimensionality of these vectors is 256. z_μ and z_σ^2 serve as input for a custom *sampling layer*, which implements the for VAE introduced Reparameterization Trick. The Reparameterization Trick allows the sampling layer to be deterministic, by sampling the latent vector z by $z = z_\mu + z_\sigma^2 \odot \epsilon$, where ϵ is sampled from a Gaussian distribution $\mathcal{N}(0, 1)$. This allows that the parameters $z = z_\mu + z_\sigma$ can be learnt by applying gradients with backpropagation through making the *sampling layer* deterministic [19]. The decoder architecture is the inverse of the encoder architecture and uses as input the sampled vector z being the output of the *sampling layer*. After that a dense layer and a reshape layer is added to scale up the latent vector for the following convolution. The rest of the decoder architecture consists of three blocks each having a upsampling layer, a transposed convolution layer and a batch normalization layer. The output of the model has the same dimensions as the input of the encoder. The whole architecture is listed in the Appendix B.

3.3.2 Model training

To train a VAE the objective is to minimize the $ELBO$ loss, which consists of a term defining the reconstruction loss and a second term defining the divergence of the learnt distribution to a Gaussian distribution. For the former mean squared error (MSE) is used while for the latter Kullback-Leibler divergence (KL). The whole loss is constructed like following [19]:

$$ELBO = ReconstructionLoss + DistributionDivergence$$
$$ELBO = MSE + KL$$
$$MSE = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2$$
$$KL = -0.5 \cdot \sum_{i=1}^{N}(1 + log(z_\sigma^2) - z_\mu^2 - e^{log(z_\sigma^2)})$$

The VAE was implemented as a class receiving encoder and decoder layers for construction [20]. To train the VAE model a custom train step is defined in order to get the outputs z_μ, z_σ^2 and apply the loss function $ELBO$. The VAE class contains as well custom functions for generating new tracks by feeding the decoder with random numbers and to encode decode a given track, which is mainly used for testing if the encoding and decoding is working. The VAE is trained with the whole dataset without any train test split as new data is generated by just supplying a randomly sampled vector z. The model is firstly trained on the dataset representing tracks with size (256,256) and then trained with the dataset having (512,512) arrays. During testing with the MNIST dataset the model converged very nicely using 30 epochs, but for bigger sized inputs it was unsure if the same effect could be achieved, so the model was trained for [30,50,100,200] epochs.

3.3.3 Model evaluation

Concluding if the music generation is successful in this part of the project is not trivial. Borji [22] discusses several metrics for the evaluation of GANs, which serve for the same task of generating new data, but concludes that dependent on the domain subjective evaluation might be the most appropriate choice. Hence, for this task the generated array of the VAE is plotted and inspected concerning to musical patterns. If the results *"look promising"*, the array is converted back into MIDI format.

4 Results

4.1 Song extension with LSTM

Evaluating the LSTM on a sequence of the training data set results in the output displayed in Figure 1, with MAE of 0.555. In the Figure we can see the actual music track in shades of purple,green and blue, while the output of the model is plotted in red. As the the forecasted sequence looks very distinct from the song input, it is concluded that the model is not capable of even learning a single track. Nevertheless, we see that some dots in the middle of the Figure try to imitate some of the previously occurring notes in the song.

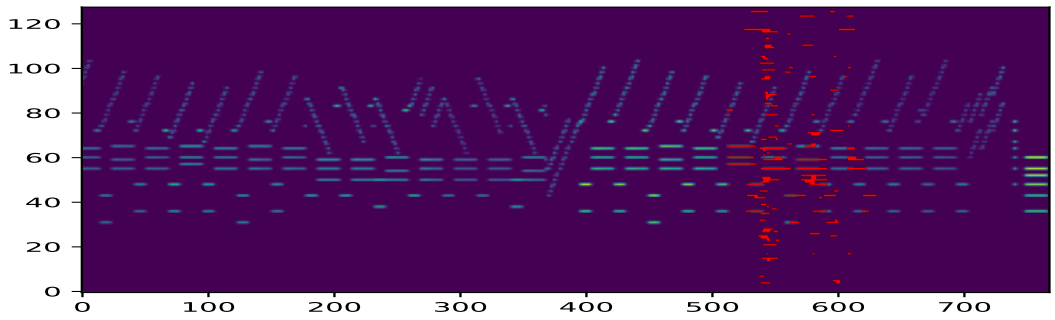

Figure 1: Music track in shades of purple,blue and green, with the predicted extension of the LSTM in red.

4.2 New music generation with VAE

Plotting of the outputs of the VAE showed that it does not generate any new music, given any random input vector z for the decoder of the VAE. The same results are obtained by training the VAE on the dataset with array size of (256,256). As the resulting plots would not add any information, they are ommited.

5 Discussion

5.1 Song extension with LSTM

The approach of extending a given song did not lead to satisfying results, as seen in Section 4. This could have various reasons. Firstly, the model could have been trained for not enough epochs or with a wrong learning rate to fully converge. Secondly, the hyperparameters like number of timesteps to use for one sample and number of output timesteps could be set not appropriately. These have been determined by inspecting plotted tracks and using an amount of timesteps that seems to encompass enough timesteps to

reflect the pattern of the music. Furthermore, the architecture of the model was half way assumed after the hyperparamter search, but it could be further optimized with a broader spectrum for hyperparameters. To solve this issue further experimentation, that leads to further insights would be neccessary. After having good preliminary results, the training of the model needs to be adapted by splitting the dataset into a train- and test-set. If that model is trained properly on the train-set it should be evaluated on the unseen test-set.

5.2 New music generation with VAE

Likewise, the VAE did not produce usable results. Training of VAE is prone to posterior collapse, in which the model input's information is not porperly learned by z_μ and z_σ, so that the output of the model is a generic that average the training data [23]. During preliminary tests with the MNIST dataset this issue could be witnessed randomly during some training initiations, but was minimized integrating max pooling layers in the model architecture according to [24]. As the array representation of music tracks is very sparse, the actual signal that could be extracted from it is quite small compared to the whole data size, which might be a reason why this approach was not working as intended. Same as for the the task of *song extension with LSTMs*, this approach needs more experimentation to produce valueable results.

In conclusion, the task of music generation is not a trivial one as the data is complex and interdependent. The approach of picturing MIDI files as 2D arrays achieved to incorporate the timing information of notes and pauses in music, but also increased the complexity of the training data compared to other approaches extracting only the melody and neglecting the timings. Furthermore, valuable knowledge was gained in designing the models using Tensorflow. Especially the design of the VAE was challenging as no out-of-the-box model was available. Through this project, an approach was formed for the tasks of extending songs and generating new music, which needs more experimentation to receive results that show the validity of this work.

References

[1] B. Clark, "How AI May Transform the Music Industry for Better or Worse," Aug. 2020. [Online]. Available: https://www.musicianwave.com/ai-music/

[2] I. Tham, "Generating Music using Deep Learning," Aug. 2021. [Online]. Available: https://towardsdatascience.com/generating-music-using-deep-learning-cb5843a9d55e

[3] S. Skúli, "How to Generate Music using a LSTM Neural Network in Keras," Dec. 2017. [Online]. Available: https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5

[4] "How I Built A Lo-fi Hip-Hop Music Generator | by Zachary | Artificial Intelligence in Plain English." [Online]. Available: https://ai.plainenglish.io/building-a-lo-fi-hip-hop-generator-e24a005d0144

[5] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, "MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation," *arXiv:1703.10847 [cs]*, Jul. 2017, arXiv: 1703.10847. [Online]. Available: http://arxiv.org/abs/1703.10847

[6] 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," *arXiv:1709.06298 [cs, eess, stat]*, Nov. 2017, arXiv: 1709.06298. [Online]. Available: http://arxiv.org/abs/1709.06298

[7] "Standard MIDI file format, updated." [Online]. Available: http://www.music.mcgill.ca/ ich/classes/mumt306/StandardMIDIfileformat.html

[8] "The MAESTRO Dataset." [Online]. Available: https://magenta.tensorflow.org/datasets/maestro

[9] "The Lakh MIDI Dataset v0.1." [Online]. Available: https://colinraffel.com/projects/lmd/

[10] "Classical Music MIDI." [Online]. Available: https://www.kaggle.com/soumikrakshit/classical-music-midi

[11] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[12] "Convert midi file to numpy array (Piano Roll) | by Huangwei Wieniawska | Analytics Vidhya | Medium." [Online]. Available: https://medium.com/analytics-vidhya/convert-midi-file-to-numpy-array-in-python-7d00531890c

[13] "music21 Documentation — music21 Documentation." [Online]. Available: https://web.mit.edu/music21/doc/

[14] "Mido - MIDI Objects for Python — Mido 1.2.10 documentation." [Online]. Available: https://mido.readthedocs.io/en/latest/

[15] "Home." [Online]. Available: https://opencv.org/

[16] "TensorFlow." [Online]. Available: https://www.tensorflow.org/

[17] D. Mazza, "Are Tensorflow/Keras generators too slow for you?" Sep. 2021. [Online]. Available: https://medium.com/@dantemazza1/are-tensorflow-keras-generators-too-slow-for-you-fae289cfe5ac

[18] "Variational Autoencoders as Generative Models with Keras | by Kartik Chaudhary | Towards Data Science." [Online]. Available: https://towardsdatascience.com/variational-autoencoders-as-generative-models-with-keras-e0c79415a7eb

[19] "Variational Autoencoder in TensorFlow (Python Code)," Apr. 2021. [Online]. Available: https://learnopencv.com/variational-autoencoder-in-tensorflow/

[20] K. Team, "Keras documentation: Variational AutoEncoder." [Online]. Available: https://keras.io/examples/generative/vae/

[21] "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges." [Online]. Available: http://yann.lecun.com/exdb/mnist/

[22] A. Borji, "Pros and Cons of GAN Evaluation Measures," arXiv, Tech. Rep. arXiv:1802.03446, Oct. 2018, arXiv:1802.03446 [cs] type: article. [Online]. Available: http://arxiv.org/abs/1802.03446

[23] A. Goyal, A. Sordoni, M.-A. Côté, N. R. Ke, and Y. Bengio, "Z-Forcing: Training Stochastic Recurrent Networks," arXiv, Tech. Rep. arXiv:1711.05411, Nov. 2017, arXiv:1711.05411 [cs, stat] type: article. [Online]. Available: http://arxiv.org/abs/1711.05411

[24] "Preventing Posterior Collapse in Sequence VAEs with Pooling," Nov. 2019. [Online]. Available: https://deepai.org/publication/preventing-posterior-collapse-in-sequence-vaes-with-pooling

A Model architectures

A.1 LSTM model architecture

Model: "LSTM"

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 256, 1024)	4722688
dropout_1 (Dropout)	(None, 256, 1024)	0
lstm_2 (LSTM)	(None, 1024)	8392704
dense_1 (Dense)	(None, 128)	131200

B VAE model architecture

Model: "Encoder"

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	[(None, 512, 512, 1)	0	
conv_1 (Conv2D)	(None, 508, 508, 64)	1664	encoder_input[0][0]
pooling_1 (MaxPooling2D)	(None, 254, 254, 64)	0	conv_1[0][0]
batch_norm_1 (BatchNormalizatio	(None, 254, 254, 64)	256	pooling_1[0][0]
conv_2 (Conv2D)	(None, 252, 252, 64)	36928	batch_norm_1[0][0]
pooling_2 (MaxPooling2D)	(None, 126, 126, 64)	0	conv_2[0][0]
batch_norm_2 (BatchNormalizatio	(None, 126, 126, 64)	256	pooling_2[0][0]
conv_3 (Conv2D)	(None, 124, 124, 32)	18464	batch_norm_2[0][0]
pooling_3 (MaxPooling2D)	(None, 62, 62, 32)	0	conv_3[0][0]
batch_norm_3 (BatchNormalizatio	(None, 62, 62, 32)	128	pooling_3[0][0]
flatten (Flatten)	(None, 123008)	0	batch_norm_3[0][0]
dense_1 (Dense)	(None, 512)	62980608	flatten[0][0]
z_mean (Dense)	(None, 256)	131328	dense_1[0][0]

```
__________________________________________________________________________________________
z_log_var (Dense)               (None, 256)          131328      dense_1[0][0]
__________________________________________________________________________________________
sampling_output (Lambda)        (None, 256)          0           z_mean[0][0]
                                                                 z_log_var[0][0]
==========================================================================================


_________________________________________________________________
Model: "Decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
decoder_input (InputLayer)   [(None, 256)]             0
_________________________________________________________________
dense_1 (Dense)              (None, 123008)            31613056
_________________________________________________________________
reshape_1 (Reshape)          (None, 62, 62, 32)        0
_________________________________________________________________
upsampling_1 (UpSampling2D)  (None, 124, 124, 32)      0
_________________________________________________________________
convtrans_1 (Conv2DTranspose (None, 126, 126, 64)      18496
_________________________________________________________________
batch_norm_1 (BatchNormaliza (None, 126, 126, 64)      256
_________________________________________________________________
upsampling_2 (UpSampling2D)  (None, 252, 252, 64)      0
_________________________________________________________________
convtrans_2 (Conv2DTranspose (None, 254, 254, 64)      36928
_________________________________________________________________
batch_norm_2 (BatchNormaliza (None, 254, 254, 64)      256
_________________________________________________________________
upsampling_3 (UpSampling2D)  (None, 508, 508, 64)      0
_________________________________________________________________
convtrans_4 (Conv2DTranspose (None, 512, 512, 1)       1601
=================================================================
```