TDDE19 - AI and Machine Learning

Music Generation Using Machine Learning

Autoencoders and Long Short-Term Memory

Yohan Ayoub, Edvin Bergström, Edwin Forsberg

yohay608, edvbe696, edwfo764

Carl Harris, Rolf Kargén, Leon Li Persson

carha789, rolka274, leope892

Department of Computer and Information Science

Linköping University

December 20 2021

Contents

1 Introduction

Music is an integral part of human culture and for at least 40 000 years mankind have found numerous ways of constructing tools for the sole purpose of producing music. In our age of emerging artificial intelligence (AI) and machine learning (ML) technologies, it is perhaps not surprising that these tools would be used for creating music too.

While these technologies have traditionally been utilized for applications of logical or analytical nature, AI in the context of arts and creative businesses is something that is getting more relevant. As the main art form pertaining to the sense of hearing, music is enjoyed by people from all ages and backgrounds. Being able to automate music generation could have interesting and widespread applications, such as being able to generate novel music on the fly in video games or as custom ambience in shops or elevators.

The aim of this project is to use machine learning to create music that is nearly indistinguishable from music made by humans. To solve this problem, a small literature study was performed to research already existing solutions or techniques that could be used. Out of these, two techniques were chosen for this project: autoencoders (AEs) and long short-term memory (LSTM) networks. To evaluate the outcome of the project, a Turing test was employed to see if people could tell apart our machine learning generated songs to human produced ones. To implement an AE model, an already existing project on GitHub, HackerPoet, was used as a starting point[1]. Similarly, to implement the LSTM-model, the pipeline created by Sigurður Skúli was used as a foundation [2].

[1] https://github.com/HackerPoet/Composer

[2] https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5

2 Method

To investigate whether artificial intelligence can be used to create music indiscernible from music created by humans, two ML models were created and evaluated: one AE model and one based on LSTM. Randomly selected songs generated from both models were compared to human composed music in a rating scale survey. The survey was sent out to 29 respondents wherein they were asked to rate the likelihood of each song being generated by a computer or not, without knowing the true source.

2.1 Data

To simplify pre-processing, interpretation and the easy [?] by which music could be decomposed and re-composed as discrete notes or chords, MIDI format was chosen for music data representation in the project. To use this data as input to the ML models, pre-processing was used to make the input more suitable for the models.

The data set used for training both models was the LPD-5 data set, originally created by Collin Raffel in 2016 [9] and expanded by Hao-Wen Dong et al. in 2018 [5]. Music from this data was also used in the rating scale survey as the authentic music to which the synthetically generated music by the models were compared.

For both models, the data was pre-processed by removing all but the piano MIDI tracks. For the LSTM model a vocabulary was created, mapping a note or chord to an integer token. Using this vocabulary, all tracks were translated from notes and chords to a sequence of such tokens. The training set was then created by taking token sequences with length n as input to the network, where the last token in the sequence was used for output prediction. When the input data was fed to the network, every integer token was divided by the length of the vocabulary to produce a value between 0 and 1.

For the AE, only songs with a tempo between 110 and 130 quarter-notes per minute were used for training data. This was to avoid the model mixing tempo when generating songs. To make sure that the AE trained on actual music and not on silence, the first part of every songs were cut out by looking at when the first note was played for every song. Lastly, to prevent the model from mixing the key when generating songs, every training song was transposed to be in the same key.

2.2 Autoencoder

Autoencoders are a type of neural network that compress a high-dimensional input down to a smaller feature space and then decompress it back to the high-dimensional space. It is often used as either a feature extractor or as a generator by providing a vector in the feature space and decompressing it. Since it just compares the input to the output there is no need for labelled data for learning a representation of the input data. The model trains by attempting to reconstruct the input data that is passed through the model and updating its weights to minimize the difference between the input and the output. This is advantageous for music generation as this unsupervised method does not require labelled data or a human supervisor to grade the output [1].

The model can be split up into two parts, an encoder part and a decoder part, see figure 1. The encoder is the first half of the model and its goal is to learn a representation of the data that is to be generated. The decoder is the second half of the model. In the training stage of the model, it is trying to reconstruct the input to the model from the new representation of the data that the encoder has generated. The loss function of the AE is measured by a reconstruction loss, i.e. how close the output data is to the input data. Once the model has been trained, the decoder can be used to generate new data by taking noise as input as if it were a deconstructed song in the learned latent vector space [1].

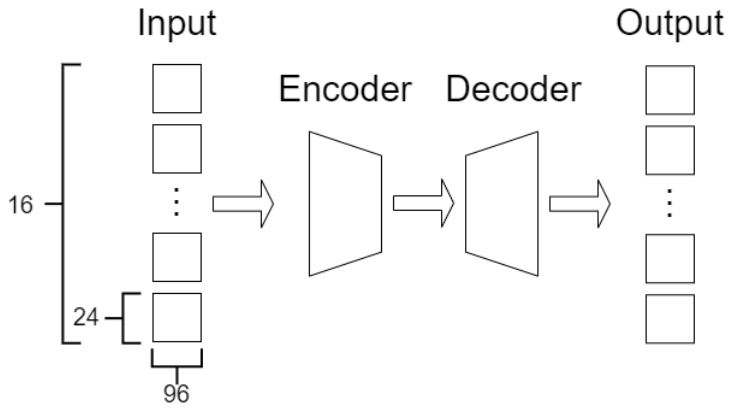


Figure 1: Representation of an autoencoder.

The input for the AE is a 16x24x96 matrix where there is 96 possible pitches in every bar, which in turn is split up into 24 time steps with 16 bars in every input. See figure 1 for input dimension representation. The input is first passed into the encoder, which consists of five layers. In the first layer the dimensions representing the pitches and time steps within a bar are concatenated. This gives a layer of size 16x2304. This is passed onto a dense layer with size 16x2000, then another dense layer of size 16x200. After that it is sent to a layer where the remaining dimensions are concatenated to make a vector of length 3200. This vector is sent to another layer that turns it into a vector of length 1600. Lastly it is passed through a layer resulting in a vector of length 120. This final vector representation is the new latent representation of the music that is to be reproduced and learned from when creating new music.

The decoder is a mirror image of the encoder, taking the latent vector representation of the music, decompressing it and reshaping it into the expected output dimensions.

In every layer except for the last layer a ReLU activation function is used. In the last layer a sigmoid activation function is used instead since the notes that are to be generated are either playing (1) or not playing (0). The sigmoid function returns a number between 0 and 1 which is then rounded to nearest integer based on a threshold, which we set to be 0.5.

2.3 Long short-term memory

The other model implemented in the project is the LSTM-based one. LSTM is a form of a recurrent neural network (RNN) architecture, which involves a collection of deep neural networks (DNN) that operates on sequential data. It is commonly used for temporal or ordinal machine learning tasks such as speech recognition and natural language processing [3, 10]. LSTM and RNN networks have previously been used to generate music, e.g. music by Keunwoo Choi et al [4] and Kalingeri et al [7].

Traditional DNNs assume inputs (and corresponding outputs) to be independent of each other. In RNNs, however, the current output and input are influenced by prior elements within the input sequence. To accommodate this, the individual layers of RNNs are organized as time steps, where each layer maps to an input in the sequence in order, and prior elements are cumulatively remembered within consecutive layers. For example, if the input of two prior layers influence the output of a current layer, in addition to the current input, the prior inputs would be stored as a hidden state in the current layer. Figure 2 contains a visual representation of a RNN. RNNs can be unrolled to visualize how information is propagated through the network, an example of this can be seen in figure 3.

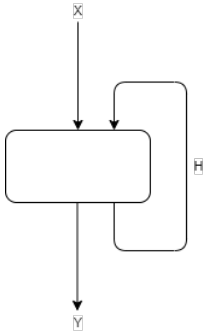


Figure 2: A schematic RNN network.

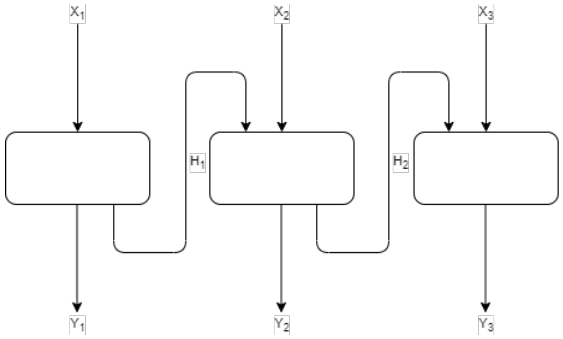


Figure 3: A rolled out RNN network.

Hence, RNNs share weight parameters across each layer and cannot use the same backpropagation algorithms that are used in traditional feedforward net-

works. Instead, backpropagation through time (BPTT) algorithms must be used to adjust weight parameters. BPTT is similar to other backpropagation algorithms in that the network trains itself by calculating the residual between its input and output nodes, and uses gradient descent to update the weight parameters. However, in BPTT the residuals are summed at each time step. Thus, RNNs commonly suffer from the exploding and vanishing gradient problem, respectively [2, 8].

These exponentially decaying or exploding errors lead to long training times. To solve this problem, Sepp Hochreiter and Jürgen Schmidhuber proposed the LSTM model [6]. A LSTM layer typically has a state that can store previous input, take new input, and take the output from the last time step as well to produce a new state and output. The idea is that the state contains the important information from the previous time steps, i.e. it facilitates memory in the layer.

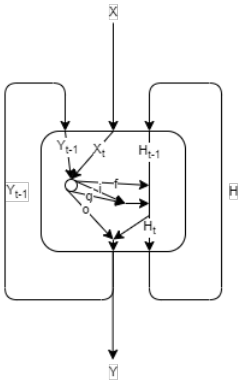


Figure 4: An LSTM layer.

In figure 4, H represents the state, Y represents the output, and X is the input to the node. The input is stacked with the output from the last time step. The new vector is used to update the state and create the output. The state is updated in two steps. The first step is to forget parts of the state, the new vector is scaled with the widths f and then passed through an activation function ($f = sig(W_f * [Y_{t-1}, X_t] + b_f)$). The second step is to add the new information that should be stored in the state. The information is chosen with the i and g weights and an activation function. The i weight describes what parts of the state should be updated while the g weight describes what should be written in the new state. After the state is updated the output is created; the new state is combined with the input to create the output. In the backpropagation calculations for LSTM, the weights that are updated are the weights f, I, g, and o.

To generate music, the LSTM network's task was to predict which element from the vocabulary that should follow a given sequence of elements from the same vocabulary. Two different networks were created to investigate the usefulness of LSTMs, one small network which consisted of two LSTM layers, both with 256 nodes. The networks were followed by a dense layer with 256 nodes and finally a dense layer with the same size as the vocabulary. Three dropout layers with a dropout chance of 0.3 were also included in the model to reduce

overfitting. The model predicted the next note based on the 64 previous ones. The small network was trained on two different-sized data sets to produce two different models. The small data set consisted of 128 songs, while the big data set consisted of 256 songs. The network was trained for over 200 epochs with batch sizes of 64. When new music was generated, a sequence of 64 random elements of the vocabulary was chosen and fed to the network. The input for the second note was then the 63 last notes from the random sequence and the generated node one.

The other network is a deeper version of the first one; it consists of ten LSTM layers, where the number of nodes were investigated for better performance. Layers of 128 nodes showed more promising results and was thus used in the model. Similarly, for the shallower version, the network has two dense layers after the LSTM layers (but with 128 nodes) and it also has dropout layers after each LSTM layer. The dropout chance was investigated and was tested between 0.01 and 0.3, but the more appropriate values ended up being 0.01. The deep network was trained on a small data set of 128 songs and a big set of 256 songs, with 100 epochs of batch size 64.

2.4 Evaluation

To evaluate the songs generated by the two models, a survey was carried out to determine how good the models were at replicating music that sounded like "real" music. The survey was designed like a Turing test where every applicant was asked to rate ten songs on a scale from 1 to 5 on how confident they were that the song was computer-generated or created by a human. A rating of 1 signifies that the participant was completely confident that the song was created by a human and a rating of 5 signifies that the participant was completely confident that the song was computer-generated. Two songs in the survey were taken from the training data and thus created by humans. Note that these two songs were pre-processed the same way as the training data used for the LSTM model. The remaining 8 songs were generated by the models: two were created by an AE trained on 10 songs, two were created by an AE trained on 256 songs, two were generated by the LSTM model trained on 128 songs and the last two were generated by the LSTM model trained on 256 songs.

3 Results

The results of this project are presented in three sections: first, the results of the survey and the quantitative human blind test evaluation, then the different characteristics of the AE and LSTM models, respectively.

3.1 Survey

In the survey, a total of ten songs were evaluated on their perceived authenticity; eight were generated by the two models and two were randomly selected from the training dataset to provide a point of comparison. Each of the five different model sources of the songs had two songs randomly selected to represent that source. The responses for these pairs of songs have been aggregated to give more robust statistical material to analyse. There were 29 respondents to the survey and thus there are 58 responses to each source, which are presented in figure 5.

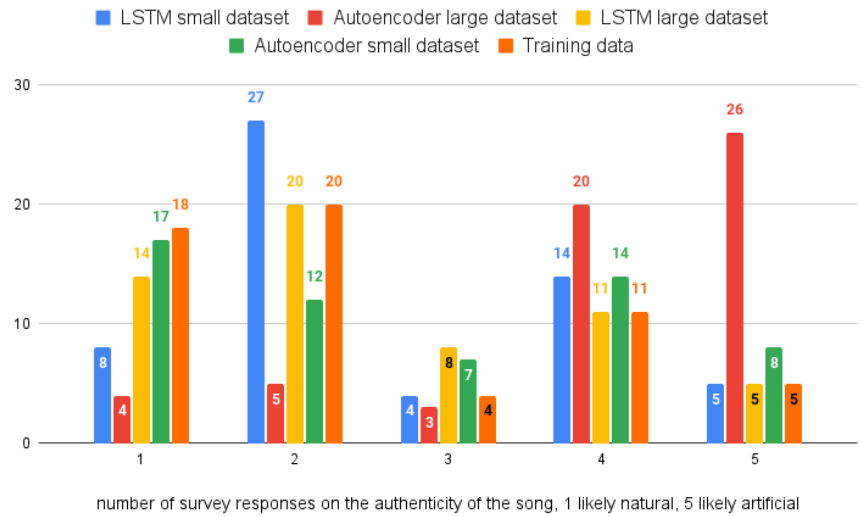


Figure 5: Aggregated comparison of experienced authenticity.

The bars in figure 5 represent the number of times a song from that category was graded with that particular score. For example, 27 of the responses to the songs generated from the LSTM-based model (trained on the smaller dataset) had a score of 2, i.e. the song was experienced to be more likely to be natural than artificial.

There were a few songs that got the grade 3, which would signify it is equally likely to be natural as it is to be artificial, regardless of the source of the song. The training data was most likely to be perceived as natural, but the LSTM models and the autoencoder trained on the small dataset were also often perceived as natural.

The probability to be perceived as natural is the proportion of grades 1 and 2 in comparison to the total number of responses. In other words, what is the likelihood that one response for a given model is at least 2.

Table 1: Statistical measurements on the survey responses.

	Training data	LSTM small dataset	LSTM large dataset	AE small dataset	AE large dataset
Arithmetic mean	2.40	2.53	2.67	2.72	4.02
Variance	1.79	1.66	1.52	2.13	1.49
Probability to be perceived as natural	70.37 %	68 %	64.81 %	56.86 %	16.36 %

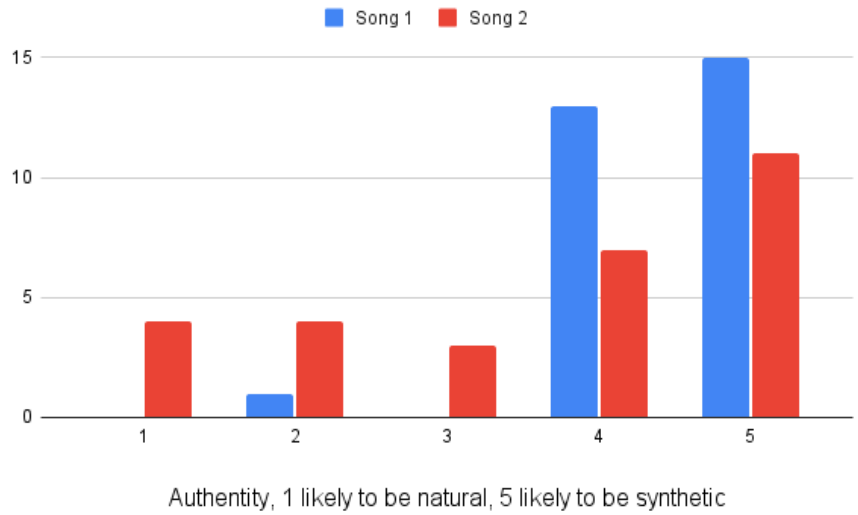


Figure 6: Comparison between the two individual songs from the autoencoder model trained on the large dataset.

A limitations of the survey was that only two songs were selected per source. In some cases there was also some significant difference between the two randomly selected songs from the same source. In particular, the responses to the songs generated by the autoencoder model trained on the large dataset, see figure 6.

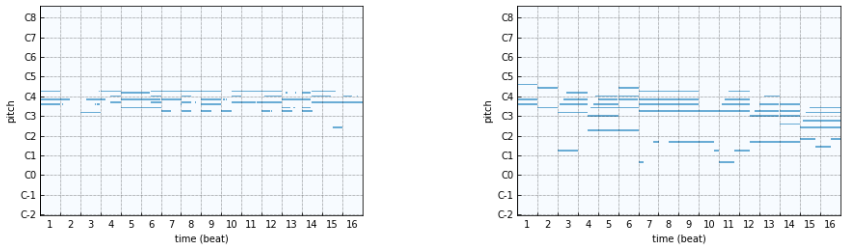
Table 2: Statistical measurements on the survey responses.

	Song 1	Song 2
Arithmetic mean	4.45	3.59
Probability to be perceived as natural	3.45 %	27.59 %

The difference between the two randomly selected songs from the same source is also evident in the statistical metrics as can be seen in table 2.

3.2 Autoencoder

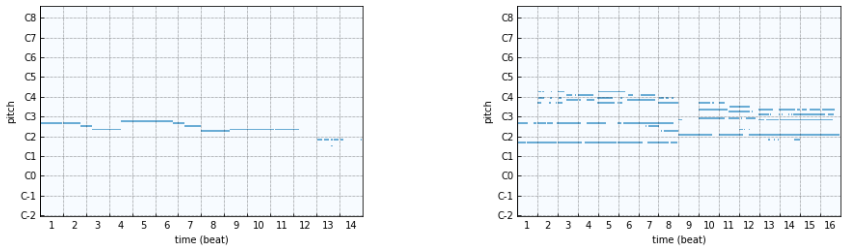
For the survey, four songs were generated using the autoencoder architecture. Two songs were generated from a model trained on ten songs and the remaining two songs were generated from a model that had been trained on 256 songs. All four songs were generated with a tempo of 120 quarter-notes per minute. A plot of their MIDI representation can be seen in figure 7 and 8.



(a) Song 1 generated from model trained on 10 songs.

(b) Song 2 generated from model trained on 10 songs.

Figure 7: Plot of MIDI representation for the 2 songs generated from the autoencoder model trained on 10 songs.



(a) Song 3 generated from model trained on 256 songs.

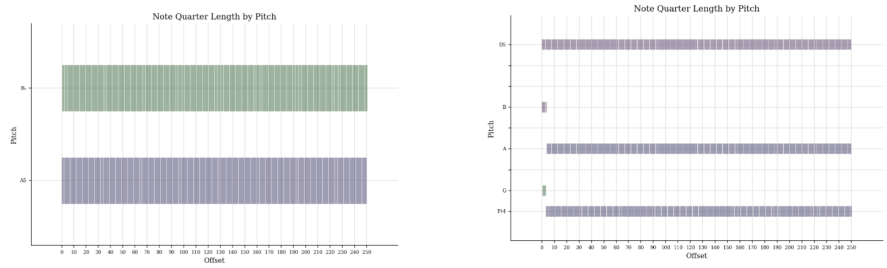
(b) Song 4 generated from model trained on 256 songs.

Figure 8: Plot of MIDI representation for the 2 songs generated from the autoencoder model trained on 256 songs.

3.3 Long short-term memory

For the survey, four songs were generated using the LSTM architecture, where two songs were generated by training on a dataset of 128 songs, and the other two were trained on 256 songs. The plots of the songs were not saved, but a general pattern of song quality was detected for all songs that were generated throughout the project.

3.3.1 Basic note songs



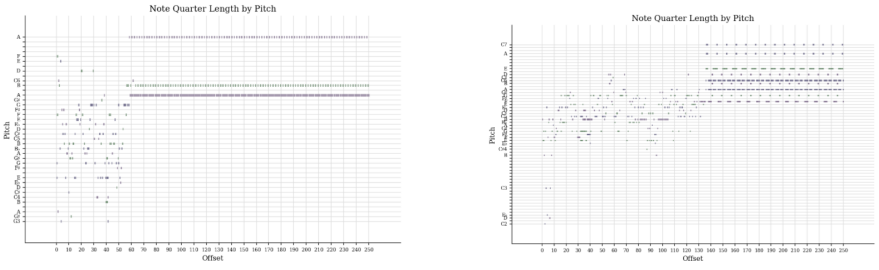
(a) Song generated with two notes.

(b) Song generated with few notes.

Figure 9: Plot of MIDI representation for two songs generated in a simple fashion using LSTM.

In figure 9, one can see a clear behaviour of the songs, generating a couple of notes only, where the same notes are pressed throughout the entire song.

3.3.2 Varied note songs



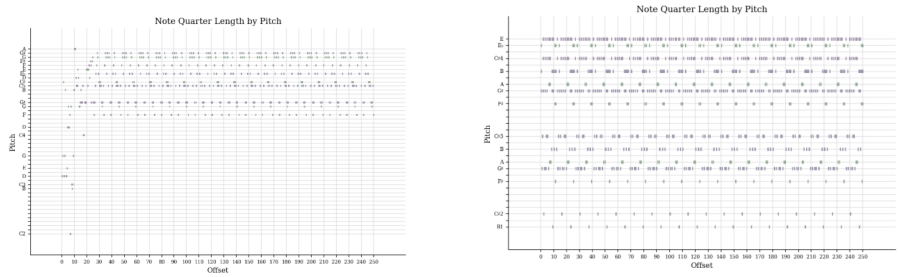
(a) Song generated with varied notes.

(b) Song generated with varied notes.

Figure 10: Plot of MIDI representation for two songs generated in a more varied fashion using LSTM.

In figure 10, one can see a wider range of notes being generated throughout the songs in a more "randomized" fashion. However, in some cases the songs tend to alter structure toward the end. The songs either play some single notes throughout the end as a finisher, or they copy a snippet of a song which is repeated throughout the rest of the song.

3.3.3 Overfitted songs



(a) Song generated with overfitted sequence of notes.

(b) Song generated with overfitted sequence of notes.

Figure 11: MIDI representation of two songs generated in a more overfitted fashion using LSTM.

Lastly, in figure 11, one can see very structured songs with repeated notes. This is due to the model taking snippets from the training set and repeating them endlessly.

4 Discussion

In this chapter the results will be discussed, as well as the choice of method. Lastly, a brief outlook on possible improvements and future work is discussed.

4.1 Survey

The results from the survey suggest that the perceived authenticity of the songs generated are not much worse than the training data. This could suggest that the best performing model performs almost as well as the human equivalent. Thus, the project's aim was achieved.

There is a spread in the results and notable differences between the models, as well as the amount of data used to train on. The results from the survey suggest that the LSTM-based model trained on the smaller dataset performed the best, and with the same model trained on the larger dataset being slightly worse. The autoencoder model performed slightly worse than the LSTM when it was trained on the small dataset and a lot worse when it was trained on the larger dataset.

The reason that the models performed better with less training data could be a sign that the model overfitted on the data instead of generalizing the task to generate truly novel data. In some cases parts of the generated music could be very similar to songs from the training data, this was mostly noticed in the models with smaller training data.

When making the survey it is desirable to have lots of questions and to sample many songs from each of the sources in order to have strong statistical backing when making claims about the results. However, it is not feasible to have too long surveys as the respondents might not bother finishing the survey at all, resulting in even less data being obtained. Therefore a trade-off was necessary, where only two songs from each source could be used in the survey, as well as using only excerpts of the whole songs. This means that there is some risk that the randomly sampled songs from each source do not accurately represent the general quality of the sources. For example, the two songs selected from the AE model trained on the large dataset had a significant difference in quality, as are presented in the results chapter.

The possibility of the randomly selected songs not being entirely representative of their sources would mean that the results have some additional uncertainty in what the models' true performances are. A future study attempting to replicate this project could arrive at other results. However, deeming on the overall quality of all songs generated, including those that could not be part of the survey, it would seem reasonable that the same general trend would be observed. That is, the models perform slightly below the training data in terms of perceived authenticity.

The survey had 29 respondents in total, meaning each type of source had a total of 58 grades, this is not a statistically insignificant amount, but as the grades have some variance within them, a larger survey would have been more desirable. It can be argued that it could have been better to have a survey that focused on fewer models, but more in-depth. This could have given a statistically more reliable result. However, it would not have been able to answer the question of how the amount of training data affect the resulting quality of the model.

4.2 Structure in songs

In this section, the structures observed in the songs will be discussed.

4.2.1 Autoencoder songs

The AE model generated four songs that were used in the survey, two of which were generated from the smaller dataset and two from the larger dataset. The songs generated from the smaller dataset got a significantly lower arithmetic mean and a lot higher probability to be perceived as natural sounding. The reason for this may be that the model does not have to adapt and adjust to as many songs. This may result in the model becoming very good at recreating the ten songs that it is trained on. A common trait for most of the generated songs were random notes being played that disrupted the melody. However one of the songs generated by the AE sounded significantly better and more natural with only a few notes disrupting the melody. The reason for this is probably that when the random noise vector was generated and combined with the feature space vector it became very similar to one of the songs in the training data. This results in the model recreating that training song almost identically. Since most songs created by the AE did not sound natural, this is likely the reason why one of the generated songs sounded more natural than the rest. The model did not learn enough characteristics of natural songs. Also, the reason that the AE trained on the smaller dataset got a better arithmetic mean is probably because it was better at replicating the training songs.

As mentioned above, random notes were often generated in places that you would normally not expect. This is probably due to the fact that the model did not completely learn how notes interact with each other. The model seemed to learn the general trend of how notes progress but there seems to be a lot of noise. The model outputs a number between 0 and 1 of how certain it is if a note should be played or not. In this project we choose a threshold of 0.5, however, you could set this to be lower in the hopes of removing these seemingly disruptive notes.

4.2.2 LSTM songs

Songs generated using the LSTM model were generally perceived as more natural sounding according to the survey results. However, as in the case of the autoencoder model, they also contain randomly occurring notes that sound dissonant or disharmonious. Also, sequences of long repetitions of the same key appear in the songs generated by the LSTM model.

We believe that the leading cause for these problems is insufficient training, which could be ameliorated by using larger training sets and longer training periods. Long repeats of the same note could simply be overfitting to the most commonly occurring note in the training set. Another possibility is that the imposed limitation to the generated music tempo, i.e. using a static duration of 0.5 seconds for each note, might cause the system to try and model drawn-out notes or chords in the training data, as multiple recurring presses of the same note. If that is the case, extending the system to learn how to model varying note duration might alleviate this problem.

4.3 Models

Both models generated music with varying degrees of success. According to the results of the survey, the LSTM model was better than the AE. However, as discussed above, the model may have overfitted and just reproduced the training data. Thus, any well-grounded conclusions are hard to draw. It is also possible that with different parameters, depth, and/or layer sizes the models could have performed better. Therefore, no general conclusions regarding the different techniques as such can be drawn.

One advantage with using LSTM over AE is the ability to generate songs of any length. The AE has a fixed-sized output, while the LSTM can generate music of any length. This could be useful in applications such as generating music in video games.

4.4 Future work

This study presents a way to generate music using LSTM and AE. Given the chosen methods, there were some limitations in performance that could have been further explored if time allowed.

4.4.1 Model improvements

Given the generated songs, one could conclude that the current models perform decently, but that there are areas that could be further improved. Hyperparameters, as well as the size and depth of the model, were chosen in a way that worked for the project. But no further investigations were made to find better performing models due to the lack of time. It is thus concluded that this could be a candidate problem for future works, where research of suitable models is to be made.

Some other aspects also affected the models' performances, e.g. the size of the data set. There were not enough data to properly train the models. As a result, overfitting became a common problem. Therefore, future studies should investigate the performance of the models when using larger sets of training data.

When implementing the LSTM model, there were several problems that had to be considered. For instance, the implementation cannot model the tempo of the songs, nor if a note is drawn out, i.e. played for a longer period of time. It only considers which notes are played at a certain time point. Thus, it became a discrete version of a song. When a note is played for multiple consecutive ticks, it gets represented as several distinct notes, one for each tick. Thus, one area of improvement would be to make the model able to encode notes for multiple consecutive ticks as one token instead of several separate ones. Since the entire generated song has a constant tempo, one could also investigate whether one can generate songs with varied tempo.

Lastly, an idea that the group had, but could not execute due to lack of time, is the usage of generative adversarial networks (GANs) in the system to generate songs. This could be an improvement that could be tested and added in future works.

4.4.2 Survey

An improvement for future work is to make the survey more exhaustive, to capture more of the opinions of the respondents. For instance, some of the real songs, mixed with the artificial songs, could sound artificial themselves. So it could be imagined that the participants struggled with the survey itself and had opinions that they wanted to express. One could for instance allow the respondents the options to provide feedback at the end of the survey. Another way to expand the survey in a more general and unbiased way is to have a bigger variety of participants, where they differ in age, sex, cultural background, musical background, etc. Since we mostly had participants in the age range of 20-30 who were interested in technology (we mostly got answers from participants in the course TDDE19), it would be interesting to investigate how people from other backgrounds and professions would rate the generated music.

References

[1] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 37–49, Bellevue, Washington, USA, 02 Jul 2012. PMLR.

[2] Yoshua Bengio, Paolo Frasconi, and Patrice Simard. The problem of learning long-term dependencies in recurrent networks. In *IEEE international conference on neural networks*, pages 1183–1188. IEEE, 1993.

[3] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[4] Keunwoo Choi, George Fazekas, and Mark Sandler. Text-based lstm networks for automatic music composition. *arXiv preprint arXiv:1604.05358*, 2016.

[5] Li-Chia Yang Hao-Wen Dong, Wen-Yi Hsiao and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. 2018.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[7] Vasanth Kalingeri and Srikanth Grandhe. Music generation with deep learning. *arXiv preprint arXiv:1612.04928*, 2016.

[8] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

[9] Colin Raffel. Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching. 2016.

[10] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.