Predicting Mozart's Next Note via Echo State Networks

Ąžuolas Krušna, Mantas Lukoševičius
Faculty of Informatics
Kaunas University of Technology
Kaunas, Lithuania
azukru@ktu.edu, mantas.lukosevicius@ktu.lt

***Abstract*—Even though algorithmic music has been around the world since the old days, it has never attracted as many researchers as in the recent years. To our knowledge it existed in Iran back in the Middle Ages and in Europe during the Age of Enlightenment. Though the form has changed and it has grown layers of complexity, the very foundations of the algorithm that generates musical compositions have not changed, i.e. most of them are based on structures of fortuity. Additionally, models that are able to learn have been discovered allowing us to imitate the music of the incredible artists throughout history. The thought alone is crazy to think of and seems to be from the sci-fi. In this paper, a research trying to find the best model of an echo state network in order to mimic the music of the legendary Wolfgang Amadeus Mozart has been carried out. As it turns out, the best models are the ones that rely on long-term dependencies.**

Keywords—algorithtmic composition, echo state network, MIDI, recurrent neural network

I. INTRODUCTION

Algorithmic music is by no means a new trend in our techy world. In fact, three Iranian brothers collectively known as Banu Musa were successfully devising automatic and even programmable musical instruments back in 850 AD [1]. They were most likely invited to the best parties in the city back then. Moreover, an algorithmic game circulated around Europe since the Enlightenment Age, i.e. the 18th century in a form of *Musikalishes Würfelspiel.* It has been attributed to Mozart in a form of myth, yet never proven to be true. This game took small fragments of music and combined them in a random order by chance, often tossing a dice [2]. Since then, the scope of the algorithmic music has augmented layers of complexity, but the foundations have not changed. The main difference is that now we do not toss a dice, but run a random number generation function in our favourite programming language.

In this article we are trying to imitate classical piano music. For a quantitative rather than qualitative analysis only one composer was chosen. Mozart has been opted for his indisputable genius and some haphazardness.

Rather than working with sound signals, we chose to work with notes for several reasons. Firstly, it is a lot less intricate. Therefore, it is a lot easier for us to understand and analyse it as well as it is for the algorithm in the means of computational resources and dependency on previous notes. In the note level we are also able to compare it with the musical theory. And it helps us stay in the realm of classical music as well.

Furthermore, we are lucky enough to have the MIDI (musical instrument digital interface) protocol for a *.mid* is a musical file format that captures the notes, the times piano keys were pressed and released, how strong they were pressed etc. MIDI supports 128 notes whereas general pianos usually provide 88 keys.

Musical composition has been one of the long term goals of artificial intelligence (AI) [3]. Broadly speaking, music generation by AI is based on the principle that musical styles are in effect complex systems of probabilistic relationships, as defined by the musicologist Leonard B. Meyer. In the early days, symbolic AI methods and specific grammars describing a set of rules had driven the composition [4], [5]. Then these methods were significantly improved by evolutionary algorithms in a variety of ways [6] as represented by the famous EMI project [7]. More recently, statistics in the form of Markov chains and hidden Markov models (HMM) played a major part in algorithmic composition [8]. Next to this development was the rapid rise of neural networks (NN) due to the growing capacity of computational powers. It has made a remarkable process not only in the AI world but also in music composition [9].

As music is a sequence of notes, a sequential model was chosen to train on Mozart's music. Markov models are not very suitable for this task due to their monophony (although it is possible to design a system for polyphonic music as well). Currently, the cutting-edge approach to generative music modelling is based on recurrent networks [4], [10], [11] like the long short-term memory (LSTM) network. Traditional recurrent neural networks (RNN) lack long-term dependency, thus are able to generate melody yet no harmony, i.e. the music gets stuck at some point or turns out to be repetitive. LSTMs are better in this case since they have a stronger long-term dependency. Though fine-tuned LSTM algorithms are able to overcome the obstacles that traditional RNN algorithms confront, they still face the same problems in a way that the music lacks the theme, i.e. *the big picture*. Long short-term memory algorithms have been extensively studied in the recent years. Besides, LSTM algorithms are also heavy and require a lot of resources. We have been looking for a light-weight solution.

For these reasons, we chose to work with a type of recurrent neural networks – echo state network (ESN) – that have barely been researched for musical composition.

II. Data & Tools

Musical data were downloaded in the format of *.mid* from the website http://www.piano-midi.de/. From now on MIDI and *.mid* will be used interchangeably meaning the same, i.e. the file format unless stated otherwise, e.g. MIDI protocol. In total, 21 pieces by Mozart were gathered (all that are found on the website).

MIDI format is a sequence of notes (and commands such as tempo change and sound perturbations) whereas the time difference is represented in ticks. A quarter note is usually 480 or 960 ticks but that depends on the resolution. Thus, a full note or, in other words, a tact is 1920 or 3840 ticks respectively.

Later on, the data had to be transformed in a format that is easier to read, maintain and process. Hence, it was read and transformed into notes as messages into a *.csv* (comma separated values) format.

Every message consists of information of this type:

- note pitch
- on tick
- off tick
- length

The length parameter is not in the MIDI file and had been artificially generated for the purpose of data analysis.

Table I shows the types of information as well as their ranges in a message. Note pitch ranges from 0 to 127, thus a byte is more than enough to store it. The beginning and the end of a note tick is undetermined and can grow to infinity when the data grows. Length parameter is purely the difference between on and off ticks. It may grow to a large number due to software bugs or a divergence of the algorithm, but usually it shall stay in the realm of classical music and get a value up to a full note.

TABLE I. INFORMATION INSIDE A MESSAGE

Info	**Note pitch**	**On tick**	**Off tick**	**Length**
Type	byte	long	long	integer
Range	0-127	0-*infinity*	1-*infinity*	1-*full note*

A message in MIDI that signifies the event of pressing a note is the *note_on* message. It represents an event when a note is released as well, only the velocity then is equal to zero. The algorithm (Fig. 1) that was applied for treatment of raw MIDI files looks as following:

```
iterate through the messages:
--check if it is a 'note_on' type of message:
----if velocity > 0:
------take the time of the note that was pressed
----else if velocity equals 0:
------check if the actual note was pressed:
--------release the note
--------measure the length of the note
----------append the note as a message to the CSV
```

Fig. 1. Algorithm of raw MIDI file treatment into a CSV file

The programming language of choice was *Python* due to its recognition in data science and machine learning among scientists and developers. Also, due to the many data processing as well as machine learning libraries although none of the machine learning libraries were used for this work. For the purpose of *.mid* processing, *Mido* library was chosen [12].

Machine learning algorithms perform better under more data. We could have just taken in all of the composers from the website full of classical MIDI files, but we chose only one for the purpose of thorough analysis. Despite the fact that our choice was only Mozart's music and that had given us only 21 pieces of scores, this resulted in around 68 thousand notes.

III. Initial Data Analysis

Prior to the research, an analysis of the data was performed based on the distribution of note pitches as well as their lengths. As we can clearly see in Fig. 2, there are 2 maximums. One is of a higher pitch while the other is of a quite lower pitch. This is most probably due to the fact that piano is played by 2 hands and that the left hand usually wanders in the region of lower pitch notes whilst the right hand sits in the region of higher pitch notes.

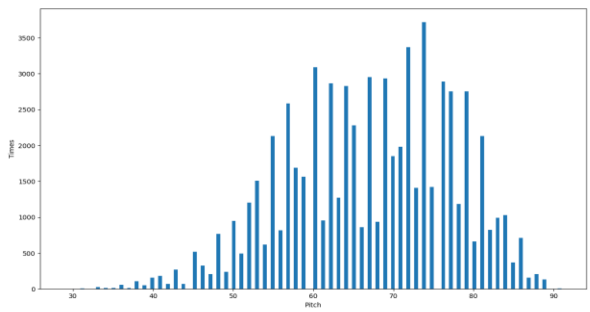

Fig. 2. Distribution of Mozart notes. One can obviously spot 2 maximums of a higher and a lower pitch. This is most likely due to the fact that piano is played by 2 hands and that the left hand usually wanders in the region of lower pitch notes whilst the right hand sits in the region of higher pitches.

These data are not so much relevant for our research, but provide us with insights such as it would make perfect sense to study the hands in more detail. We ought to bolster our research either by adding an additional dimension of the hand or by having 2 different outputs for each hand by the network. This analysis is also useful for future comparison and judgment of generated music.

Analysis of lengths (Fig. 3) provide us only one maximum, meaning both hands share the same maximum or that the note lengths of one hand are very dispersed.

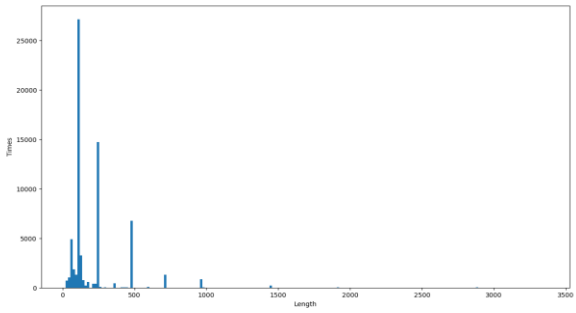

Fig. 3. Distribution of Mozart note lengths.

IV. Network

Echo state networks supply an architecture and principles of supervised learning for recurrent neural networks. The idea behind an ESN is to drive a large, random and fixed *reservoir* of neurons with the input signal (Fig. 4). Thence, inducing each neuron within it with a nonlinear response signal. After, combine the desirable output data by a trainable linear combination of all of these response signals [13]. In practice, it is important to keep in mind that the *reservoir* acts not only as a nonlinear expansion, but also as a memory input at the same time [14].

Echo state network may be tuned by altering the following parameters:

- leaking rate
- input scaling
- spectral radius

Leaking rate of the network can be regarded as the speed of the *reservoir* update dynamics in discrete time.

Another key parameter to optimize an ESN is the input scaling. It multiplies the input weight matrix W^{in} by its value either strengthening the input weights or diminishing them.

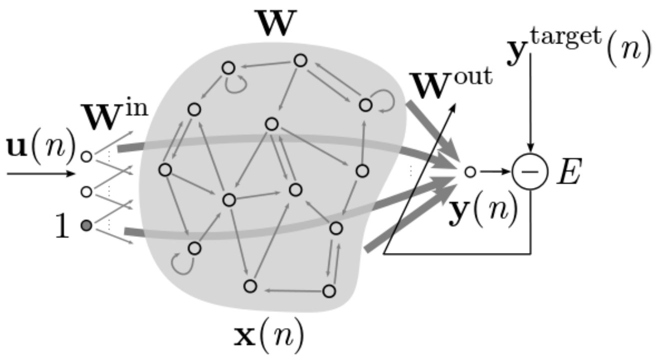

Fig. 4. Design of an echo state network [14]. Here u is the input data, W^{in} is the input weights matrix, x is the *reservoir* nodes and their outputs, W is their weights, W^{out} is the output weights and y is the output data.

Spectral radius is one of the most global parameters of an ESN, i.e. the maximum absolute eigenvalue of the *reservoir* weights matrix W. It scales the matrix W, or in alternate words, scales the width of the distribution of its nonzero elements [14].

In order to avoid overfitting, regularization is used.

The number of neurons inside the *reservoir* has been opted be equal to 1000.

Programming code for ESN has been adapted from [15] and expanded for multidimensional input data as well as output.

V. Experimental Setup

Research has been accomplished in a manner that can be seen in Fig. 6. First of all, music was accumulated in *.mid* format (hex code). As stated before, it was processed by *Mido* library and stored in a *.csv* format in a form of messages that carry the information of notes as the pitch number, on and off ticks and length.

Then the messages were read from the *.csv* file and quantized. Quantization was performed for the beginning and the end of the notes in the following way. A quantization unit of 60 ticks (represents a 32nd of a note) was chosen. Next, if the residual value of the tick was less than half the quantization unit, it was reduced by the residual. If the residual value was equal or higher than half of the quantization unit, i.e. 30 ticks, it was increased by the difference between the quantization unit and the residual. The lengths of notes were recalculated afterwards.

In Fig. 5 we can see the distribution of the notes after quantization. Hereby, the number of notes of the length of the quant (60 ticks) has increased. The most frequent note stayed the same (120 ticks). Also, a tiny part of the very shortest notes was quantized to zero length, thus, eliminated.

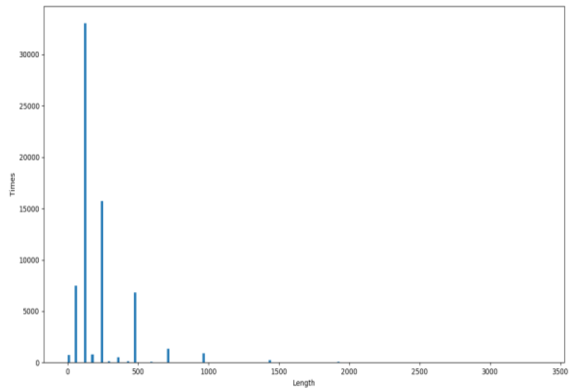

Fig. 5. Lengths of quantized notes whereas the quantization unit is 60 ticks.

As a further step, these quantized music messages were turned into a state matrix of length that is equal to the division of the total length of the pieces by the quantization unit rounded to integer. Another dimension of the state matrix were the note pitches, that is 128 values in total. Then the value at each time step at a certain note represents its state (1 for pressed and 0 for not pressed). 80% of the data were sent to the echo state network whilst 20% were used for validation of the model, thus finding out the error. Error was calculated in the shape of root mean squared error (RMSE).

An ESN was generated according to given parameters. This ESN was then trained on input and predicted music based on its learned weights as a one time-step prediction. The training process was initialized by 300 time steps, that is by 300 quants (60 ticks).

To find out the best parameters for our echo state network, we would repeat the procedure of generating the network according to different parameters and training the new network model on the very same data. Then we predicted next notes based on the newly gained weights and found out the error by comparing with the original Mozart data. Prediction of notes was a sequel of the training process. To be more precise, the model predicted notes as a one time-step prediction. Summarizing, a grid search analysis of 4 parameters of the echo state network has been performed.

Parameters that have been investigated for tuning our network are the following. Leaking rate, input scaling, spectral radius and regularization which are the most important ESN parameters explained in Section IV. Since their ranges usually go from 0 to 1, 0 to 2, 0 to 2 and almost anything respectively, they have been tested for values in these ranges. An exhaustive grid search analysis had been performed looking for the best parameters. In addition to RMSE, mean and standard deviation were calculated. Original Mozart music had the mean of 0.04238 and standard deviation of 0.0779. Mean represents the probability of a note to played at each time step in the note spectrum. In Mozart's case note spectrum is from the 29^{th} to the 91^{st} note. Standard deviation represents the mean of standard deviations of the notes in the note spectrum.

Leaking rate has been tested from 0.0025 to 1, spectral radius varied from 0.0015 to 2 in this test, input scaling from $2*10^{-6}$ to 2 and regularization from 10^{-6} to 10^{5}.

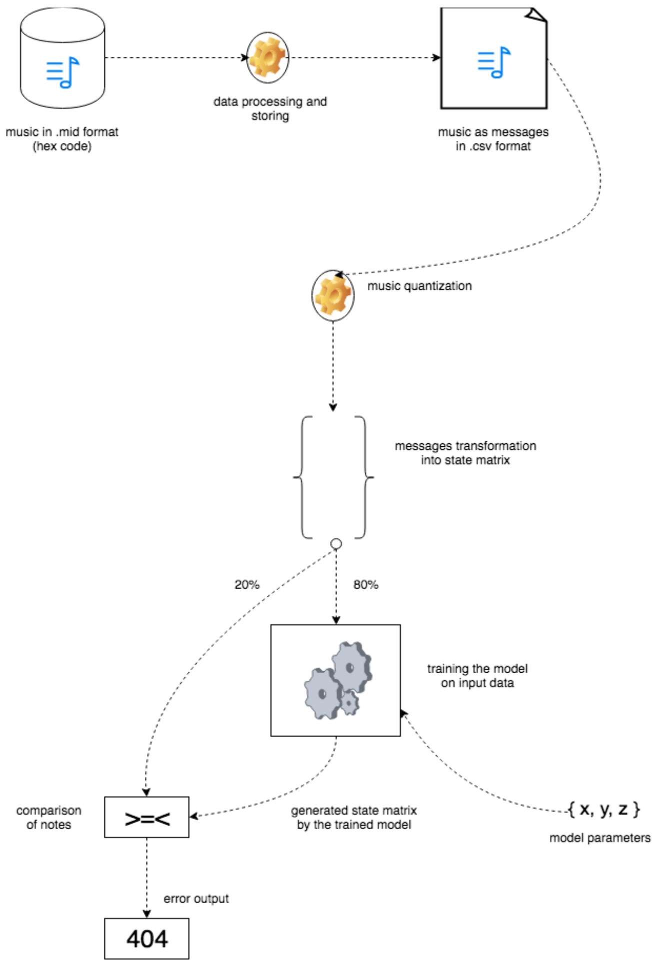

Fig. 6. Scheme of research. Music is processed from the *.mid* format into *.csv* format. Then quantized and transformed into state matrix. 80% of the data are fed to the network while 20% are compared to the predicted data from the trained network model generated with given parameters. Lastly, the errors for given parameters are printed out.

VI. Results

As we can see from the sorted by error (top 10) Table II, the lowest value of error (RMSE) is a tiny bit above 0.0307. It is clear that the best leaking rate for our model is about 0.025 while the combination of input scaling and spectral radius vary a little bit. Input scaling goes from 0.002 to 0.0002 and spectral radius from 0.01 to 0.1. We can notice that while RMSE is the lowest, the mean of the notes is about the same of the quantized original Mozart music data mean but standard deviation is quite different.

reg stand for regularization, *rmse* stands for RMSE and std stands for standard deviation in the tables of error (Table II, Table III, Table IV).

TABLE II. Sorted error data (top10)

leaking rate	input scaling	spectral radius	reg	mean	rmse	std
0.025	0.0002	0.1	0.0001	0.0410	0.030769	0.05696
0.025	0.0005	0.01	0.001	0.0411	0.030771	0.05699
0.025	0.0005	0.01	0.001	0.0411	0.030771	0.05698

0.025	0.0002	0.02	0.0001	0.041	0.030772	0.05697
0.03	0.0005	0.1	0.001	0.0411	0.030772	0.05699
0.025	0.0005	0.05	0.001	0.0411	0.030773	0.05699
0.03	0.0005	0.06	0.001	0.0411	0.030774	0.05699
0.02	0.0006	0.02	0.001	0.0411	0.030775	0.05697
0.015	0.0006	0.02	0.001	0.0411	0.030776	0.05697
0.02125	0.002	0.1	0.01	0.0410	0.030776	0.05697

Having low leaking rate suggests us that the state has a lot of *inertia* and the change of the state is slow. Input scaling scales the W^{in} matrix, thus, the input weights are very low and the model depends on its input just a tiny bit. Since it is lower than the spectral radius, it has a lot of memory, i.e. follows a long-term dependency. Having low spectral radius as well tells us that the models are almost linear. To summarize, the prediction function is not very complex and the model has a lot of memory.

From Table III we see that a high regularization value gives us huge errors. It has to be noted that for this particular grid search step, the maximum value of input scaling and spectral radius was 0.2. Thus, we can also deduce that high input scaling values lead to higher error. Though leaking rate is not as important, we can still see that some of its higher values lead to higher errors.

High regularization significantly reduces the mean value and standard deviation of the notes.

TABLE III. SORTED ERROR DATA (WORST10)

leaking rate	input scaling	spectral radius	reg	mean	rmse	std
0.175	0.2	0.2	100000	0.0345	0.064397	0.00959
0.175	0.2	0.14	100000	0.0357	0.064656	0.00952
0.1	0.2	0.02	100000	0.035	0.064968	0.00801
0.1	0.2	0.2	100000	0.0352	0.065097	0.00817
0.1	0.2	0.14	100000	0.0353	0.065158	0.00806
0.1	0.2	0.08	100000	0.0354	0.065258	0.00808
0.025	0.2	0.02	100000	0.035	0.066049	0.00572
0.025	0.2	0.14	100000	0.0352	0.0662	0.00584
0.025	0.2	0.08	100000	0.0353	0.066243	0.00585
0.025	0.2	0.2	100000	0.0354	0.066295	0.00582

In order for us to see tendencies beyond regularization, we filtered the data for regularization below or equals 10. This brought us back to the maximum values of input scaling, spectral radius and leaking rate.

In Table IV we see that high input scaling produces high error once again. Interestingly, leaking rate stays at 0.25 for the highest error. Although spectral radius stays quite high, it is not of the highest value for the highest error. Mean is almost as with the best results. Standard deviation is higher in this case than with the best results. It is even closer to quantized Mozart's music standard deviation than the one provided with the best results.

TABLE IV. SORTED ERROR DATA (WORST10) WHILE REGULARIZATION IS SMALLER OR EQUAL TO 10

leaking rate	input scaling	spectral radius	reg	mean	rmse	std
0.25	2	0.8	1	0.0405	0.043065	0.06195
0.25	2	0.8	0.1	0.0405	0.043094	0.06198
0.25	2	1.4	0.01	0.0405	0.043098	0.06199
0.25	2	1.4	1	0.0406	0.043128	0.06173
0.25	2	1.4	0.1	0.0406	0.043139	0.06175
0.25	2	1.4	0.01	0.0406	0.043142	0.06175
0.25	2	1.4	10	0.0406	0.043169	0.06171
0.25	0.8	1.4	1	0.0413	0.043178	0.06141
0.25	0.8	1.4	0.1	0.0413	0.043234	0.06145
0.25	0.8	1.4	0.01	0.0413	0.04324	0.0615

Fig. 7 shows us the minimum error dependency on leaking rate. It is worth to note that although leaking rate 0.25 yields worst results when regularization is not high, it may also yield very good results with other values of ESN parameters as can be seen in Fig. 7.

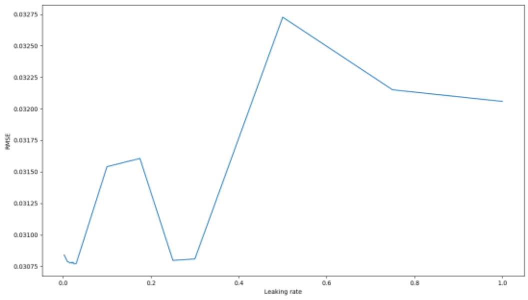

Fig. 7. Minimum RMSE dependency on leaking rate.

The errors were grouped by leaking rate and the minimum value of the error was taken to plot the dependency graph. In Fig. 8 we can see the most promising region of leaking rate for our echo state network.

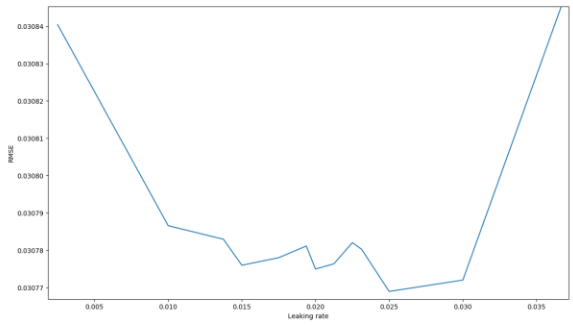

Fig. 8. Zoomed minimum RMSE dependency on leaking rate.

Fig. 9 shows us the minimum error dependency on input scaling whereas Fig. 10 zooms us to the most promising region of input scaling. The best values of input scaling are 0.0002 and 0.0005. Going even lower, the values increase dramatically.

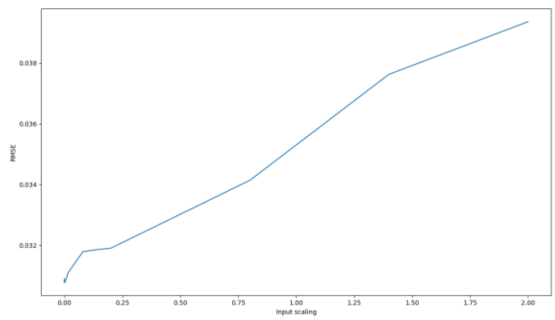

Fig. 9. Minimum RMSE dependency on input scaling.

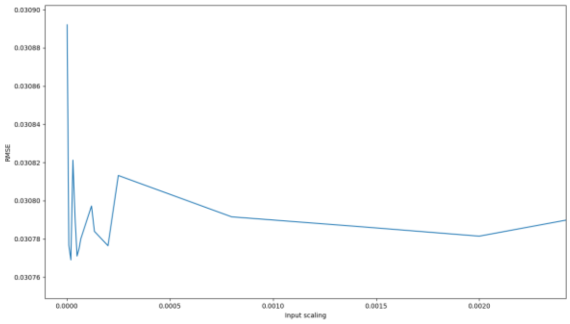

Fig. 10. Zoomed minimum RMSE dependency on input scaling.

Fig. 11 shows us the minimum error dependency on spectral radius. From Fig. 12 and Fig. 13 we can see that the minimum RMSE stabilizes and reaches the minimum on spectral radius below 0.1. Then starts growing again above 0.01.

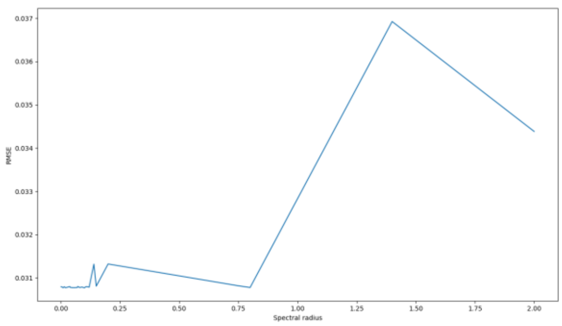

Fig. 11. Minimum RMSE dependency on spectral radius.

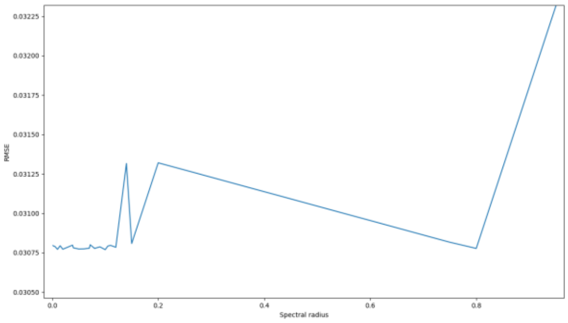

Fig. 12. Zoomed minimum RMSE dependency on spectral radius.

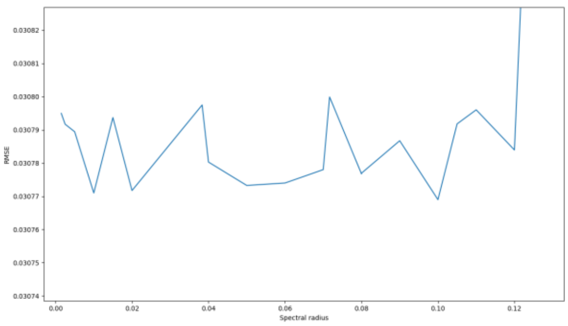

Fig. 13. Zoomed minimum RMSE dependency on spectral radius to the most promising region.

Fig. 14 and Fig. 15 implies us that the best regularization values are of the power 10^{-4} to 10^{-2}.

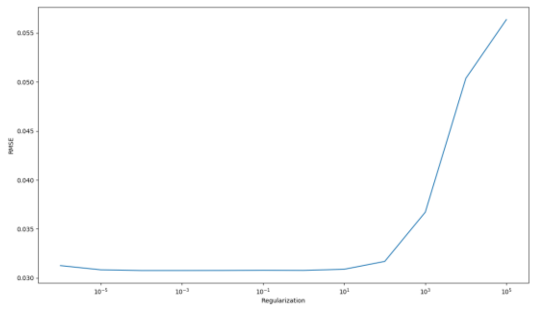

Fig. 14. Minimum RMSE dependency on regularization.

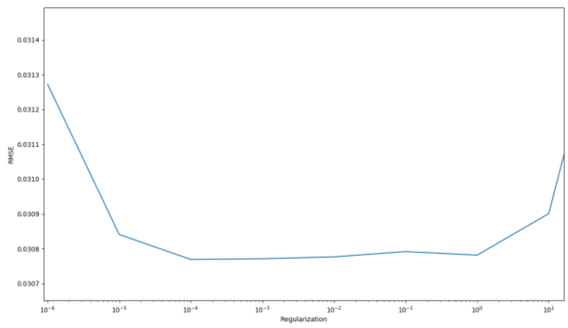

Fig. 15. Zoomed minimum RMSE regularization on dependency.

Since it was a lot easier to find the optimal leaking rate than input scaling and spectral radius, we grouped the errors by input scaling and spectral radius taking the minimum RMSE value in Fig. 16. Regularization is an additional parameter that prevents overfitting and we have not grouped by it. It was quite easy to find as well.

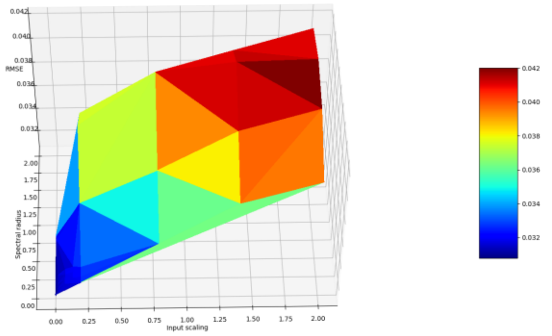

Fig. 16. Grid search minimum error (RMSE) grouped by input scaling and spectral radius.

As it was found out that the most optimal leaking rate is 0.025, the errors were grouped by input scaling and spectral radius once again by a set leaking rate. Now they we grouped having leaking rate set to 0.025. In Fig. 17 we can see pointy triangles in the lower region where the errors are the lowest.

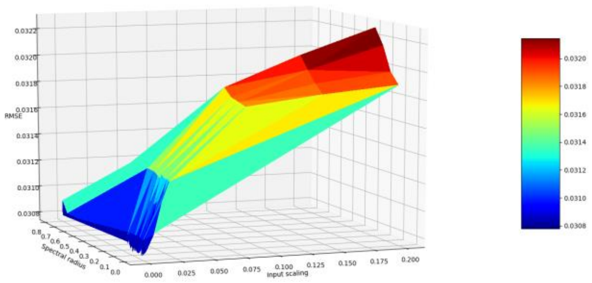

Fig. 17. Grid search minimum error (RMSE) grouped by input scaling and spectral radius while leaking rate equals 0.025.

It has to be taken into account that producing an even finer grid might give us even better results but this takes time. Also, it seems from all this analysed data that the reduction in error would be quite low.

VII. Conclusions

We can affirm that the best value of leaking rate in our research proved to be 0.025. The best values of input scaling are 0.0005 and 0.0002 whereas the most optimal values of spectral radius and regularization vary from 0.1 to 0.01 and from 10^{-4} to 10^{-2} respectively. Having said that, the values will not produce the best results in separation, they will only produce the best results in a proper combination with other variables as it can be seen in the tables and figures.

To summarize our research, we can state that to predict Mozart's music, one has to memorize a lot of the notes in order to predict the next note. In the terms of our echo state network, it has to follow a long-term dependency because the input scaling is lower than the spectral radius. Having low spectral radius as well implies that the prediction function ought to be quite simple because the reservoir operates in an almost linear regime.

VIII. Future Work

Our main aim is to produce good music so that people would like to listen to it. To achieve this goal, we analysed the best models to replicate Mozart's music. Lately, we have been planning to include information of piano hands into our composition model. In the future work we would like to expand the dimensions of this research since MIDI files have additional information such as the velocity of the pressed note as well as tempo changes and sound perturbations. We are also eager to expand this study for more great composers and then tune our models to not only imitate but also generate new music that people would value. If echo state networks do not prove to be *deep* enough, we are determined to broaden our research including deep learning models such as hierarchies of regular recurrent neural networks or long short-term memory networks and other recurrent types. We could then compare them and possibly combine the best parts of them. We are hoping that the artificial network is able to learn the rules or tendencies of music theory implicitly, at least partially. If this is not the case, we could augment it with heuristics.

References

[1] MIDI history: chapter 1 – 850 AD TO 1850 AD, https://www.midi.org/articles/midi-history-chapter-1, accessed on March 2018.

[2] Cope, D. (1996). Experiments in musical intelligence. A-R Editions, Inc.

[3] Z., Sun, *et al.*, "Composing music with grammar argumented neural networks and note-level encoding," *arXiv preprint arXiv:1611.05416v2*, 2016.

[4] G. M. Rader, "A method for composing simple traditional music by computer," Communications of the ACM, vol. 17, no. 11, pp. 631–638, 1974.

[5] J. D. Fernandez and F. Vico, "AI methods in algorithmic composition: a comprehensive survey," *Journal of Artificial Intelligence Research*, vol. 48, no. 48, pp. 513–582, 2013.

[6] K. Thywissen, "Genotator: an environment for exploring the application of evolutionary techniques in computer-assisted composition," *Organised Sound*, vol. 4, no. 2, pp. 127–133, 1999.

[7] D. Cope, "Computer modeling of musical intelligence in EMI," *Computer Music Journal*, vol. 16, no. 16, pp. 69–87, 1992.

[8] M. Allan, "Harmonising chorales in the style of Johann Sebastian Bach," Master's Thesis, School of Informatics, University of Edinburgh, 2002.

[9] D. Silver, *et al.*, "Mastering the game of Go with deep neural networks and tree search." Nature, vol. 529, no. 7587, pp. 484–489, 2016.

[10] H. Chu, R. Urtasun, S. Fidler, "Song from PI: A Musically Plausible Network for Pop Music Generation", 2016, https://arxiv.org/abs/1611.03477H, accessed on March 2018.

[11] A. Huang, R. Wu, "Deep learning for music," 2016, https://arxiv.org/abs/1606.04930, accessed on March 2018.

[12] Mido – MIDI objects for Python, https://mido.readthedocs.io, accessed on March 2018.

[13] H., Jaeger "Echo state network," Scholarpedia, http://www.scholarpedia.org/article/Echo_state_network, accessed on March 2018.

[14] M., Lukoševičius, “A Practical Guide to Applying Echo State Networks,” *Neural Networks Tricks of the Trade*, 2nd e., Springer, 2012

[15] Sample echo state network souce codes, http://minds.jacobs-university.de/mantas/code, accessed on March 2018.