Music generation using autoencoders and CONLON pianoroll

1st Jonne Kurokallio
Aalto University
School of science
Espoo, Finland
jonne.kurokallio@aalto.fi

***Abstract*—This project is a reconstruction of CONLON pseudosong generation study conducted by Angioloni et al. [1] In that study authors presented a more accurate and efficient pianoroll structure to be used in pseudo-song generation. This project uses similar model architecture and data processing methods, but also compares the received results of Wasserstein auto endocer (WAE) to a more simple variational autoencoder (VAE). All in all the goal is to produce smooth and continuous music to be played in the background, and also to compare the two music generation models by listening tests and reconstruction loss.**

***Index Terms*—Wasserstein, autoencoder, music generation, pianoroll**

I. Introduction

In music generation the data processing plays a huge role. Where in image generation it is only needed to produce from one to three channels, in music generation there needs to be channel for each individual instrument. All those instruments have characteristic qualities such as pitch range, note velocities and note duration. Recently deep learning methods have become the number one choice in especially symbolic music generation. Multiple different architectures using for example autoencoders, LSTM-networks and generative adversarial networks (GAN) have been widely studied. Despite the very different architecture, all these models use data in a similar form. The musical data is often in a wave or midi format. Some neural network architectures can be trained using the data in wave format, eg. WaveNet [2], but all the aforementioned rely on symbolic notation, which means the data needs to be preprocessed. Previously pianoroll data format has become the state of art choice in music generation. [3] One pianoroll data point is basically a 2D matrix of certain time steps and pitch range, where a played note is marked as 1 and everything else as 0. The music data is transformed to MIDI-format which is then read to pianoroll tensors, which are then fed to the model in training. The models try to learn if note is played at the given time step and if yes, then at what pitch. Autoencoders and GAN architectures that are using convolutional layers try to find hidden structures from the whole input matrix, while for example the LSTM- network tries to guess the next time step based on the previous ones.

This project focuses on two types of autoencoders, variational autoendocers and Wasserstein autoencoders, and is carried out using same approach as L.Angioloni et al. used in their study "Conlon: A pseudo-song generator based on a new pianoroll, wasserstein autoencoders, andoptimal interpolations.". [3] The traditional pianoroll format is incredibly lossy, as it is for example impossible to differentiate consecutive same pitch notes from one another. To tackle this L.Angioloni et al. presented a new pianoroll format named CONLON, where the pianoroll is divided into two separate matrices, one for velocity and one for duration. L.Angioloni et al. trained Wasserstein autoencoder using new curated data set that had been special made for their study, and introduced a method to generate smooth curving trajectories in to the autoencoders latent space.

First the two models are trained using the MIDI-dataset preprocessed to CONLON form. Then multiple clips of music are generated using different trajectory swirls in the latent space. Finally the models are numerically evaluated using reconstruction loss and the generated clips are played to a test audience for human-oriented evaluation.

II. Data set and Preprocessing

The data set used contains multi-instrument and polyphonic MIDI-parts of a complete songs. The data clips are around 7-8 seconds long and labeled according to the part they are played in the song. In the CONLON study the authors used multiple data sets and different genres to train models and to compare if the data set quality and genre effects the generated pseudo-songs, but here the focus is only in one data set, the Acid Jazz dataset ASF-4. [1] This data set has been produced in collaboration with artists to suit in particularly this type of music generation task. The data is read from the MIDI- files to Python environment using Muspy- library and preprocesssed before turning it into CONLON pianoroll format. Muspy- libary was easy and intuitive to use. It has functions to read clips from midi to a special Muspy music class, which consists of multiple subclasses, for example track class, which represents each instrument.

The data contains multiple clips across few complete songs. The instruments present are drums, bass, Rhodes piano and Hammond flute, from which all are not played simultaneously

[1] https://github.com/paolo-f/CONLON

all the time. As is visualized in figure 1 there are clips with 2,3 and 4 instruments. For the training the data of each instrument need to be compressed to a format of NxTx2, where T is the fixed length of pattern, N is the fixed number of pitches and 2 is channels for velocity and duration values. For one clip these matrices are stacked together so one complete input tensor is NxTx8 by shape. The time scale is quantized 4/4 bars at 1/32th to create fixed length patterns of 128. As can be seen in the figure 1 the pitch ranges for instruments are drums [36,58], bass [31,60], rhodes [36,83] and similarly hammond [36,83], so all of them can be transposed to range [0,55] by subtracting 30. In the velocity and duration channel, there is an event ON(n,t,v or d) if note is played at that time step. Otherwise the matrix has value 0 as is visualized in figure 2.

On top of the transposition of the instruments the data is also scaled. Scaling turned out to be the most challenging part of this project. The distribution of note durations is strongly leaning towards small values, which can be seen in figure 1. Due to the note duration data distribution the models were unable to learn when to play long notes effectively. Two different scaling methods were tested, normalization to [-1,1] and quantile transformation to [0,1]. When using basic normalization to range [-1,1] most of the values were packed to the negative end which lead the models to select only those small values. When using quantile transformation the data became a little bit less skewed, visualized in figure 3, and the models were able to pick up at least some longer notes. However even with the quantile transformation the models were unable to truly learn the long notes, which affects greatly to the coherence of the generated clips. This is definitely something that needs to be addressed in future work. It was also noticed in training that a smaller reconstruction loss was achieved with quantile transformed data.

III. Models

For the training the data is divided into training and evaluation set by 90/10. The models compared in this project are variational autoencoder VAE and Wasserstein autoencoder WAE, which both are constructed from two convolutional networks, encoder and decoder. The encoder takes in one clip in CONLON pianoroll form and maps that data to a latent space. For this project a three dimension latent space was used as suggested in the original paper. [1] This space is also easy to visualize once the model is trained. The decoder takes in a random vector that represents a point in the latent space and tries to reconstruct the data that was initially encoded to that latent point. For the generation of music only the decoder is used.

The encoder and decoder architectures for both VAE and WAE are the same. These two models only differ on how the loss is calculated. In VAE the loss function is called ELBO-loss where reconstruction loss is subtracted from the KL divergence.

$$Loss_{ELBO} = L_{rec} + D_{kl}(q(z|x)|p(z))$$

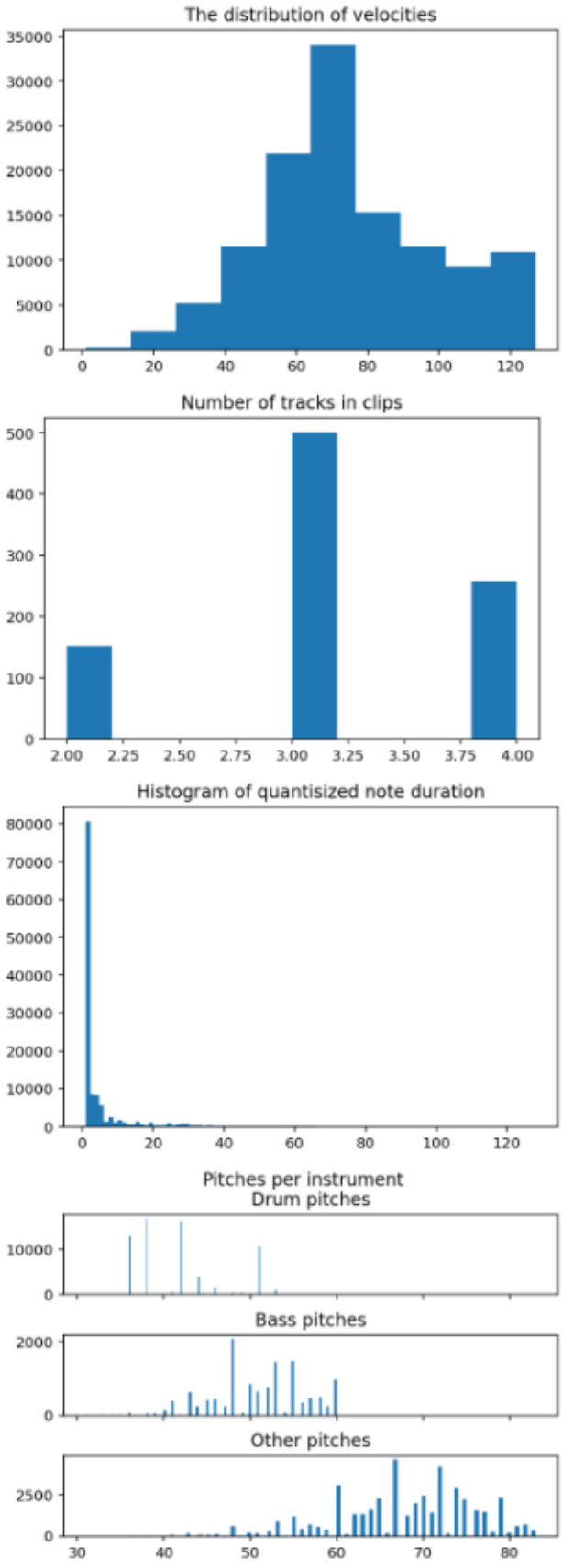

Fig. 1. Data qualities

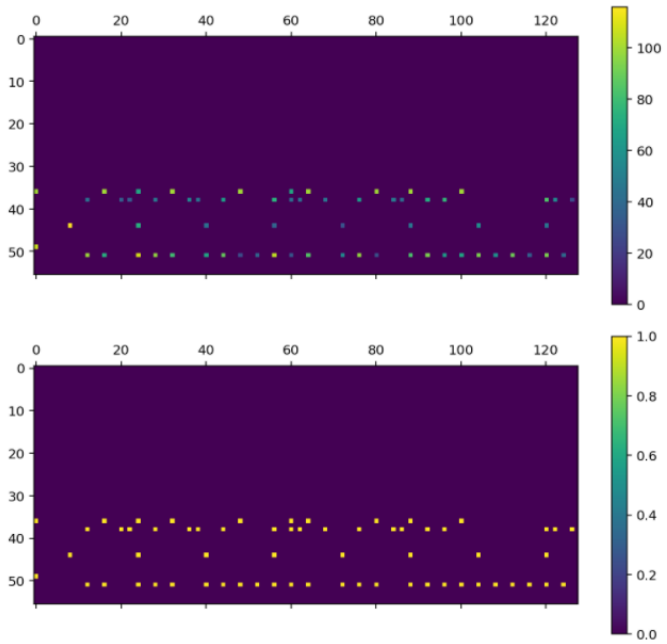

Fig. 2. Drum data matrices in CONLON pianoroll format for velocity and duration

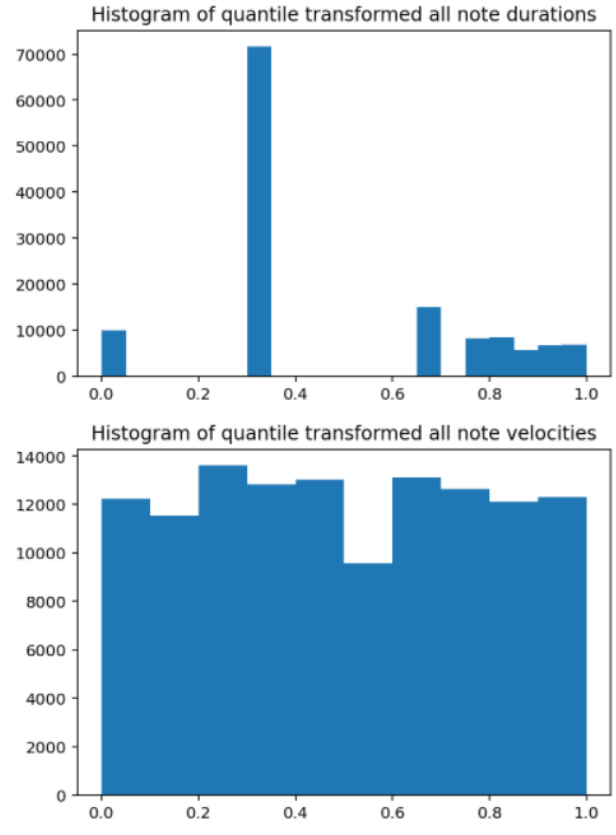

Fig. 3. Quantile transformed velocity and duration distributions

In the KL divergence term the random sample z is assumed to be drawn from normal distribution $Normal(0, 1)$, and it is pushing the latent values, posterior distribution, closer to normal as can be seen in figure 4. [4] [5]

In WAE the loss is calculated by the sum of reconstruction loss, c, and the Wasserstein distance between the model distribution and target distribution. [1]

$$WAE_{loss} = minE_pE_{q(z|x)}c(x, G(z)) + \lambda D(q_z, p_z)$$

Minimizing the the difference, D, of the prior q_z and aggregated posterior $q_z(z = E_p(q(z|x)))$ the expected values are pushed to match prior p_z as can be seen in figure 4. The total loss of WAE is sum of the reconstruction loss and the divergence D, which is calculated using mean maximum discrepancy MMD [6] and gaussian prior p_q. [2] As a result this enables different latent points to be further apart and so give better reconstruction. [7]

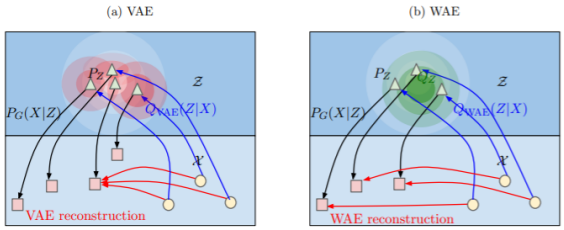

Fig. 4. VAE and WAE reconstruction visualized. Both models try to minimize reconstruction loss and discrepancy between the P_z and $Q(Z|X)$. VAE tries to match every $Q(Z|X = x)$ point (red balls) to P_z which is represented as the white area. In WAE the continuous Q_z is forced to match P_z, which gives the latent points possibility to be further away from one another. [7]

Originally WAE was introduced to tackle the blurriness of VAE and the issue that due to randomness in its structure VAE would generate different outputs from the same latent points. This can be seen in figure 4 where the VAE regularizer forces the red areas to cross each other in the P_z white area which complicates the reconstruction. [8] [9]

The models were built using Pytorch and their architectures are illustrated in figure 5. Both models have similar encoder and decoder architectures which follow the basic convolutional neural network scheme. [10] The hyperparameters such as learning rate, loss weight factors, batch size and number of epochs, were set using cursory grid search and the reconstruction loss was calculated using mean square error (MSE) between the input and output tensor. In the original paper the authors used Tensorflow in which the convolutional layer has ”same” padding argument, which automatically pads the input tensors symmetrically. This feature is missing in Pytorch so the padding had to be manually set in every layer. One extra ”padding layer” was added before the last convolutional layer in decoder in order to get the desired output shape. The final zero padding is only applied to bottom and right side of the tensor as the output is one off on those dimensions.

[2]https://github.com/ZongxianLee/MMD$_Loss.Pytorch$

ENCODER

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 28, 64]	16,416
BatchNorm2d-2	[-1, 32, 28, 64]	64
Conv2d-3	[-1, 64, 14, 32]	131,136
BatchNorm2d-4	[-1, 64, 14, 32]	128
Conv2d-5	[-1, 128, 7, 16]	524,416
BatchNorm2d-6	[-1, 128, 7, 16]	256
Conv2d-7	[-1, 256, 4, 8]	2,097,408
BatchNorm2d-8	[-1, 256, 4, 8]	512
Flatten-9	[-1, 8192]	0
Linear-10	[-1, 3]	24,579

Total params: 2,794,915
Trainable params: 2,794,915
Non-trainable params: 0

DECODER

Layer (type)	Output Shape	Param #
Linear-1	[-1, 1, 28672]	114,688
ConvTranspose2d-2	[-1, 128, 14, 32]	2,097,280
BatchNorm2d-3	[-1, 128, 14, 32]	256
ConvTranspose2d-4	[-1, 64, 28, 64]	524,352
BatchNorm2d-5	[-1, 64, 28, 64]	128
ConvTranspose2d-6	[-1, 32, 56, 128]	131,104
BatchNorm2d-7	[-1, 32, 56, 128]	64
ZeroPad2d-8	[-1, 32, 57, 129]	0
ConvTranspose2d-9	[-1, 8, 56, 128]	16,392

Total params: 2,884,264
Trainable params: 2,884,264
Non-trainable params: 0

Fig. 5. The encoder and decoder architecture.

IV. Training and Postprocessing

The models were trained using multi step learning function to decrease the rate as both models were noticed to be easily overfitted. The optimal learning rate to start the training was 0.0001 for both the models. With 0.001 the models got stuck to local minimum and the loss did not decrease. While training the VAE the learning rates were cut in half at 20,100 and 500 epochs and in WAE training at 100, and 900 epoch marks. This prevented major overfitting and retained smooth reconstructions all over the latent space. The VAE model was not as prone to overfitting as the WAE model. This could be due to the fact that in WAE latent values were fed straight from the encoder to the decoder without any randomness in between.

The divergence coefficients for both models were selected using trial and error. It was noticed that if the value of KL-divergence was not restricted in ELBO loss, the training created completely homogeneous latent space where every point generated same reconstruction. By using $\lambda = 0.1$ the KL-term was able to cluster similar clips correctly to the latent space. If $\lambda = 0$ the point clusters in latent space would not be pushed to normal form (range -1,1), but rather be randomly shaped and located in the latent space. For the WAE the clusters were always in nice spherical area, but with the λ coefficient the range of the point cloud be adjusted. With trial and error $\lambda = 100$ was selected. When using smaller λ value points in latent space covered a lot larger space, almost -15 to 15. And when using large λ like 500, the points were pushed closer to range -1 to 1.

Batch size was also selected using trial and error for VAE $b = 10$ and WAE $b = 26$. By using smaller batch size both models learned faster and learned more details from the clips. For example longer note duration in generated clips were more present in clips that were trained using smaller batch size. Also on the other hand training with too small batch size often led to fast overfitting.

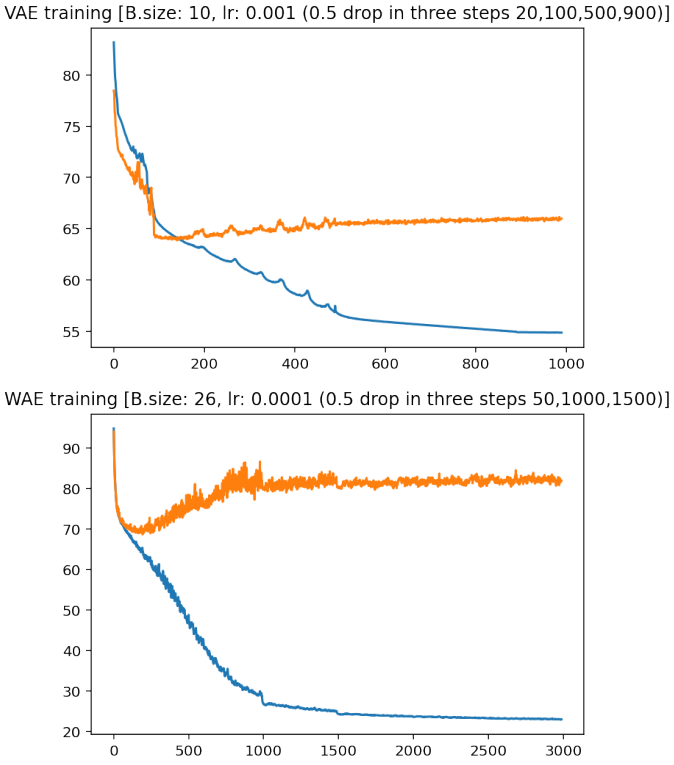

Fig. 6. Training (blue) and evaluation (orange) loss of both VAE and WAE models.

In figure 6 is presented the reconstruction losses of both models, the blue line representing training loss and orange evaluation loss. As is visible there is minor overfitting in the WAE model but as the evaluation loss does not massively increase it can be tolerated. Out of both models the VAE was much easier to train as it did not overfit as easy as the WAE. However the reconstruction were noisy and after postprocessing lead to clips with almost only notes with duration 1. The WAE on the other hand was able to learn a bit more longer notes, but struggled with the overfitting.

For postprocessing the authors presented a transformation similar to gamma correction function . [1] to get correct velocities for each instrument. The reconstructed clips were first inverse transformed and notes with durations below 1 were discarded. Then by using the correction transformation we could find the correct velocities as follows:

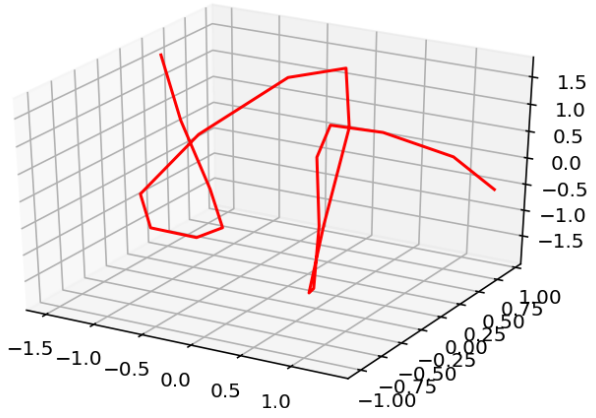

Fig. 7. Swirl for the VAE model

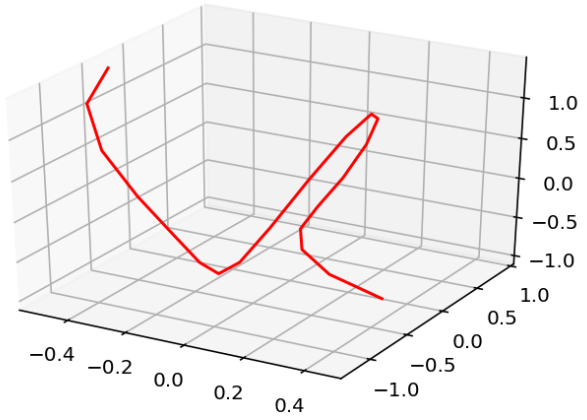

Fig. 8. Swirl for the WAE model

$$v_c(t,n,i) = \left\lfloor 127\left(\frac{v(t,n,1)}{127}\right)^{\frac{1}{\gamma_i}} \right\rfloor$$

V. Generating pseudo-songs

One latent point (x,y,z) represents approximately 8 seconds of music after its decoded with the decoder and postprocessed. During training the latent space should have become homogenous, i.e. by selecting any point from that space a nice reconstruction should be the output of the decoder. This means that longer clips with smooth musical transitions could be generated by selecting multiple points along a line through the latent space. In the original paper and in this project those lines were created using periodic complex-valued parametric function. The used function was

$$f(t,a,b,c,d) = e^{jat} - \frac{e^{jbt}}{2} + \frac{je^{jct}}{3} + \frac{e^{jdt}}{4}$$

where a, b, c, d were selected randomly so that the trajectory was inside the most dense latent space created by training data.

For the listening test a total of three swirls, out of which two are presented in figures 7 and 8, were generated for both of the models. The parameters were chosen manually to keep the curls in latent bounds while trying to cover wide range of different areas. In order to secure smooth transitions the distance between two consecutive points was kept as small as possible. Long distances between two points were detected as abrupt changes in the generated songs as the trajectory switched from one area to another.

VI. Comparison and Conclusion

By looking at the loss plots in figure 6 it is obvious that both models suffered from small overfitting. The loss refused to decrease after certain loss level, but neither did the evaluation loss increase too much. The overfitting had no major negative impact on the generated clips, on the contrary it brought up more longer notes. Out of the two models WAE was more inclined to overfitting, hence a special care was needed when tuning the hyperparameters.

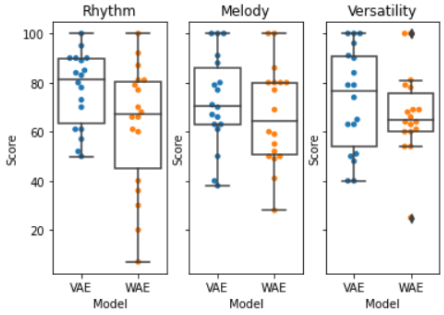

Fig. 9. The listening test results. Different answers are plotted with scatter plot. The box represents the standard deviation of the answers.

The output tensors from the VAE model were expectedly noisier than the ones out of WAE. However most of that noise was discarded in the post processing stage so it is not present in the listened clips.

In table I is presented the average reconstruction loss on the whole train and test set using mean error (L1) and mean squared error (L2). The train set reconstruction for WAE is obviously lower than VAE's due to the overfitting. Interestingly in test set reconstruction there is much smaller difference between the VAE and WAE. This indicates the similar generation potential between the models.

In addition a listening test was performed on a randomly selected group of family members and friends. The participants were diverse group with very different amounts of musical background. Those results are presented in figure 9. In this listening test the participants were blindly asked to evaluate 60 second clips from both models by their rhythm, melody and versatility. In every category also a reference clip of only random notes was played. A total of 9 persons between the ages of 25 and 57 took the test.

According to the test results, the participants evaluated the VAE clips higher in all categories, which can be seen in figure

TABLE I
THE AVERAGE RECONSTRUCTION LOSSES FOR BOTH MODELS ON TRAIN AND TEST DATA SETS

	VAE L1	VAE L2	WAE L1	WAE L2
Train set	154.1	54.9	76.4	31.4
Test set	168.2	67.1	161.6	83.1

9. Same phenomenon was also noticed when the results were transformed to binary VAE vs. WAE. Majority evaluated VAE to be better. The answers distributed evenly across gender and age. However there exists some answers where the decision was done clearly by random where the random reference clip received the best score. Also some answers gave every clip 100 in one category, so the results are not perfect.

All in all the aim in this project was to follow similar path as Anglioni T. et al. set in their CONLON study. Although the model architecture and data set were similar, I was not able to receive as good generative results as they did. The WAE should have been less overfit and have more accurate reconstruction. The missing long notes especially affect negatively to the listening experience, which is potentially the reason, why standard VAE got better score in the listening test. One major difficulty that needs to be addressed in future work is the scaling or normalization of the data. I believe that was the most significant factor to the issue that the models did not learn the long notes. The melody and rhythm was good for both models but the lack of long notes made the generated clips sound automated. The CONLON pianoroll representation was easy to implement and it worked really well with the convolutional networks in both models. One interesting experiment would be to couple it with LSTM- networks and see if those would also be able to create accurate and coherent pseudosongs.

REFERENCES

[1] L. Angioloni, T. Borghuis, L. Brusci, and P. Frasconi, "Conlon: A pseudo-song generator based on a new pianoroll, wasserstein autoencoders, and optimal interpolations." 10 2020.

[2] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," 2016.

[3] S. Ji, J. Luo, and X. Yang, "A comprehensive survey on deep music generation: Multi-level representations, algorithms, evaluations, and future directions," 2020.

[4] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2014.

[5] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A hierarchical latent vector model for learning long-term structure in music," 2019.

[6] M. A. M. Binkowski, D. J. Sutherland and A. Gretton, "Demystifying mmd gans," 2018.

[7] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf, "Wasserstein auto-encoders," 2019.

[8] J. Duda, "Gaussian autoencoder," 2019.

[9] P. K. Rubenstein, B. Schoelkopf, and I. Tolstikhin, "On the latent space of wasserstein auto-encoders," 2018.

[10] L. M. A. Radford, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2016.