Applying WaveNet on frequency data

Luzi Sennhauser
Christoph Dehner
Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—This paper introduces a new attempt to generate audio by a deep neural network based on a stack of dilated convolutional layers. It adopts the network architecture from the paper "WaveNet: A Generative Model for Raw Audio" to operate in the frequency space. When applied to piano recording, the network is able to learn low basis frequencies, but struggles to grasp and generate sound features of dynamic and well-sounding audio. Nevertheless, different attempts of normalization and data preprocesing yield conceptual insights in the functioning of the WaveNet architecture.**

I. Introduction

In 2016, [1] introduced WaveNet, a generative model for raw audio achieving astonishing results in learning sound features and generating harmonious audio. Trained on piano music, highly dynamic piano music varying smoothly in melody, volume and speed could be generated.
The aim of this paper is to enhance the WaveNet architecture and apply it to a more feature-oriented representation of input data. Instead of performing a classification task on raw audio, the network is adopted to operate on the frequencies of the audio data, processed by Fourier transformation.
As training data in the experiments, piano samples from the MusicNet dataset, a collection of 330 freely-licensed recordings of classical music are taken. The music is recorded with 44100 samples per second. The data is publicly available on the Internet [2].
In particular, the following topics are covered:

- Based on WaveNet from [1], a network architecture to generate music is introduced operating on the frequencies of audio input.
- This new network is trained and evaluated on piano music from the MusicNet dataset. Furthermore, optimization approaches as data normalization and variations in preprocessing the data are included in the experiments.
- Results from the evaluation are analyzed in detail. In the context of the introduced network, advantages as well as weaknesses are discussed.

The source code of the experiments can be found on GitHub.[1]

II. Models and Methods

The following section describes the underlying model of the network used in the experiments of this paper. As core network the WaveNet architecture with a stack of dilated causal convolutional layers is taken. Hereupon, the data input layer is changed conceptually.

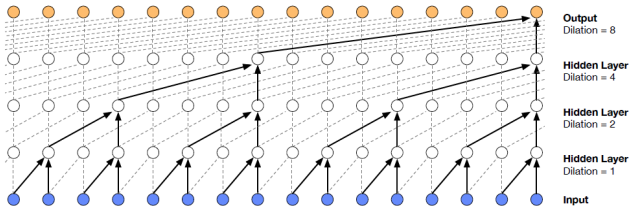

Figure 1. dilated convolutional network

A. WaveNet architecture

Given a sequence of amplitudes, WaveNet predicts the subsequent amplitude. This task is done in a classification setting: The amplitudes are quantized in 256 classes [3]. The input of the WaveNet network is a sequence of such quantizied amplitudes, each encoded as a 256-dimensional one-hot vector. Performing a softmax-function in its last layer, the network outputs a probability distribution $P(x_{i+1}|x_1, ..., x_i)$ in the ith step. The amplitude x_{i+1} of the next step is then sampled according to that distribution. To model the the dependency of an audio sample on previous samples, WaveNet uses a stack of dilated convolutions. Figure 1 depicts this architecture exemplarily for such a network of depth 4. By using exponentially growing dilations in depth, a network with n dilated convolutional layers can observe the last 2^n input values to predict an output.
Training WaveNet is done similar as an auto-encoder: The ith amplitude of the input data is predicted by only using previous input data and taken as label for the loss calculation and backpropagation. Thus, training only requires the input data and can be done in parallel. Against that, the generation of a audio sample has to be processed sequentially, since the ith step relies on the output of the $(i-1)$th one.
As a basis for WaveNet, the implementation of Igor Babuschkin is taken [4]. The original WaveNet paper mentions only very few coefficients for how their network is configured. Therefore reproducing the paper exactly is very hard. The following numbers are assumptions mainly

[1] https://github.com/sennluzi/tensorflow-wavenet

based on the experience of Babuschkin's implementation. In the following experiments, the dilations are 2^n for n taking the values $(0, 1, ..., 8)$ and are repeated 5 times. This leads to 50 hidden dilation layers. The convolution filter size is set to 2 and the number of channels of the residual and the dilation is 32.

B. Data preprocessing

To extract frequencies, the raw audio data consisting of pressure measurements are divided into chunks of size 2048, which are then transformed into the frequency space using FFT. Therefrom, the first 150 coefficients are taken as dominant frequencies for the subsequent training. To ensure that audio features are transformed smoothly, two neighboring raw audio chunks overlap in a few samples, described by a variable $stride$. Equation 1 summarized the frequency extraction of the i^{th} chunk compactly.

$$\text{frequencies\_coeffs}_i = FFT(\text{inputData}[i \cdot stride : i \cdot stride + 2048])[:150] \quad (1)$$

Considering real and imaginary parts of the complex Fourier coefficients separately, each chunk of 2048 raw audio samples is transformed into 300 real frequency values.
To decrease the dimensionality even further, principle component analysis (PCA) is used to reduce these 300 frequencies to 100 values. These 100 PCA coefficient are considered one tone and fed into the network as one time step.
Whereas in the original WaveNet architecture, 16000 samples represented on second of audio, the adopted network only needs to process $44100/2048 \approx 22$ samples per second. However, this change infers a fundamentally different approach of audio training and generation, described in the next section.

C. Evaluation of the preprocessing

The following analyses the amount of information which is lost by the compression steps of the previous session. Figure 2 plots the explained variance of the first 250 eigenvectors during the The vast majority of the variance of the frequencies is captured by the first 100 PCA components. Thus, PCA is a reasonable measure to reduce the 300 dimensional data from the FFT to 100 PCA components.
The best way to evaluate a compression procedure is to compare a compressed and afterwards reconstructed input to the original data. Figure 3 compares exemplarily the recorded raw audio input of a music sample from the MusicNet dataset to a compressed and afterwards reconstructed version of the signal. The plot shows that the two signals are similar in their general behavior and only vary vary only in small details.

The same results can be observed by listening to the original and the compressed and reconstructed audio file: After preprocessing the audio contains some cracking noise, but nevertheless is the melody and the tone of the music well audible.
Summarizing, in neither by cutting after the first 150 FFT coefficients, nor by the PCA significant information of the original input is lost. Therefore, the described preprocessing leads to a decent compression of the audio signal.

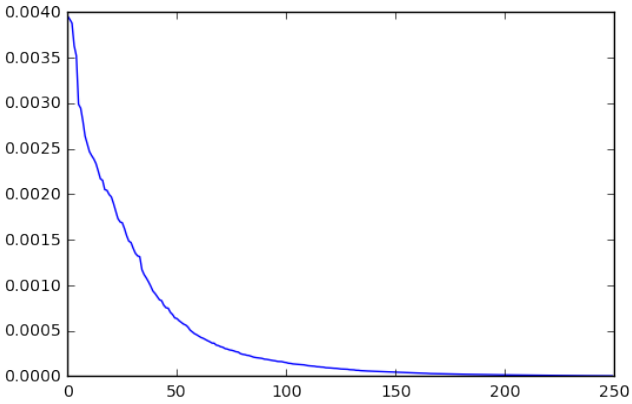

Figure 2. Explained variance of the i'th eigenvector in the principal component analysis (with i being the horizontal axis)

D. Network architecture and loss

Whereas the original WaveNet paper designed the network to perform a classification task, the adopted network aims to train and predict the frequencies of a audio sample directly. The network input during one time step is now not only one class describing the current tone, but a 100 dimensional vector representing the frequencies of the tone of time step

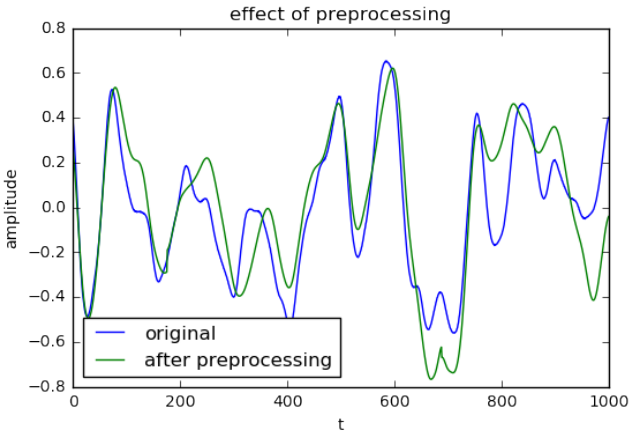

Figure 3. An example of how the output audio differs before and after compression.

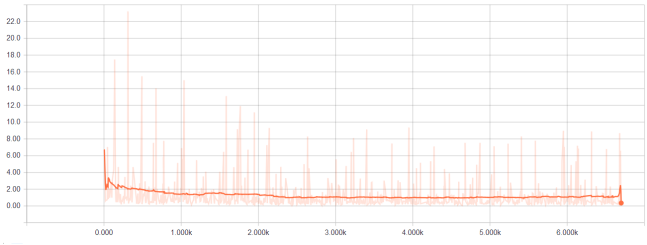

Figure 4. Training error

in a compressed way. Similarly, the output of the network is no longer a softmax layer but also a 100 dimensional vector. This change in architecture also affects the loss during training. Instead of a softmax cross-entropy loss the adopted network uses the mean squared error as loss function during training.

III. Results

The results of all experiments are summarized in a python notebook of the GitHub repository.[2] A suitable training session required approximately 6000-8000 training steps. Figure 4 plots the typical process of the error during training.

A. General results

Unfortunately, the adopted network could not perform as well as the original WaveNet architecture. All generated audio files stabilized on one distribution of frequencies after a few seconds. From then on the values of the frequencies did not change any more. Instead of learning the complete range of frequencies, which define the features of the audio input, the network could only learn some low basis frequencies being constantly present all the time.
Figures 5 and 6 compare the frequencies of a recorded audio piece from the input data and the frequencies of a generated sample. In the first 200 time frames, the frequency distributions look quite similar. However, afterwards the neural network is not able to produce frequency changes continuously, while a real piece of music observes a change every 20-60 frames.
Furthermore the original and generated audios differ in their frequency range. Whereas the original music piece only consists of frequencies up to 32, the generated audio contains frequencies up to 105. This shows that the network has difficulties to decide which frequencies are important and cannot produce as compact and compressed audio output as it is achieved by using FFT and PCA on recorded audio data.
Nevertheless, the experiments of this paper can be analyzed more detailed. Varying the stride and including normalization yielded conceptually different results.

[2]"evaluate_results.ipynb", available in the subfolder "notebooks"

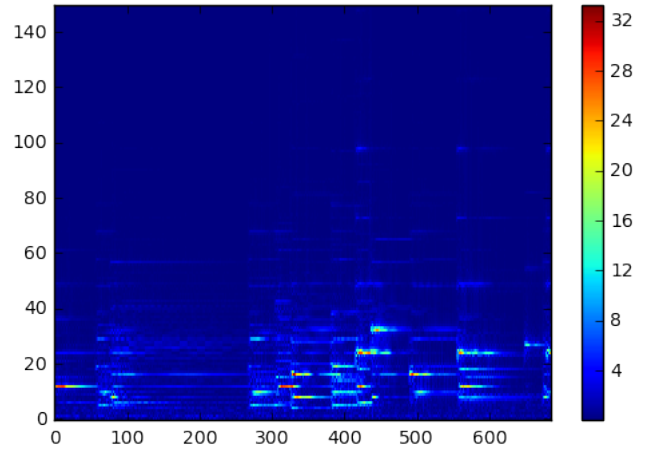

Figure 5. Frequencies of a piano piece used for training (Recording 2322 from the MusicNet dataset).

B. Striding

Overlapping chunks of input data turned out to be essential for learning audio features properly. With a stride of 512 (resulting in an overlap of 75% of neighboring data chunks) results with dynamic volume changes and frequencies within the range $[0, 105]$ within the first seconds of the generated audio could be achieved. Figure 6 visualizes the frequencies of the first 30 seconds of the output. In contrast to that, with a stride of 2048 (meaning no overlap) the network only grasped really frequencies in the range $[0, 1.6]$ producing no volume or frequency dynamic in the generated audio at all.

C. Normalization

A second conceptual attempt during the experiments was to normalize the PCA results before the training. The aim of this additional preprocessing task was to balance the contribution of all 100 PCA dimensions to the loss, hoping that all frequencies are learned more equally. In a first attempt, data normalized to zero mean and unit variance turned out to produce too small loss for training. With vanishing gradients the network could not learn at all. Therefore, in a second attempt the input data was normalized to zero mean and an equal variance of all dimensions of 10. With this specifications, the network was able to learn as well as without normalization.
As intended, the normalization helped to learn higher frequencies. The generated results with this setting contained more different frequencies from the range $[0, 360]$. But the generated audio still converged to a constant distribution of frequencies after a few seconds.

IV. Discussion

The approach of this paper could be promising due to the following reason: The original WaveNet paper's network has to learn the behaviour of sound consisting of repetitive

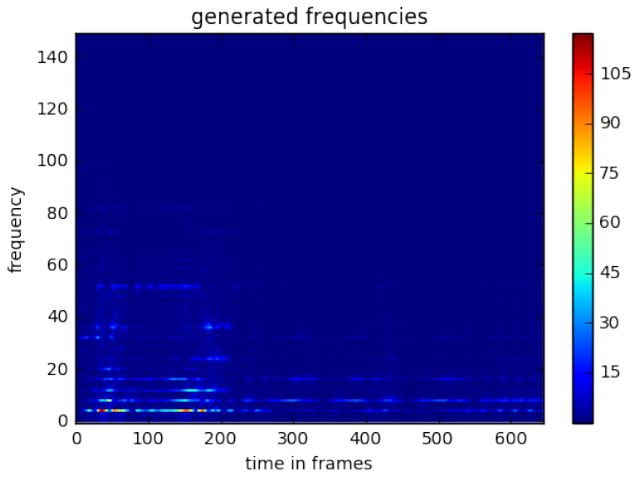

Figure 6. Frequencies of a generated audio piece, model trained on the MusicNet dataset with $stride = 512$ and no normalization.

oscillations. With transforming the signal to the Fourier space, we take this burden from the neural network. The network can focus on data which is more related to the final tone. For example while a tone is the same for a certain time, the output of the network is also the same.
But it might be exactly this phenomenon which makes good music generation impossible. During training, the network might do really well only staying on one and the same pitch, because changes happen in too rare occasions to influence the overall loss.
Future work based on that paper, one could try to include the rate of changing tones into the loss function and to punish a network when producing too many times in a row a similar output.
There are two reasons one could think of, why this approach's result does not match its expectations:

- The original paper quantizes the amplitudes and can then use a softmax layer as a last layer of the network. In this new approach, we need continuous values (PCA coefficients). Since the results are much worse than with WaveNet, the missing quantization (and connected to it the missing softmax layer) might be the reason behind this phenomenon.
- The original WaveNet has a probability distribution as output. The exact amplitudes are then sampled according to that distribution. In the approach presented in this paper, one relies on continuous values, which are directly calculated (no probabilities included). In future work, one can change the model to produce a probability distribution of the output values.

It has been shown that the stride is of great importance for the strength of the frequencies. Strong frequencies are interpreted as the network being confident in its prediction while weak frequencies are a sign of uncertainty. When the training timesteps do not overlap (stride = window size), the network produces much weaker frequencies. But a big stride might also lead to what we experience as not-changing frequencies. With a small stride most of the data chunks are the same as the previous one (when the length of a tone is much longer than one time unit). With larger strides, the frequencies change abruptly from one window to another. When a change in frequencies occurs on an dataset with smaller strides, the absolute value of the difference of frequencies is much smaller.
Therefore one can assume that with changing the setting (stride, but also window size, dilations, normalization etc.), also this approach might produce good results.

V. Summary

With WaveNet a very promising music generation network is presented. It relies on predicting amplitudes when the preceding amplitudes are given. In this paper it is investigated whether transforming the data to the Fourier space improves the predictions. After transforming, the dimensionality of the audio file is vastly diminished. But at the same time, the ability to quantize the network's outputs is lost. Unfortunately the network could not return as good results as the original WaveNet paper. With the adopted network architecture, the generated audio files rapidly converge to one steady tone, while changes of pitches is not learned. However, by the approaches and discussions in the paper yields conceptual insights and motivate further investigations.

References

[1] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016. [Online]. Available: http://arxiv.org/abs/1609.03499

[2] "Musicnet dataset," https://homes.cs.washington.edu/~thickstn/start.html, accessed: 2017-01-16.

[3] R. G. ITU-T and I. Switzerland, "711. pulse code modulation (pcm) of voice frequencies," *International Telecommunication Union, Geneva, Switzerland*, 1988.

[4] I. Babuschkin, "A tensorflow implementation of deepmind's wavenet paper," https://github.com/ibab/tensorflow-wavenet, 2016.