MODIFICATIONS TO SING, THE SYMBOL-TO-INSTRUMENT-NEURAL-GENERATOR

Stéphane Thunus, Jeffrey Thomsen, Daniel Meyer

Technische Universität Berlin
Straße des 17. Juni 135
10623 Berlin

ABSTRACT

***Index Terms*—** Machine learning, Neural Syntheziser, SING, CQT, ELU

The following paper examines possible improvements that specific activation and loss functions bring to the neural audio waveform generator SING. As activation function, the original rectified linear unit (ReLU) is replaced by the exponential linear unit (ELU). Because of its self-normalizing properties and increase in convergence speed, a better loss is expected, which should result in more realistic audio samples. Concerning the loss function, the original Short-Time Fourier Transform (STFT) is replaced by other spectral representations like the Constant Q Transform (CQT) and the Mel-frequency cepstrogram (MFC). Being closer to the human perception of pitch, they are expected to improve the perceived quality of the neurally generated audio samples.

Though in the autoencoder and fine tuning part of SING ELU leads to better convergence, in the LSTM network it performs worse than ReLU. The fact that the resulting audio samples do not reach the quality of the ReLU-models leads to the conclusion that the LSTM network is probably the most important part for audio generation quality. Unfortunately, probably due to an implementation error, neither CQT nor MFC seem to work as part of the loss function and so no information could be gained in this regard.

1. INTRODUCTION

1.1. General introduction

Over recent years, "artificial intelligence" has become an absolute buzzword, leading deep neural networks (DNN), Autoencoders (AE), convolutional neural networks (CNNs) and recursive neural networks (RNNs) to find an application in most computerized fields of human activity. The classical problem tackled by artificial neural networks and machine learning in general is the classification of data, however, substantive amounts of research dwells upon the idea of neural data generation.

This paper aims to propose ameliorations to the SING technology (described in 1.3) which is a neural audio waveform generation model. The practical uses of neural audio generation are, amongst others, text to speech transformations and audio data augmentation for various purposes including musical samplers, game engines or foley data sets for film- or video-audio. Data augmentation leads to the generation of greater data sets from limited data, which is useful for sound effects that are expensive to record or produce, while real-time audio generation allows to save enormous amounts of storage space for game engines or musical samplers.

1.2. Existing methods and related work

Composition with machine learning preceded the attempt at generating directly audio waveforms. While initial attempts tried to computationally find a suitable set of rules, more recent works such as DeepBach [1] are successfully using deep learning. Based on the notes of the individual voices, the rhythm and metadata such as fermata, key or time signature, the artificial neural network (ANN) composes new chorales resembling Bach's original work.

Furthermore, Machine learning is successfully applied to text to speech conversion (TTS). While older models convert text to an audio representation, then use a vocoder to generate the synthesized speech, newer models based on convolutional networks, such as Deep Voice 3 [2], achieve more realistic results. In addition, the TimbreTron model [3] transfers timbre between instruments using a spectral representation of the signal in combination with a convolutional architecture. The Short-Time Fourier Transformation (STFT) is replaced by a Constant Q Transform (CQT) to match human hearing (see section 1.4.2).

All these approaches do not generate audio waveforms directly but audio representations, which require other ANNs like WaveNet [4] or SampleRNN [5] which generate audio-waveforms on a sample-by-sample prediction, which surprisingly leads to decent results, albeit computationally heavy.

1.3. Symbol-to-Instrument Neural Generator – SING

SING, the Symbol-To-Instrument-Neural-Generator used for this paper, generates the audio waveforms of the desired instruments directly, from an input vector containing instru-

ment, pitch and velocity information [6]. This is done by calculating frames of 1024 samples simultaneously, which is much faster than the sample-wise generation used by WaveNet or SampleRNN.
The dataset of choice is NSynth [7], a collection of 1006 instruments from high-quality commercial sample libraries, where each instrument has 88 pitches, or less if limited by its range, and 5 velocity levels. The samples are in mono and four seconds long at a sampling frequency of 16 kHz and a bit-depth of 16 bits. In addition, each sample is annotated with its source types (acoustic, electronic or synthetic), its instrument family (brass, keyboard, voice etc.), and a maximum of 10 note qualities (bright, distortion, reverb etc.). However, as aforementioned, SING only uses the basic instrument, pitch and velocity information.
The network architecture consists of four convolutional layers on top of an LSTM network and is trained in three separate steps. First, the convolutional network is trained with an autoencoder from which only the decoder part is used in the final model. The encoder part takes the target waveform as input and translates it into a sequence representation. Each sequence element represents an audio frame of 1024 samples with a hop size of 256 samples, thus the encoder performs a dimensionality reduction of the raw waveform. The generated sequence is then used as the target output of the LSTM network which has one cell per audio frame. Its inputs are the above-mentioned feature vector, corresponding to the audio sample used to train the autoencoder, and some additional information about each time step, which is not further discussed by the authors of the original paper. Once the training of the individual modules is completed, the LSTM network and the convolutional decoder are plugged together and fine-tuned end-to-end in a third training. Now, the feature vector serves as input and a waveform is directly generated. This training is performed with a loss function the authors call spectral loss, which is further described in 1.4.2 and is the subject of the experimental modifications of the paper.

1.4. Theoretical Background

1.4.1. Activation functions

Activation functions (= transfer functions) are mathematical functions applied to the sum of the node inputs to determine the output value, emulating the firing rate of a biological neuron. They are hyperparameters and thus have to be set before training and usually don't change once set.

Rectified Linear Units (ReLU)

The ReLU activation function is given as follows:

$$f(z) = max(z, 0) \tag{1}$$

ReLU are the successors of sigmoid functions types, correcting diminishing gradient among other problems encountered during neural network training. However, ReLUs are unbounded functions but neural networks layers work better on normalized data, which is why batch normalization is used [8]. Batch normalization is a z-standardization of the neuron weights of each layer, which can be thought of pre-processing the input of each hidden layer to ameliorate their training (ibid). The ELU activation function being a special case of the SELU family, allows the construction of Self Normalizing Networks (SNNs) [9], where some sort of normalization is done by the function itself [10].

Exponential Linear Units (ELU)

ELUs were introduced by Clevert et al. [10] as an amelioration of the Rectified linear Units (ReLU) family in terms of convergence speed and classification accuracy. It is given as follows (in our case a=10:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ a\left(e^x - 1\right) & \text{if } x \leq 0 \end{cases} \tag{2}$$

Like ReLUs, their identity in $\mathbb{R}+$ eliminates vanishing gradients problems encountered in deep neural networks with sigmoid functions [10]. As aforementioned, ELUs also normalize the weights by having saturated negative values, which pushes the mean unit activation towards zero, speeding up learning (reduced bias shift) and increasing accuracy [11]. This type of normalization is of interest, being of lower complexity than batch normalization. However, ELU have higher computational complexity than ReLU, since an exponential has to be calculated for negative values rather than being rectified to zero. Additionally, ELU have a noise robust deactivation state by saturating the negative values to the hyperparameter a [10], avoiding neuron deactivation.
Faster convergence means either that for a constant number of epochs (=model refinement iterations) the resulting model generates better samples or that in order to achieve the same quality of samples, the generating model needs fewer epochs, which means less training time.

1.4.2. Loss functions

Loss functions in machine learning measure how far off the prediction or the generated medium is from the ground truth. This function should encapsulate all relevant aspects of the medium to be generated or subtleties of the classification to be an accurate measurement tool.
Thus, the second proposed change to the SING architecture is the use of CQT and Mel frequency cepstrums, which, by being closer to human perception of sound, should yield results that are of better perceived quality.
The respective sample norms are squared element-wise to generate a power spectrum and passed through a logarithm to approximate the decibel scale of human hearing. The absolute value is used to lose the information about the relative

phase of the partials, which is not very perceptually relevant, thus would unnecessarily increase the loss. Furthermore, a small constant scalar ϵ is added to prevent the logarithm to tend towards $-\infty$ when its argument approaches zero. The value of each sample is thus:

$$l\left(x\right) := \log\left(\epsilon + \left|\left\{STFT,\ CQT,\ MFC\right\}\left[m\right]\right|^2\right) \quad (3)$$

The loss compared to the ground truth is then calculated as follows:

$$L_1\left(x, \hat{x}\right) = ||l(x) - l(\hat{x})||_1 \quad (4)$$

The rectilinear metric induced by the L_1 norm was preferred over Euclidian distances in order to increase the loss function sensitivity by avoiding the vanishing of small values due to squaring [6].

Short time Fourier Transform (STFT)

The STFT is a sequential application of a discrete Fourier Transform (DFT), which transforms time domain signals into frequency domain signals. The disadvantage for audio related purposes is that the frequency bins are equally spaced and thus are not representative to human hearing.

Constant Q analysis (CQT)

The Constant Q analysis is designed to represent audio in a way that mimics human hearing. This means a higher frequency resolution and lower time resolution in the lower frequencies, and the opposite in the higher frequencies. The signal is filtered in the desired bands and a variable size discrete Fourier transform (DFT) is done proportionally to the variable length of the window, meaning for example smaller segments for higher frequencies. As described in [12], it is defined as follows:

$$X\left[k\right] = \frac{1}{N\left[k\right]} \sum_{n=0}^{N[k]-1} W\left[k, n\right] x\left[n\right] e^{-j\frac{2\pi Q n}{N[k]}} \quad (5)$$

To illustrate, a DFT analysis with a Δf of 5Hz, would only have two bins to describe the octave between 10 and 20Hz, while having 2000 to describe the octave between 10kHz and 20kHz, while the CQT keeps the same number of bins per octave. Furthermore, the CQT analysis keeps a constant relative y-distance on Spectrograms, because of its proportional scale, which helps the analysis of the convolutional neural networks (CNN). The bin spacing can be calculated as follows:

$$\delta f_k = 2^{\frac{1}{n}}\ \delta f_{k-1} = \left(2^{\frac{1}{n}}\right)^k \delta f_{min} \quad (6)$$

Mel frequency cepstrum (MFC)

Similarly to the two transformations described above, the Mel frequency ceptrum analysis represents the power-spectrum passed through a log to emulate human amplitude perception. Furthermore, like the CQT, the frequency scale is adapted to correspond to the human frequency perception. This is achieved by applying a mel-filter scale that measures the perceived equal distances between pitches as well as using triangular filterbanks that emulate different regions of the basilar membrane. It is defined as follows:

$$MFCC_m\left[n\right] = \sum_{l=1}^{L} \log\left(\sum_{k=0}^{N-1} \left|tri_l\left[k\right] X\left[m, k\right]\right|^2\right) \cdot \cos\left(\frac{n\pi}{L}\left(l - 0.5\right)\right) \quad (7)$$

One main difference between the MFC and the previous transforms is that a discrete cosine transform (DCT) is used instead of a DFT. The DCT has the property of de-correlating the energy content of the overlapping frequency bins and tends to concentrate energy in the lower bins, which assures a greater compression of information.

2. METHODS

2.1. General

This research aims to compare the differences between the usage of ReLU and ELU as activation functions concerning convergence speed, loss minimization and quality of the generated instrument samples. Furthermore, the ameliorations brought by transformations based on psychoacoustical frequency scales are investigated, measured by the same aforementioned means.
To our knowledge, no further optimization like batch normalization or dropout is used during the SING training. The use of batch normalization would reduce the difference between ReLU and ELU, since ELUs are self-normalizing activation functions [10].

2.2. Training data and calculated models

Due to the limited available computational capacity, smaller models had to be calculated and the existing NSynth data set was reduced to a few selected instruments. In order to still have as much variety as possible, the three instruments keyboard_acoustic_007, mallet_acoustic_011 and vocal_acoustic_011 were selected. The mallet is mainly percussive, the voice mostly sustained, while the piano sound is both. Furthermore, the voice is especially well suited as a benchmark for perceptive quality, being what the human ear is the most sensitive to.
To be able to compare the different activation and loss functions and further spot effective combinations, the models

were trained with all possible combinations of the aforementioned instruments and functions: {Piano, Mallet, Voice, all three together} x {ReLU, ELU} x {STFT, MFC, CQT}, resulting in 4x2x3=24 different models. To increase statistical significance the piano-only-models were calculated 10 times. A second set of models was calculated, where the two activation functions were combined with data sets of different sizes. To increase variety, a new set of five instruments was used, consisting of brass_acoustic_001–005. The following combinations were calculated: {brass_1, brass_1–3, brass_1–5} x {ReLU, ELU} = 6 models. Due to time restrictions, these models could only be calculated once. No model with CQT or MFC loss functions were calculated because of technical difficulties as explained in later sections.
An overview of the number of models for different combinations of instruments and functions can be seen in table 1. To evaluate the perceived quality of the models, samples were generated for each model.
Each instrument has its own pitch range with five velocities each. This is of importance as no previously unknown pitches could be generated, despite counter indication in the original paper, which would have been of interest for model generalization measurements. For an overview about the instruments, their ranges and number of samples, see table 2.

Table 1. Number of calculated models and training samples for different combinations of instruments, training losses and activation functions

instr.	STFT		CQT		MFC		sampl.
	ReLU	ELU	ReLU	ELU	ReLU	ELU	
keys	10	10	10	10	10	10	440
mallet	1	1	1	1	1	1	440
voice	1	1	1	1	1	1	225
k+m+v	1	1	1	1	1	1	1105
brass1	1	1	–	–	–	–	170
brass1–3	1	1	–	–	–	–	680
brass1–5	1	1	–	–	–	–	1330

2.3. Activation functions

As explained, the exponential linear unit (ELU) is expected to increase model accuracy (reducing the loss) and convergence speed, thus creating higher quality samples or reduce the model training time. This is made visible by plotting the training loss for each model and activation function at every training epoch for each of the three training steps metrics (AE, LSTM, full model).

Table 2. Training data instruments with their pitch range, the number of training samples and generated samples

instrument	range	sampl.	gen. samp.
keyboard_acoustic_007	A,,–c”’’	440	89
mallet_acoustic_011	A,,–c”’’	440	89
vocal_acoustic_011	D–b♭”	225	52
brass_acoustic_001	G–e’	170	41
brass_acoustic_002	C,–e”’	325	41
brass_acoustic_003	F–f”	185	32
brass_acoustic_004	C,–e”’	325	41
brass_acoustic_005	C,–e”’	325	41

2.4. Loss functions

Though only one spectral representation – CQT, MFC or STFT – is used as training loss, the loss for the other two representations is calculated as well. This makes it possible to see how a particular loss evolves when another loss is used for training. For terminological disambiguation purposes, this paper arbitrarily defines ”training loss” as the loss function which is used to actually train the model in contrast to ”side losses”, which are calculated for comparison purposes. It is expected that during training the side losses are reduced as well. However, the details might be of interest – for example an STFT loss could be reduced faster using a CQT as training loss function rather than the STFT itself.
Similarly to the ReLU/ELU comparison, all training and side losses of all models will be written to a log file, which will then be used to evaluate and analyze the different loss functions.

3. RESULTS

3.1. Training Loss

Figures 1 to 3 show the side losses of the 10 piano models when training the auto-encoder, which is the first network of the model. The plots group the evaluation functions in STFT, MFC and CQT and compare their evolution over epochs for the different training losses.
To illustrate, on figure 1, the yellow lines show the evolution of the STFT loss when STFT/ELU is used as training function, while on figure 2 they show the evolution of the MFC loss function with STFT/ELU as training loss, only differing by their starting values.
It can be observed that there is only movement and convergence when the STFT is used as training loss, irrespectively of the plotted side loss, as MFC and CQT remain straight lines throughout all three plots.
Furthermore, on all three figures the STFT/ELU loss (in yellow) tends to start at higher values than other loss functions but then decreases in 3 to 4 epochs to the level of the other

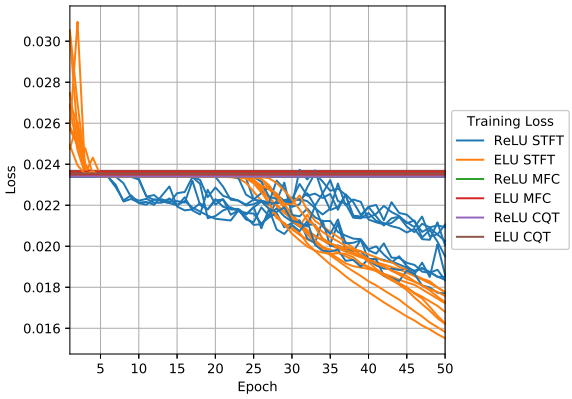

Fig. 1. STFT loss of the autoencoder for different training losses, 10 identical piano models

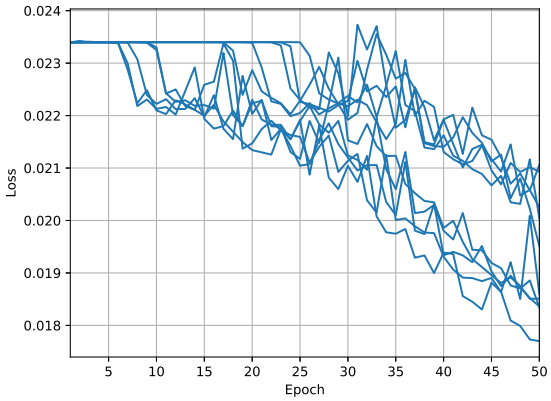

Fig. 4. STFT training loss of the autoencoder, ReLU activation, 10 identical piano models

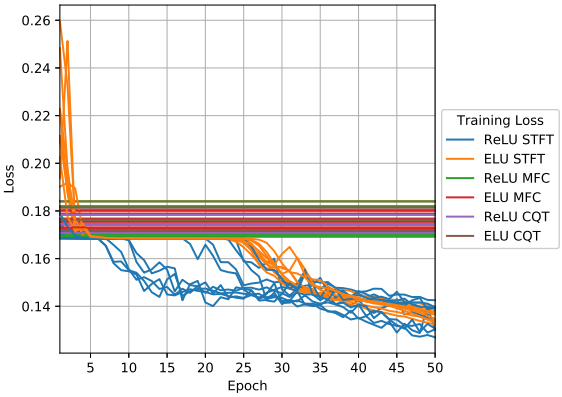

Fig. 2. MFC loss of the autoencoder for different training losses, 10 identical piano models

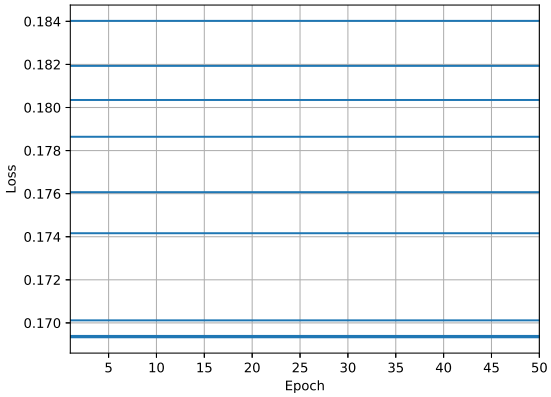

Fig. 5. MFC training loss of the autoencoder, ReLU activation, 10 identical piano models

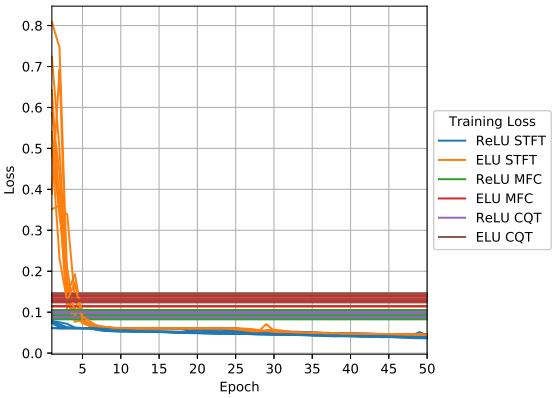

Fig. 3. CQT loss of the autoencoder for different training losses, 10 identical piano models

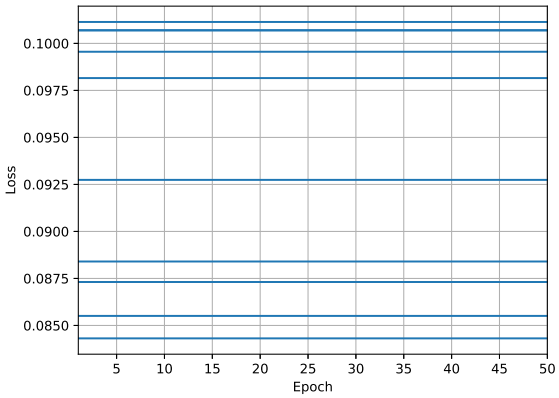

Fig. 6. CQT training loss of the autoencoder, ReLU activation, 10 identical piano models

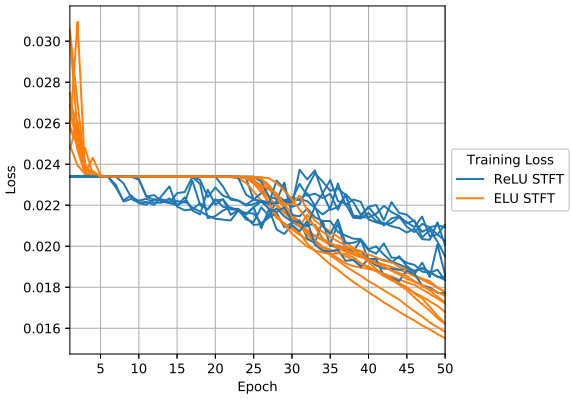

Fig. 7. STFT training loss of the autoencoder, comparing ReLU and ELU activation, 10 identical piano models

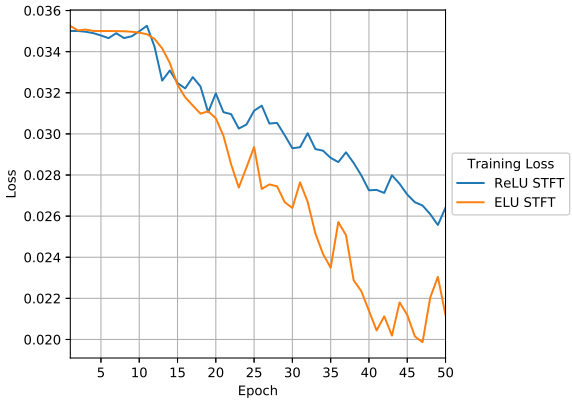

Fig. 10. STFT training loss of the autoencoder, comparing ReLU and ELU activation, 3 instrument model

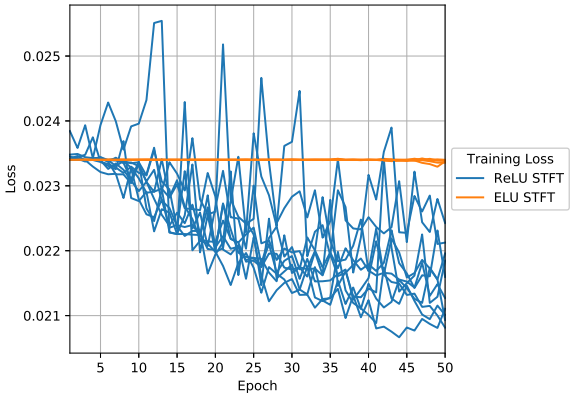

Fig. 8. STFT training loss of the LSTM, comparing ReLU and ELU activation, 10 identical piano models

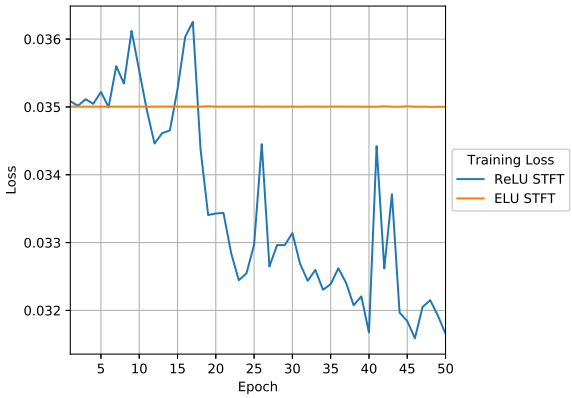

Fig. 11. STFT training loss of the LSTM, comparing ReLU and ELU activation, 3 instrument model

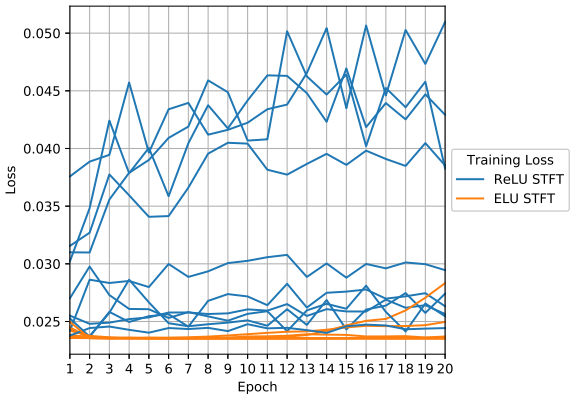

Fig. 9. STFT training loss of the fine-tuning, comparing ReLU and ELU activation, 10 identical piano models

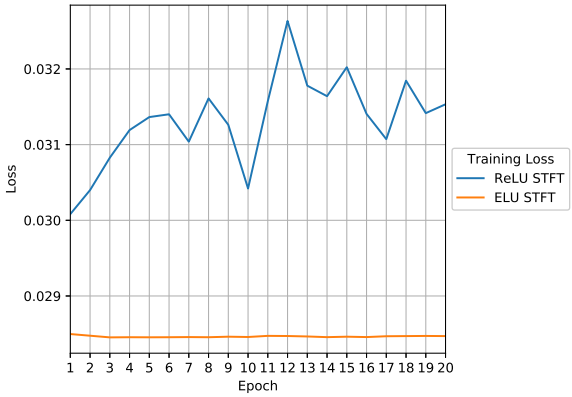

Fig. 12. STFT training loss of the fine-tuning, comparing ReLU and ELU activation, 3 instrument model

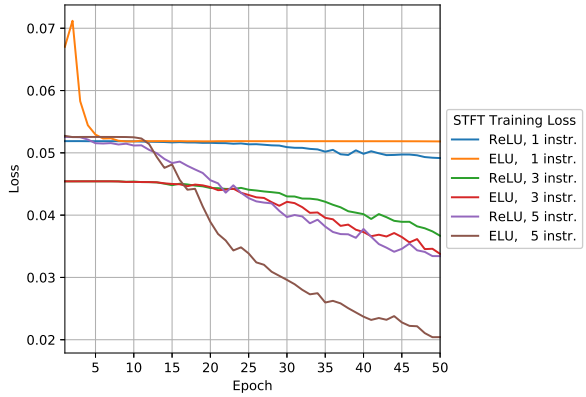

Fig. 13. STFT training loss of the autoencoder, comparing ReLU/ELU activation, brass models

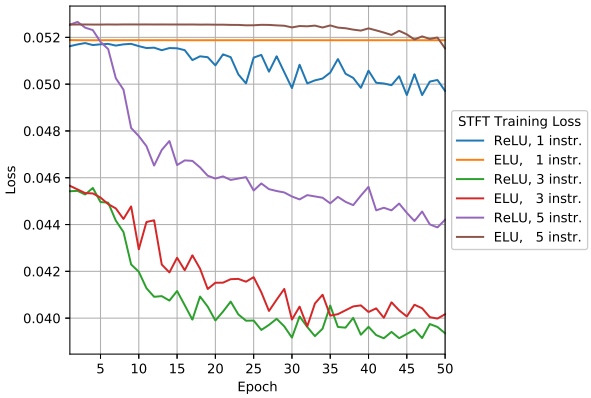

Fig. 14. STFT training loss of the LSTM, comparing ReLU/ELU activation, brass models

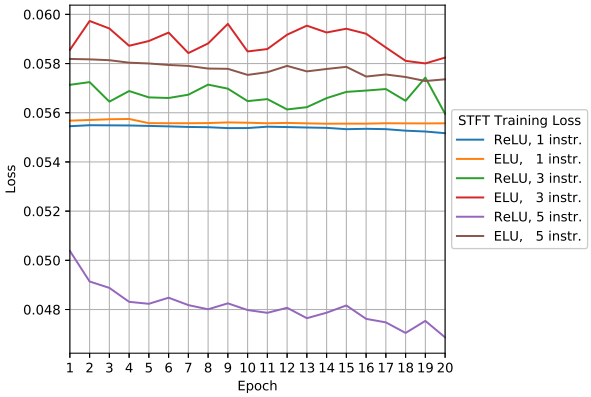

Fig. 15. STFT training loss of the fine-tuning, comparing ReLU/ELU activation, brass models

losses to then plateau for 20 epochs, after which (around epoch 25) there is a sudden continuation of the convergence. The ELU function converges in a more stable manner than the ReLU, which erratically converges, while also partially displaying the same plateau-ing behavior.
For the STFT side loss (figure 1), the STFT/ELU converges to lower points in almost all cases than the STFT/ReLU, which is not the case for MFC side loss (figure 2). It can further be observed that in the short term (less than 30 epochs) ReLU always lies beneath ELU.

Figures 4 to 6 are a training loss-only, ReLU-only version of the figures described above (1 to 3) for clarity on how training with the newly implemented MFC and CQT loss compares to the original STFT loss.

Figure 7 is the same as figure 4 but additionally displaying the STFT/ELU convergence to be compared with figures 8 and 9. Figure 8 displays the convergence for the second network that constitutes the SING model, which are LSTM cells. The ELU models roughly start at the same point as the ReLU Models but remain constant throughout the training, while ReLU losses show a distinct downwards trend and end up at lower points than their ELU counterparts. Figure 9 shows the final fine-tuning stage, where all networks of the model are put together and trained together. One can observe that the ELU losses systematically start at a lower level than the ReLU losses and remain relatively constant, while the ReLUs show an upwards trend (divergence).

Figures 10 to 12 display the three training stages of the SING model for the 3 instrument data set. The same trend as for the piano data sets is to be observed, which is that auto-encoders and the final stage converge faster with ELU loss functions, while the LSTM stage performs worse with ELU loss.

Figures 13 to 15 show the convergence of the different models trained with the brass data sets. As before, in the auto-encoder stage, the ELU loss outperforms the ReLU loss, except for the single instrument data-set, where the ReLU loss is slightly under the ELU loss. Figure 14 shows again that ReLU losses outperforms ELU losses for LSTM networks, while ELU losses outperforms ReLU losses in the final stage in figure 15.

3.2. Perceptive evaluation of generated audio samples

For the first set of models (piano, mallet, voice and their combination, see 1), three quality levels can be identified. Representative STFTs for each category can be seen in figure 16–18. They are keyboard samples with pitch = f♯' and velocity = 100. The according models used three instruments, with ReLU/CQT (no convergence) for the low quality example,

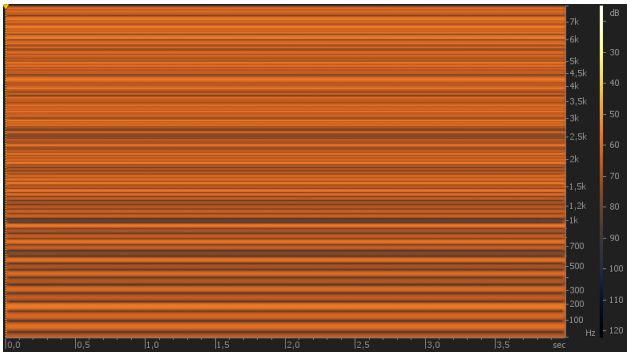

Fig. 16. STFT of a low quality keyboard sample

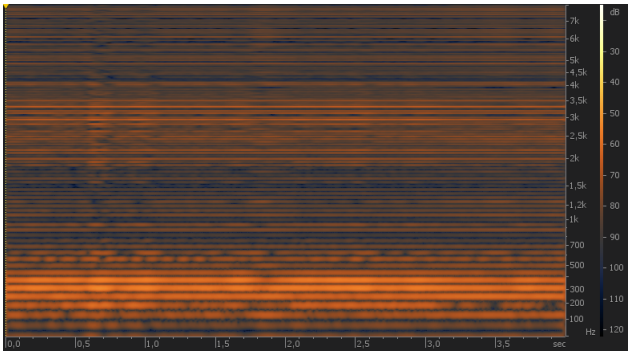

Fig. 17. STFT of a medium quality keyboard sample

ELU/STFT for the mid quality example and ReLU/STFT for the high quality example.

All samples generated by models using CQT or MFC belong to the low quality category, where the STFT consists of vertical, non-modulating bars with the same energy. The resulting sound is static tonal noise, which doesn't resemble the original keyboard samples in any way. Samples generated with the STFT/ELU combination mostly belong to the medium-quality category, where mild modulation and therefore adaption is visible. In the STFT models, the vertical bars have different energy content which slightly modulates over time. The resulting sound is amplitude modulated static noise with more tonal content, still not resembling the original keyboard samples.

The samples generated by the STFT/ReLU combination mostly belong to the high-quality category, which is substantially sonically superior to the others. In most cases, the fundamental frequency, as well as the harmonics are clearly visible, resulting in a correct perceived pitch. In some other cases, the pitch can't be correctly recognized, though there is a spectral structure. Most samples have a distinct temporal structure according to the original piano sample with an initial transient and decaying energy envelope over time. Velocity is, however, rarely recognizable and the samples sound electronic due to additional tonal noise.

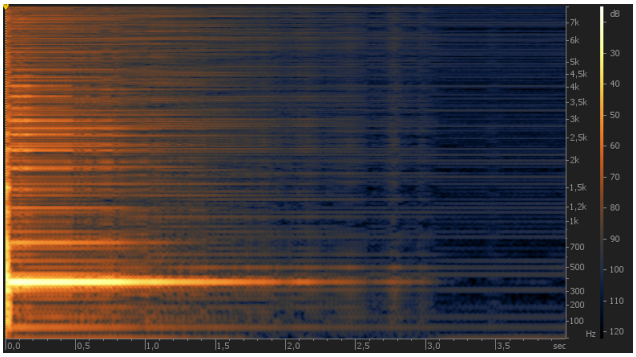

Fig. 18. STFT of a high quality keyboard sample

For the second set of models (between 1–5 brass instruments as training data, see table 1), the representative STFTs can be seen in fig. 19–22. They are samples of brass_acoustic_001 at pitch d' with a velocity of 100. Applying the same qualitative categories, the combination brass1/ELU (see fig. 19) results in low quality samples with no visible modulation in both frequency and time, and as a result only static, tonal noise. The brass1/ReLU-model (see fig. 20) already shows a clear time structure but still contains a lot of noise being a good medium quality sample, which might be put in the high quality category. All other models, which are brass1–3 and brass1–5 for both ELU and ReLU definitely produce high quality samples. Often, the pitch can be recognized, and the velocity perceived because of the different spectral energy in the harmonics.

Interestingly, the quality of the combination brass1–5/elu (see fig. 21) seems to be lower compared to the other three models, even lower than the combination brass1–3/elu, which uses less training data. From the three best models, the combination brass1–5/relu (see fig. 22) seems to have the edge over the other two. A rough overview of the quality of each model can be found in table 3.

Table 3. perceived quality of the generated samples for each model

instr.	STFT		CQT		MFC	
	ReLU	ELU	ReLU	ELU	ReLU	ELU
keys	+	–/○	–	–	–	–
mallet	+	–	–	–	–	–
voice	○/+	–/○	–	–	–	–
k+m+v	++	○	–	–	–	–
brass1	○/+	–	–	–	–	–
brass1–3	++	++	–	–	–	–
brass1–5	++	+	–	–	–	–

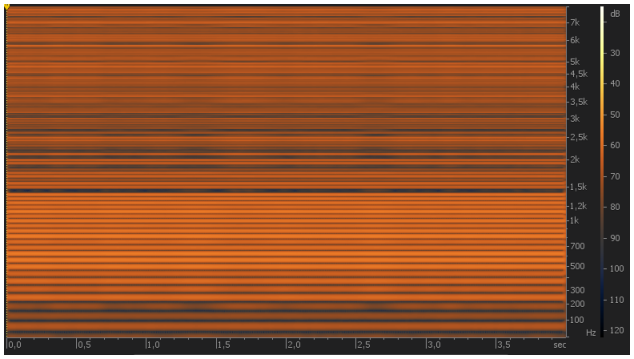

Fig. 19. STFT of a brass sample from a model using 1 brass instrument as training data and ELU as activation function

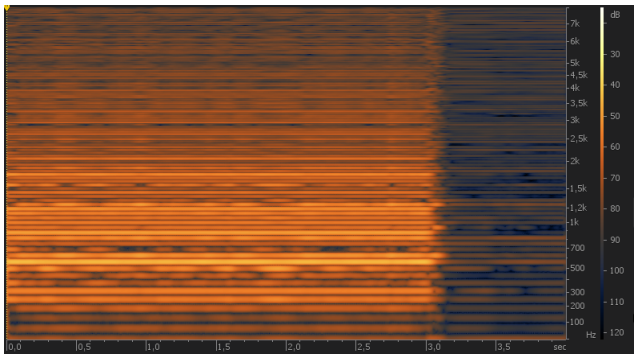

Fig. 20. STFT of a brass sample from a model using 1 brass instrument as training data and ReLU as activation function

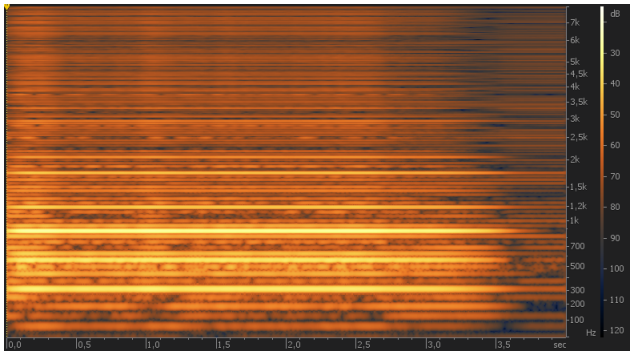

Fig. 21. STFT of a brass sample from a model using 5 brass instruments as training data and ELU as activation function

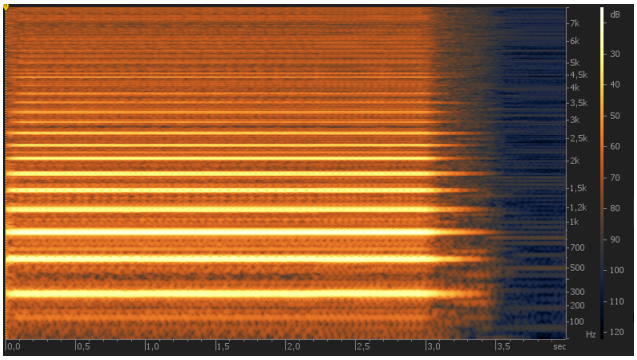

Fig. 22. STFT of a brass sample from a model using 5 brass instruments as training data and ReLU as activation function

4. DISCUSSION

Firstly, using MFC and CQT in the loss function did not work, as when used as training loss, one sees no convergence. Any side loss, even the STFT side loss, stays constant over all epochs. Also, none of the generated samples show any sign of modulation (see fig. 16) which would be a display an adaption to the training data. The fact that there is no change in loss shows that the program code not working properly. Otherwise, some movement should be visible, even if the loss does not improve.

Thorough review of our code modifications show this problem is probably due to transforming the original Torch tensors into Numpy arrays for computing the MFC and CQT time-frequency-representations with the respective LibRosa functions. While the transformation of the numerical values works as desired and the generated spectral representations are correct, issues are encountered when applying the loss function. This Torch function uses some additional metadata from the previous processing of the tensors (in our case the initial waveform is already in tensor form). It is however not possible to view, access or change this information and apparently it is lost in the process of transforming the tensor into a Numpy array and back, which results in the loss function not updating the weights used for training, thus making any comparisons to the initial STFT-based loss calculation impossible. Looking at the side losses in figures 1 to 3 lets us conclude that the loss calculation works on a numerical level. It is visible that MFC and CQT side losses also change through training, which shows that if the weights are updated correctly (using STFT training loss) and they are only used for evaluation, the expected development of the loss is shown in these as well. Conversely, training the models with MFC or CQT loss, also the STFT side loss is constant, emphasizing that the problem we encountered lies not in the calculation of the different spectral representations, but in the calculation of the training loss.

Comparing ReLU and ELU activation, it can be seen that ELU converges faster when training the autoencoder and fine tuning of the model, but seems to converge slower for the LSTM. When trying to connect this information to the quality of the generated audio samples, where ReLU models were superior to their ELU counterparts (see table 3), it seems reasonable to conclude that the LSTM is the most important part of the network, because a slower convergence in this part leads to a lower overall quality. The fact that ELU performs worse at the LSTM part yields question of how strongly all the other parts of the complex SING-network might be adapted to using ReLU as activation function, and if some of the code was consciously optimized to ReLU. In any case, it can be stated, that despite the better convergence in the autoencoder and fine tuning part, ELU does not improve the network at its original task, which is creating good sounding audio samples.

In future work, it would be interesting to see if ReLU is still superior to ELU with larger training data, or if this advantage will vanish. Also, more recently developed activation functions like Scaled Exponential Linear Units (SELU, see [9]) could lead to better results. But the main improvement idea was to replace the STFT in the loss function with a more musical spectral representation. Sadly, any questions on that matter remain unanswered. Thus, the most interesting and obvious task would be to investigate, how the authors' technical problem can be solved to be able to use CQT and MFC to calculate the training loss.

5. REFERENCES

[1] Gaëtan Hadjeres, François Pachet, and Frank Nielsen, "Deepbach: a steerable model for bach chorales generation," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1362–1371.

[2] Wei Ping, Kainan Peng, Andrew Gibiansky, Sercan O Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John Miller, "Deep voice 3: Scaling text-to-speech with convolutional sequence learning," *arXiv preprint arXiv:1710.07654*, 2017.

[3] Sicong Huang, Qiyang Li, Cem Anil, Xuchan Bao, Sageev Oore, and Roger B Grosse, "Timbretron: A wavenet (cyclegan (cqt (audio))) pipeline for musical timbre transfer," *arXiv preprint arXiv:1811.09620*, 2018.

[4] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[5] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio, "Samplernn: An unconditional end-to-end neural audio generation model," *arXiv preprint arXiv:1612.07837*, 2016.

[6] Alexandre Défossez, Neil Zeghidour, Nicolas Usunier, Léon Bottou, and Francis Bach, "Sing: Symbol-to-instrument neural generator," in *Advances in Neural Information Processing Systems*, 2018, pp. 9041–9051.

[7] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi, "Neural audio synthesis of musical notes with wavenet autoencoders," 2017.

[8] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[9] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter, "Self-normalizing neural networks," in *Advances in neural information processing systems*, 2017, pp. 971–980.

[10] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289v5*, 2016.

[11] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

[12] Judith C Brown, "Calculation of a constant q spectral transform," *The Journal of the Acoustical Society of America*, vol. 89, no. 1, pp. 425–434, 1991.