CS230 final project report : Differentiable digital signal processing

Project Category: Audio & Music
Project Mentor: Hanson Lu

Name: Jung-Suk Lee
SUNet ID: robin09
Department of Computer Science
Stanford University
`robin09@stanford.edu`

1 Introduction

For the CS230 project, we investigated into the methodology of Differentiable Digital Signal Processing (DDSP) [1] and proposed a use of well-known sound synthesis algorithm called Karplus-Strong (KS) model wihtint the framework of DDSP for musical instrument sound synthesis application. DDSP is a novel methodology that leverages classical digital signal processing (DSP) and deep learning in a symbiotic way. In [1], sound synthesis is used as an application to illustrate the concept of DDSP. DSP-based sound synthesis models have solid interpretable grounds but often struggle with tuning parameters for good sound synthesis results especially if the DSP model has a large number of parameters. DDSP integrates DSP models into the neural networks in a way that the neural networks play a role of estimating parameter values. Thanks to the expressivity of the neural networks, it is shown that the synthesis results from the DDSP framework are promising in terms of not only the quality of the sound but also flexibility of the model. Augmenting the outcome from the latest milestone checkpoint, we have implemented the ‘z‘ latent vector encoder in the existing Pytorch implementation [2]. and tried the KS model as the DSP module. Analysis the results of training of each variants of the baseline DDSP model is provided in this report as well.

2 Dataset

Following the original DDSP work, we chose subsets of the NSynth dataset [3] for training the DDSP framework. The NSynth dataset is a collection of 306043 musical notes from acoustic, electronic instruments and synthesis. The musical notes in the NSynth dataset are diversified not only in the type of the instruments but also in pitch and loudness (velocity in terms of MIDI). The dataset also provides useful pre-labeled features such as pitch, note strength, sonic quality, etc., for each note. We have tried two subsets from the NSynth as following:

- DS1 : 1517 sounds of acoustic guitar.
- DS2 : 11886 sounds of acoustic brass instruments.

As for the DS1, sounds of plucked strings of acoustic guitars are chosen in order to investigate how the DDSP framework handles the transient sounds. On the other hand, sounds of acoustics brass instruments were chosen for the DS2 for the case of stationary sounds. The sizes of the DS1 and DS2 were set different by a factor of 10 in size to observe how the size of the dataset affects the training results.

3 Approach

3.1 Baseline

The baseline of the DDSP model we chose is an existing pytorch implementation of DDSP framework [2], we tried training the DDSP framework. The architecture of the DDSP framework is illustrated in Fig. (1). The input signal, an audio signal, is fed to the DDSP framework in a frame-by-frame basis. In our experiment, the framesize is set 160 samples at the sampling rate of 16000Hz. At n^{th} frame, the encoder part of the architecture is to produce an estimate of the fundamental frequency (F0) $f(n)$ of the input signal and the latent representation $z(n)$ that serves as the timbre estimate and the loudness $l(n)$. $f(n)$ estimation is carried out by a pre-trained CREPE algorithm [4] placed in the encoder. Loudness $l(n)$ is computed as a mean of log-scaled A-weighted power-spectrum over frequency. In our attempts of training, a part of estimating $z(n)$ is excluded from the encoder and $z(n)$ is not utilized. Thus only pairs of $f(n), l(n)$ are fed to the decoder to estimate the parameters

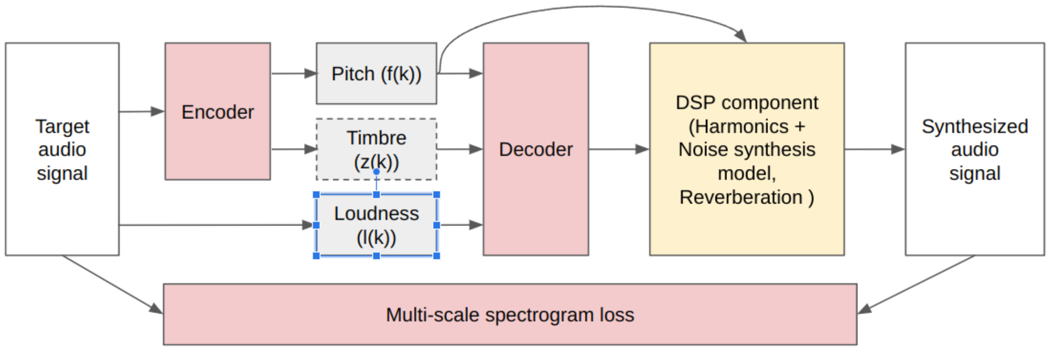


Figure 1: DDSP architecture. Components in red are for neural networks and the component in yellow represents the DSP part of the DDSP

of the harmonics + noise synthesis model, the DSP-based synthesis model, to generate sound. We used the multi-scale spectral loss proposed in [1], which is defined as

$$L_i = ||S_i - \hat{S}_i||_1 + ||logS_i - log\hat{S}_i||_1 \tag{1}$$

$$L_{total\_reconstruction} = \sum_i L_i \tag{2}$$

where S_i and $\hat{S}_i$ are magnitude spectrograms of the input signal and the synthesized signal of i^{th} FFT size, respectively. In this training FFT sizes of (2048, 1024, 512, 256, 128, 64) are used. The loss for i^{th} FFT size is a sum of the L1 distance between S_i and $\hat{S}_i$ and the L1 distance between $logS_i$ and $log\hat{S}_i$. The total reconstruction loss is given as a sum of L_i as in eq.(2). For the training, each note in a dataset is first zero-padded for its length to be an integer mulitple of the sampling rate (16000 Hz). As a result the length of each note in both DS1 and DS2 are enlongated to 96000 samples. Elongated notes are split into two 48000 sample-long notes thus the size of the datasets gets doubled. Then each 48000 sample-long note is decomposed into frames of 160 samples for which the F0s ($f(n)$) and the loudnesses ($l(n)$) are pre-estimated using the encoder. As $z(n)$, the latent timbre representation, was not considered in this initial training experiment, the training of the neural networks is essentially training the decoder component of the DDSP framework only. The details of the decoder network can be found in [1].

The minibatch size of 16 notes is used for the training on both DS1 and DS2. The iteration number is set as 500,000 thus the number of epoch for DS1 training was 500000/(1517x2/16) = 2636 and the number of epoch for DS2 training was 500000/(11886x2/16) = 337. The Adam optimizer was used with the fixed learning rate of 0.001.

3.2 Baseline + z encoder

In the baseline model, the timbre representation $z(n)$ is not utilized as the implementation of z encoder is missing in the original code base in [2]. We added the implementation of the z encoder in the code base. The architecture of the z encoder is shown in Fig.(2). The z encoder consists of 128-bin (covering the range of 20Hz to 8000Hz) Mel frequency cepstrum coefficient (MFCC) computation based on 1024-bin FFT, a normalization layer, 512-unit GRU and a linear layer with 16 units, all following the tunings originally used in [2]. With this implementation of z encoder in place

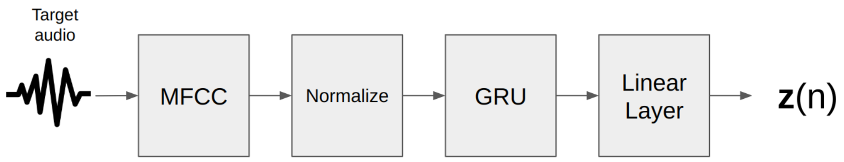


Figure 2: Architecture of $z(n)$ encoder

in the DDSP architecture, the box representing *Timbre z(k)* in Fig. (1) is enabled.

3.3 Baseline + z encoder with Karplus-Strong model

Karplus-Strong (KS) [5] is a simple physically-based sound synthesis model. As illstrated in Fig. (3), the KS model consists of a delayline defined as z^{-N} and the loop filter $R(z)$ in the z-domain. The loopfilter $R(z)$ is a filter applied to the delayline output at each loop iteration. If plucked sound of a string instrument (such as guitar) were to be modeled, rough physical interpretation of the model would be that the input $x(n)$ serves as a excitation (plucking) to a string and the delayline provides periodicity that defines the pitch of the sound. The loopfilter $R(z)$ models timbre of the sound which is freqeuency-dependent attenuation in the plucked string case. DDSP with the KS model replaces

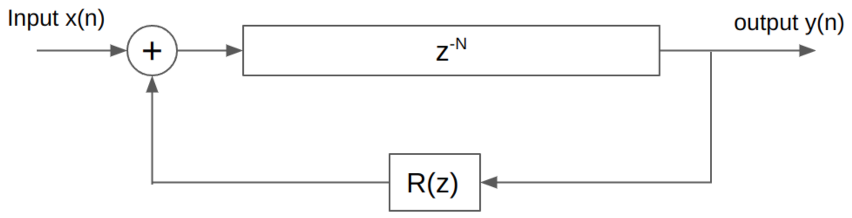


Figure 3: Karplus-Strong synthesis model

the harmonics + noise synthesis model (yellow block) shown in Fig.(1) with the KS model. In order to drive the KS model for synthesis, the delay amount N for the delayline, the input signal $x(n)$ and the loopfilter $R(z)$ need to be configured. Given the fundamental frequency $f0$ of the target sound and the sampling frequency f_s, N is defined $2f_s/f0$. The KS model is implemented to synthesize a signal at one shot, not frame-by-frame basis thus the frame-based $f(n)$ is averaged over frames to obtain $f0$. The length of the synthesized signal is set as 48000 samples so $f0$ is obtained by averaging 300 frames of $f(n)$. For the input signal $x(n)$ and the loopfilter $R(z)$ (Details of the decoder can be found in the appendix). One of the dense layer (of 512 hidden-unit) output in the decoder network (Fig.(4) is assigned as the input $x(n)$. As the decoder operates on a frame-by-frame basis, the outputs of the dense layer assigned for $x(n)$ are averaged over frames to obtain $x(n)$ of length 512 samples. The other dense layer (of 512 hidden-unit)'s output of the decoder is processed to be non-negative and turned into impulse response ([1] Appendix B.5) to obtain the FIR loopfilter $R(z)$'s coefficients. (4).

4 Training results

4.1 Baseline

In fig.(5), trajectories of training loss over iterations (500,000 times) for both DS1 and DS2 are shown. It can be seen that for both DS1 and DS2, the training loss reduces quite rapidly in the first 20,000 iterations then somehow converges (saturates) to a certain level. Since the loss used for the training is not necessarily normalized in the range of 0 to 1, it is somehow hard to conclude how well it performs in terms of training bias. Thus we checked the synthesized sounds at the beginning of the training

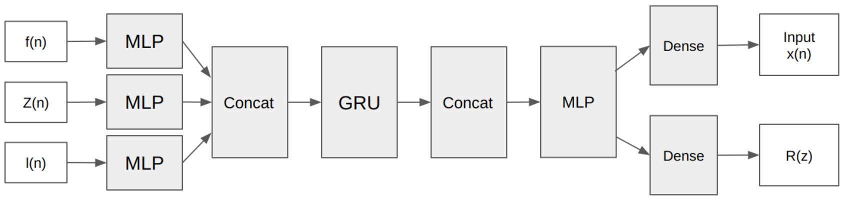
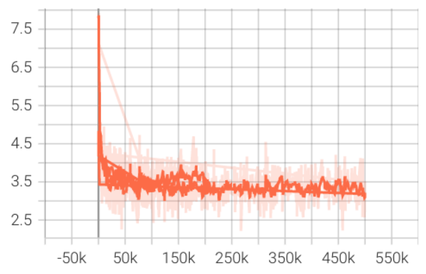
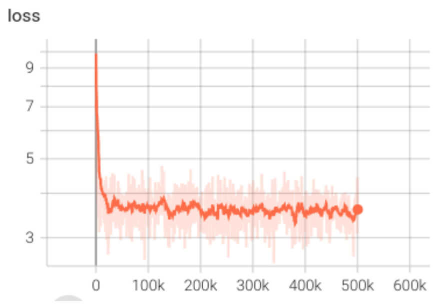


Figure 4: Decoder network for KS model

and at the of the training for both datasets. Compared to the input notes, the synthesized notes do not necessarily sound identical to the inputs but the synthesized results sound quite natural. Examples of sounds can be found in the submitted code. In the submitted code, the synthesis results as WAV files are stored in `/runs/guitar_ac` (DS1), `/runs/brass_ac` (DS2). `eval_1stRun.wav` files are from the first epoch and `eval_lasRun.wav` files are from the last epoch. In the WAV files, pairs of 3-second long input and 3-second long synthesis are concatenated for side by side comparison purpose.



(a) Training loss with DS1



(b) Training loss with DS2

Figure 5: Training losses

4.2 Baseline + z encoder

The baseline + z encoder model is trained using DS1 and the hyperparameters used for training the baseline model except the number of iterations of training was reduced from 500,000 to 400,000. As seen in Fig.(6), the training loss decreases quite rapidly in the beginning (<20,000 iterations) and reaches the convergence (saturation). When compared to the results from the baseline model, the subjective sound quality of the synthesized signals from the baseline + z encoder model is similar to that from the baseline model according to (informal) the listening evaluations. In the submitted code, the synthesis results (WAV files) from both the baseline and the baseline + z encoder models are stored in `/runs/guitar_ac` and `/runs/guitar_ac_z`. `eval_1stRun.wav` files are from the first epoch and `eval_lasRun.wav` files are from the last epoch. In each file, pairs of 3-second long intput and 3-second long synthesis are concatenated side by side.

4.3 Baseline + z encoder with Karplus-Strong model

DS1 is used to train the DDSP baseline model with the z encoder and the KS model. This choice of DS1 is based on the fact that the length of the input signal $x(n)$ is configured to be shorter than that of synthesized signal. As guitar sound is generated by a short pluck force, DS1 is considered more appropriate with the given setting. We use the same hyperparameters for training as the baseline model training case. The batch size used was 16 notes (each note is 48000-sample long) . The training was unsuccessful in that this KS model incorporated DDSP model becomes unstable quickly. After a couple of number of iterations while training, the synthesized signal becomes unstable as seen in Fig.(7). From the second batch, synthesized signals during training start diverging. As an attempt

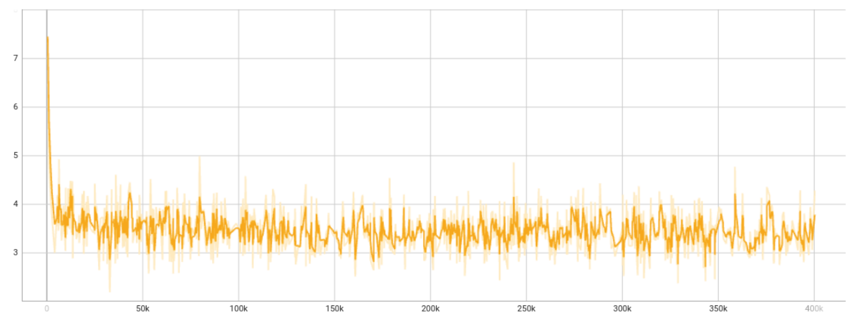
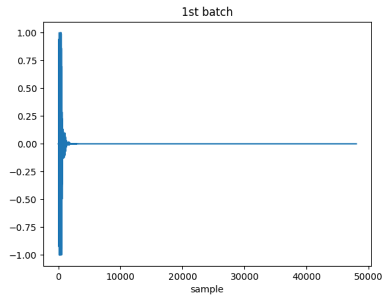
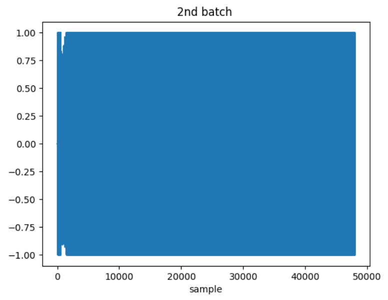


Figure 6: Training loss with DS1, Baselnie + z encoder model

to mitigate this instability, we tried normalizing both $x(n)$ and the coefficients of $R(z)$ by their norm, respectively, but no noticeable effect was observed. We speculate that the instability is due to the fact that there is no stability-related constraint applied in defining the loopfilter $R(z)$ and learning it from the network. As the KS model is an IIR filter, if the denominator of its transfer function is unstable by having poles outside the unit circle, the output of the system becomes unstable. Due to this instability, trained network only produces synthesized signals with NaN (not a number) samples.



(a) First synthesis result from the first batch



(b) First synthesis result from the second Batch

Figure 7: Unstable synthesized signals during training

5 Conclusion and future work

DDSP framework has been investigated and trained for musical instrument synthesis application. First we tried training the baseline DDSP framework using an existing pytorch implementation. In addition, we augmented the encoder part of the baseline model with the z encoder and attempted to use the KS model as an alternative DSP synthesis module. We were able to verify that the baseline and the baseline with the z encoder models are capable of synthesizing plucked string sound and brass instrument sound in terms of subjective, perceived sound quality. With the KS model as the alternative DSP module, the system was not able to produce stable synthesized signal as a result. By witnessing the synthesis results from the DDSP models (with and without z-encoder) with the original harmonic + noise model as the DSP synthesis module are in great quality, we are convinced that the DDSP is a compelling framework for sound synthesis applications. As a future work, we hope to find ways of ensuring the stability of the KS model loopfilter in the framework of DDSP model as well as alternative decoder architecutre that is better suited to estimation of the $x(n)$ and the loopfilter.

A Appendix : Decoder network of DDSP

In Fig.(4), the decoder network of the DDSP is shown. the decoder first applies MLPs to $f(n)$, $z(n)$, $l(n)$ and the outputs of the MLPs are concatenated and fed to 512-unit GRU. The output of the GRU is concatenated with the $f(n)$ and $l(n)$ MLP outputs and passed to another MLP. The output of the final MLP is fed to the dense layers that produce the input $x(n)$ and the loopfilter $R(z)$'s coefficients.

As shown in in Fig.(8), an MLP is a serial cascade of a structure that constists of a dense layer, layer normalization and the Relu activation three times. Each unit in the MLP has 512 units.

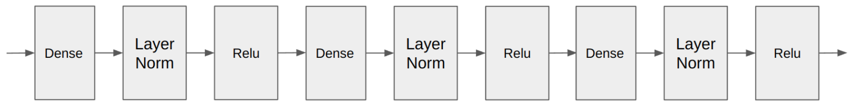


Figure 8: MLP in the decoder network

References

[1] Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. Ddsp: Differentiable digital signal processing. *arXiv preprint arXiv:2001.04643*, 2020.

[2] acids ircam\ddsp_pytorch. Differentiable digital signal processing. `https://github.com/acids-ircam/ddsp_pytorch/`, 2021.

[3] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. Neural audio synthesis of musical notes with wavenet autoencoders. In *International Conference on Machine Learning*, pages 1068–1077. PMLR, 2017.

[4] Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello. Crepe: A convolutional representation for pitch estimation. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 161–165. IEEE, 2018.

[5] Kevin Karplus and Alex Strong. Digital synthesis of plucked-string and drum timbres. *Computer Music Journal*, 7(2):43–55, 1983.