Towards Perceptual Deep Learning-based Packet Loss Concealment

Filippo Daniotti
DISI
University of Trento
Trento, Italy
filippo.daniotti@studenti.unitn.it

Luca Vignati
DISI
University of Trento
Trento, Italy
luca.vignati@unitn.it

Luca Turchet
DISI
University of Trento
Trento, Italy
luca.turchet@unitn.it

***Abstract*—Recent advancements in deep learning paved the way to novel approaches to the problem of Packet Loss Concealment. However, deep neural networks may have large inference times and therefore violate the strict temporal requirements of PLC. A promising avenue lies in the exploration of the loss function used to train the network, as loss functions have direct impact of the latent representation learned by the model during the training process without any additional cost at inference time.**

In this work, an attempt was made to define a perceptual loss function, i.e. a loss function that allows the model trained with it to have downstream performances that correlate with the human evaluation. The proposed Loss Function reweights the frequency axis of the difference between the magnitude spectrogram of the target and the prediction, allowing the model to learn a better representation of the frequency content and yield higher quality concealments.

***Index Terms*—Internet of Musical Things, Networked Music Performance, Packet Loss Concealment, Deep Learning**

I. Model

A. Architecture

In this submission we are using PARCNet, the model proposed in Mezza et al. [4], as a backbone architecture. We are changing the size of the input tensor in order to match the different packet length, which is 512 samples for this challenge.

B. Loss function

In this submission we developed a novel perceptually-motivated loss function, the *Tilt Loss*, named after the Tilt filter equaliser.

1) Intuition: The underlying motivations for the design of this loss are rooted in empirical observations on the prediction data from the PARCNet model. The analysis of such observations highlighted the fact that the concealed signal was unbalanced in the reconstruction of the frequency content, as frequencies in the low and the mid-frequency range were more accurately reconstructed, while frequencies in the high range were noticeably more inaccurate. This behaviour can be clearly seen from the spectrograms of the concealments and the waveforms. In other words, the model trained with the PARCNet loss has less sensitivity on the mid and low-end than the high-end. At the same time, the perceptual quality of the model is far superior in the higher-sensitivity low-mid frequency range than in the less-sensitive high range. Hence, it arose the idea of rebalancing the frequency spectrum in favour of the high-frequency range in order to push the model to concentrate thoroughly on learning a better representation of the spectrum high-end. This rebalancing is defined as a reweight mask applied to the frequency bins of the MAE of the predicted and the ground truth, hence acting as an upward tilt filter.

2) Definition: The loss is defined as follows:

$$\mathcal{L}_{\text{TILT}}^{(i)} = \frac{1}{T \cdot F} \left\| \vartheta \left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)} \right) \odot \left(|\hat{S}|_{t,f}^{(mel)} - |S|_{t,f}^{(mel)} \right) \right\|_1 \tag{1}$$

The spectral adaptive reweighting mask is defined through the Tilt frequency reweighting function ϑ, which is defined at a high-level as follows.

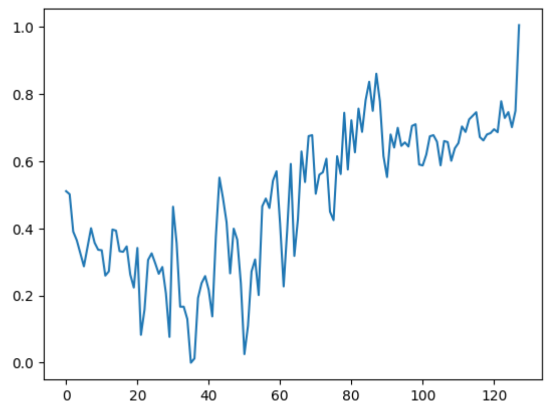
$$\vartheta \left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)} \right) := 1 - \text{butter}\Big[\text{DFT}(\mathbf{x}^{(i)} \oplus \mathbf{y}^{(i)}) \Big] \tag{2}$$

where $\oplus$ is the concatenation operator and *butter* is a low-pass filter designed as a butterworth filter.

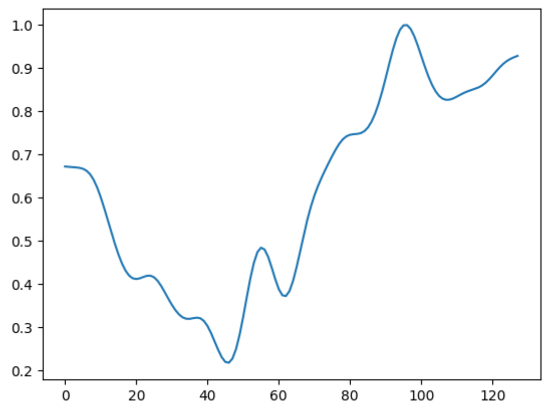
3) Design discussion: When designing the loss, a number of different reweight strategies were considered. In the end, we chose a so-called *spectral adaptive reweighting strategy* in order to make the reweighting curve dynamic, meaning that the reweight mask is adaptive with respect to the frequency content of the context. This is crucial for musical audio, as the frequency content of the signal varies a lot depending on the instrument, style or even individual passage.

a) Tilt spectral adaptive reweighting mask computation: The procedure is summarized in Algorithm 1.

The DFT is computed with a FFT computed with a window size of the context length plus the packet length and a FFT size of the next power of 2 of the window size, that is $2^{\lceil \log_2(\text{win\_length}) \rceil}$. The Mel scale transformation is computed with a 128 Mel filterbank. The transformation to DB scale is computed as $20 * \log_{10}(\text{mel\_spectrum} + \epsilon) + c$, where $\epsilon = 10^{-7}$ is a constant to prevent numerical errors on the logarithm computation and $c = 80$ is a constant to translate the values of the curve and simply the following normalisation. As a low-pass filter, a Butterworth filter of order 5 is used and the cut-off frequency 50. In order to prevent a downward ramp at the beginning of the mask, the mask is applied a left padding



(a) Normalized Mel spectrum before the filtering.



(b) Normalized Mel spectrum after the filtering.

Fig. 1: The effect of the low-pass filtering on the Tilt spectral reweighting mask.

Algorithm 1: Tilt spectral adaptive reweighting mask ϑ computation.

```
Data: Context $\mathbf{x}^{(i)}$, Ground truth $\mathbf{y}^{(i)}$, shape of the
      spectrogram $shape$
Result: Spectral reweight mask $\vartheta$
spectrum $\leftarrow$ DFT($\mathbf{x}^{(i)} \oplus \mathbf{y}^{(i)}$) ;
mel_spectrum $\leftarrow$ MelFilterbak(spectrum) ;
db_spectrum $\leftarrow$ AmplitudeToDB(mel_spectrum) ;
mask $\leftarrow$ LowPassFilter(db_spectrum) ;
mask $\leftarrow 1 - mask$ ;
mask $\leftarrow$ Normalize(mask, $[0, 1]$) ;
$\vartheta \leftarrow mask$.Expand($shape$) ;
return $\vartheta$
```

repeating the first value. The size of the left padding is equal to 50. The Butterworth designed was performed with SciPy [6] with the `scipy.signal.butter` module and applied with the `scipy.signal.lfilter`.

The impact of the low-pass filtering on the resulting curve can be seen from Figure 1.

II. Data

The model was trained on the *Bach Cello Suite* dataset [1], originally introduced by Chris Chafe. It consists of several hours of recordings of excerpts taken from the Back Cello Suites corpus. The instrument feature in this dataset is the violoncello, which produces mostly monophonic sound with low attack and high decay and release. The recordings were captured across several days and they are unedited, featuring pauses, artefacts and environmental noises, hence representing a typical rehearsal scenario. The dataset is available online[1].

In order to prepare the audio tracks to be fed to the model, they should be packetised, i.e. split into fixed-length packets. The packet length, in compliance with the challenge requirements, is set to 512 samples, which corresponds to around 11 ms of audio at the target sampling rate of 44.1kHz. Audio recordings usually feature millions of samples, which would result in prohibitive disk and memory requirements. Hence, in order to reduce the overall dimension of the problem, a *packet skip factor* is defined. The packet skip factor acts as a lag variable that indicates the distance in samples between the index of the first sample of a packet and the index of the first sample of the following. We used a packet skip factor of 2, meaning that we are using $\propto 50\%$ of the total available packets.

[1]https://ccrma.stanford.edu/~cc/som/bachCello/

III. Training experiments setup

All the training experiments were run on a 64 bit Linux machine running Ubuntu 22.04.3 LTS. The machine was equipped with an Intel(R) Core(TM) i9-10940X 3.30GHz CPU with 2 threads per core and 14 cores, 200GB of RAM and two GPUs, namely an NVIDIA GeForce RTX 4090 and an NVIDIA GeForce RTX 3090, both featuring 24GB of dedicated VRAM. All the experiments were performed with Python 3.11.5 and CUDA 12.2, leveraging the deep learning framework PyTorch [5] and its convenient wrapper Pytorch Lightning [2]. All the random generators were eeded with the same seed (42).

The RAdam optimisation algorithm [3] with $\beta_1 = 0.5$ and $\beta_2 = 0.9$ is employed as in the papepr. Each model was trained for 100 epochs with a batch size of 128 datapoints per batch. The learning rate η was set to 10^{-3} and was decayed by a factor of 0.1 when the validation loss did not improve for a number of epochs; this patience value was set to 10.

References

[1] Chris Chafe. *Bach Cello Suites Data Repository*. Accessed: July 1, 2020. 2020. URL: https://ccrma.stanford.edu/~cc/som/bachCello.

[2] William Falcon and The PyTorch Lightning team. *PyTorch Lightning*. Version 1.4. Mar. 2019. DOI: 10.5281/zenodo.3828935. URL: https://github.com/Lightning-AI/lightning.

[3] Liyuan Liu et al. “On the Variance of the Adaptive Learning Rate and Beyond”. In: *CoRR* abs/1908.03265 (2019). arXiv: 1908.03265. URL: http://arxiv.org/abs/1908.03265.

[4] Alessandro Ilic Mezza et al. “Hybrid Packet Loss Concealment for Real-Time Networked Music Applications”. In: *IEEE Open Journal of Signal Processing* 5 (2024), pp. 266–273. DOI: 10.1109/OJSP.2023.3343318.

[5] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *CoRR* abs/1912.01703 (2019). arXiv: 1912.01703. URL: http://arxiv.org/abs/1912.01703.

[6] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.