Time Domain Source Separation with Spectral Penalties

Andrey Akhmetov
The Cooper Union
New York, NY, USA

Ostap Voynarovskiy
The Cooper Union
New York, NY, USA

Abstract— **Models that perform musical source separation typically work purely in the time domain (e.g. Wave-U-Net) or only in the magnitude spectrum, thus omitting phase information. We extend the work of Wave-U-Net which achieved end-to-end source separation in the time domain using one-dimensional convolutions by investigating alternative loss functions operating on a spectral representation of audio. Our experiments show that a loss function in the spectral domain is appropriate for penalizing errors in a time-domain source separation network, with rapid convergence and comparable subjective audio quality.**

I. INTRODUCTION

Source separation is a signal processing problem with the goal of extracting one or more individual sources from a mixed audio stream. Technologies like hearing aids, home assistants like Google Home or Amazon's Alexa, and music post-production software all stand to benefit from the ability to separate audio streams from a single audio source. Nearly all of the recent success in the field has performed sound separation on the spectrogram of the audio instead of the raw wave forms [1]. However, performing source separation on the spectrograms has the major drawback of completely omitting the phase information. Before neural networks, matrix decomposition methods such as independent component analysis (ICA) [2] and non-negative matrix factorization (NMF) [3] had been used in the field. Performing NMF required constraining the information to non-negative values, which worked well for a magnitude spectrogram but could not be used for the -1 to 1 values of a raw waveform. ICA was thus the method of choice for processing wave forms. ICA struggled however as the unpredictable phase component meant that a single basis could not represent a source so either shift invariant bases or many bases were required. Though some matrix decomposition methods were designed to operate on the direct wave forms, they did not perform as well as their spectrogram based counterparts. The phase information presented difficulties which researchers could not overcome with the tools they had at the time, so they often chose to drop the phase information altogether and operate primarily on spectrograms instead.

Since neural net based source separation was introduced, the models predominantly focused on spectrogram representations of the audio [10]. The model took sequential spectrogram frames as input to the model and output a prediction in the form of a spectrogram. To recreate the audio, the output spectrogram was combined with the mixture phase and fed through an inverse STFT. Instead of using the mixture phase, the source phase could be estimated by using the Griffin-Lim algorithm but this is quite slow [4]. The spectrogram output also depends on the type of window used, the size of the window, and the overlap of frames. Ideally these parameters should be tuned to provide better results. However, doing so is computationally expensive as one would need to train multiple models to completion, therefore one set of parameters is chosen and no tuning is performed.

Wave-U-Net was introduced in order to investigate the feasibility of performing end to end sound source separation directly in the time domain [5]. The implementation proved that performing source separation directly in the time domain outperformed the state-of-the-art spectrogram based u-net architecture when trained under comparable settings.

The authors of Wave-U-Net highlight a lack of temporal input context in recent models. Audio recordings are sampled at 44.1 kHz. Thus to be able to analyze and effectively extract a source from a mixed audio track, a model requires quite a large input size. The Wave-U-Net authors also realized that when a model's input and output sizes are identical, concatenating the sequential outputs results in artifacting at the boundary between the two outputs. To tackle this they proposed using an input size greater than their output size to allow for the output to gain a contextual understanding and prevent these boundary artifacts.

Inspired by the success, we decided to explore the authors' suggested future work and analyze a variety of loss functions to see if performance improvements can be made. We propose a scheme for source separation that retains the use of the Wave-U-Net topology, which operates directly on time-domain samples, while using a penalty function that operates on the STFT of predicted and ground-truth outputs. This allows us to retain the ability to reconstruct phase (a task that models operating on the magnitude spectrum cannot do without additional post-processing), while exploring the qualitative and perceptual effect of penalizing magnitude and phase errors directly.

II. RELATED WORK

Other papers that tackle audio source separation in the time domain include TasNet [6] and MRCAE [7]. Time domain Audio Separation Network or TasNet, is a convolutional auto-encoder designed for the problem of multi-talker acoustic environments. Dissatisfied with the poor spectrogram reconstruction for speech separation, and the latency introduced by needing to calculate the spectrogram, Luo et al. decided to use an auto-encoder architecture. Their model decomposes the signal into a set of basis signals then uses masks to extract the source signal [6]. While the model is effective, the authors of Wave-U-Net highlight the fact that certain conceptual tradeoffs are made in order to allow for the model's ability to be used for low latency applications.

The Multi-Resolution Convolutional Auto-Encoder or MRCAE is a fully convolutional denoising auto encoder with 5 convolutional filters of different lengths in parallel for each of 2 layers of the encoder and decoder. The encoder seeks to extract multi-resolution features from the input mixtures and the decoder uses these features to estimate the sources [7]. The model's performance is likely limited by a few factors. First off, the encoder and decoder are only 2 convolutional layers deep, which is likely not deep enough to extract meaningful high dimensional representation of the sound source. Additionally, it only operates on 1025 samples (about 23 ms) which likely limits the models ability to exploit the context of the input signal to a much of a degree.

The Wave-U-Net[5] has shown the most success for end to end sound source separation in the time domain. Utilizing a 1-dimensional adaptation of the U-net [8], the model alternates stacking convolutional layers on top of downsampling layers. It increases the number of filters with every convolution in order to extract more complex features. Once reaching the last downsampling block and convolving one more time, it begins a series of upsampling, concatenation, and convolutional layers. On this side it decreases the number of filters with every convolution until it reaches the top layer and the concatenation is performed with the respective layer on the downsampling side along the filter dimention. Finally it uses a set of convolutions of size one to collapse the filter dimension into a stereo audio track.

III. MODEL

A. Topology

Our topology differs slightly from the one described in [5], but it follows the same basic structure. In order to save on computational time we made the decision to downsample from 44.1 kHz to 22.05 kHz. Although some may argue retaining CD quality is of utmost importance, we could not notice a significant decrease in quality and it allowed us to save time on the limited compute resources we had. We however did decide to use stereo audio channels.

The model consists of 11 downsampling layers, which detect increasingly higher level features on coarser and coarser timescales, and 11 upsampling layers which allow the information extracted by the high dimensional features to be used to reconstruct a source-separated audio stream.

The downsampling path is constructed from alternating 1D convolutions and 2x downsampling operations. Each convolution uses a kernel with length 15 samples and adds no padding; the downsampling is performed by discarding every other sample. Each downsampling layer is followed by a leaky-relu activation.

The upsampling path consists of 2x upsampling operations (performed using linear interpolation), 1D convolutions (kernel length 5, no padding), a leaky-relu activation function, and concatenation operations. The last 1D convolution uses a kernel length of 1 and is followed by hyperbolic tangent activation to provide the -1 to 1 output required for audio. Each concatenation operation takes samples from a corresponding layer of the downsampling path, crops it in time to have the correct length, and concatenates it along the channel dimension. This provides robust skip connections from input to output, allowing audio samples and higher-level features to be reconstructed at every timescale as necessary.

Transposed strided convolutions are often used as a method of upsampling. However, they are known to introduce high frequency noise into the final output. Instead we upsample with linear interpolation. To avoid extrapolating the last point, the output of the upsampling is kept uneven, and only interpolation is performed. This furthers our attempts to prevent artifacting.

In addition padding was intentionally avoided. Padding enables the output of a convolution to be equal to the input size, however the zeros padded to either side of the data introduce artifacting at the edges of the final audio stream. We also wanted to ensure that the model obtained temporal context. We accomplished this by setting the output of our model to be smaller than the input. This means that the model only has to extract the source audio for the middle 0.65 seconds in an approximately 3 second audio clip. By starting with an uneven number of samples at the input, avoiding padded convolutions and not extrapolating when upsampling, we ensured that our model would have contextual information about the source it was trying to extract.

Table 1 and 2 depict the specific size of the model's output samples and channels after each layer of the network. A complete table was included to help readers better understand the output sizes of the model at each step and offer a sanity check to anyone attempting to re-implement our model.

B. Loss functions

The previous section described the network topology and structure that was utilized. Here, we discuss a number of loss functions that we tested on this topology.

1) Time-domain losses: The prior work performed by Wave-U-Net uses mean squared error over all samples in a batch [5]. Since we predict only one stereo track at a time (for example, drums) we simply take the average mean squared error over all samples in both left and right channels, with respect to the corresponding samples of the ground-truth audio. This result is used as a baseline for our experiments.

We also briefly experiment with mean absolute error, applied in the same way. Because mean absolute error penalizes severe outliers more weakly than mean squared error, we expect that it will yield a noticeable difference that we can investigate, as compared to MSE.

2) Frequency-domain losses: We also investigate a few loss functions that operate in the frequency domain. Although prior work preceding Wave-U-Net operated purely in the frequency domain, we retain the time-domain convolutions and resampling operations of Wave-U-Net, moving only penalization to the frequency domain.

Computation of each of these losses begins by performing an STFT over the prediction and ground truth samples. The frame length is 4096 and the stride is 2048; the last frame is padded to length with zeros. For each training example, we obtain ground truth complex-valued STFT $Y(m,n)$ and prediction STFT $\hat{Y}(m,n)$ with m ranging over frame index and n ranging over DFT frequency bins. We consider the following error functions $E(m,n)$, which are averaged over all possible (m,n) pairs to obtain the final loss:

1) Squared error in phase and magnitude considered separately:

$$E_m(m,n) = \Big(|Y(m,n)| - |\hat{Y}(m,n)|\Big)^2$$

$$E_p(m,n) = \Big(\operatorname{Arg}(Y(m,n)) - \operatorname{Arg}(\hat{Y}(m,n))\Big)^2$$

$$E(m,n) = aE_m(m,n) + bE_p(m,n)$$

with hyperparameters a and b both positive.

2) Euclidean distance between phasor representations:

$$E(m,n) = |Y(m,n) - \hat{Y}(m,n)|^2$$

3) Euclidean distance between stretched phasors:

$$E(m,n) = \left| \frac{Y(m,n)}{(|Y(m,n)| + \epsilon)^\alpha} - \frac{\hat{Y}(m,n)}{(|\hat{Y}(m,n)| + \epsilon)^\alpha} \right|^2$$

Includes hyperparameter α within $[0,1]$. If $\alpha = 0$ this reduces to case (2).

4) Magnitude-only squared error:

$$\Big(|Y(m,n)| - |\hat{Y}(m,n)|\Big)^2$$

IV. EXPERIMENTS

We evaluate the model's performance on the task of extracting the drum component from a mastered song.

A. Dataset

We make use of the MUSDB18 [9] database, which consists of mastered audio tracks along with separated components consisting of drums, bass, accompaniment, and vocals. We retain both left and right channels and downsample all audio to 22050 Hz.

B. Training Procedure

The model was built using Tensorflow. Training was performed on a machine with an i7-4790k and 20 GB of RAM running Ubuntu 16.04. Since the machine had an Nvidia gtx980ti, gtx980 and a Tesla K40c GPU, different models were trained on different GPU's simultaneously instead of trying to train a single model on mixed hardware. The net was optimized with the ADAM optimizer (rate $= 0.0001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. The training dataset was first resampled to 22050 Hz and partitioned into overlapping blocks (length 65523 samples, stride 14341 samples). The center 14341 samples of the desired source audio were used as the ground-truth output for each training example, while the entire block of mixed audio was used as the input. Each batch contained 10 examples, and training proceeded sequentially through the dataset. The gtx980 could not fit a batch size of 10 so a batch size of 7 was used for some models.

C. Evaluation Procedure

In order to construct a long audio sample for evaluation, we first take overlapping blocks of the mixed audio using the same procedure as during training, and infer the separated audio for each of these blocks independently. The stride is selected so that the output samples can be directly concatenated; we do not perform any filtering or interpolation between these blocks.

V. RESULTS

We compare the qualitative results achieved when we attempt to extract the drum beat from samples in our test dataset. The baseline result, using MSE as a loss, produces acceptable results after 300 epochs. Drums are heard distinctly, although they sound muffled and low-pass filtered; incomplete separation causes some accompaniment and vocals to be heard faintly, especially at higher frequencies.

Mean absolute error appears to penalize small errors more strongly than MSE, and as a result, the MAE separated audio becomes distinctly silent, rather than quiet, between drum beats. Leakage of other sources remains. However, the extracted audio sounds a bit more noisy, perhaps for the reason that outliers are not as severely punished with this error function.

Training using STFT loss function (1), i.e. squared errors in magnitude and phase considered separately, leads to numerical instabilities within the first epoch. We believe that this arises from the numerically poor behavior of the phase of the STFT for near-zero STFT coefficients. Adding low-magnitude gaussian noise to the signal delays but does not resolve this problem.

Phasor distance loss (STFT loss function (2)) directly penalizes the norm of complex-valued discrepancies in the output signal's STFT, meaning that it takes both phase and magnitude into account, and penalizes phase errors less when the signal magnitude is small, while avoiding numerical instabilities for near-silent signals. It shows remarkable performance with only 26 epochs needed for qualitatively good sound (at this point, the MSE loss had not yet achieved subjectively acceptable separation). Although samples taken at both 26 and 100 epochs show similar leakage of other channels, the sound does not appear to have severe artifacts. Training to 100 epochs only shows modest improvement, with a slightly better silence between beats. Moreover, the leakage of other channels (especially voice) and short artifacts move from being consistent over all frequencies to being localized to higher frequencies as the training continues.

STFT loss function (3) is a generalization of phasor distance loss, where errors in magnitude are considered less significantly than errors in phase. It did not show any subjectively detectable differences with $\alpha = 0.75$ and did not appear to converge with $\alpha = 0.25$. Further work is needed to ascertain the effect of the α as a hyperparameter.

Magnitude-only loss, i.e. STFT loss function (4), compares the magnitude spectra of the STFT, while ignoring all phase information. It shows significantly worse results than baseline and phasor distance loss. Although the drum beat is somewhat enhanced, other channels leak audibly, and crackling is heard in certain sections. The sound appears to have distortion similar to an overdriven amplifier, with significant harmonics and a similar timbre.

None of the resulting outputs appear to have artifacts at the boundaries where each inference outputs were stitched (section IV.B) to construct continuous audio, suggesting the model topology was effective at utilizing contextual information in the larger input audio fragment to cleanly predict the entirety of the output audio fragment.

A. *Further work*

All of the STFT-based losses use one fixed set of STFT hyperparameters: window length 4096 with stride 2048. It is very likely that these parameters strongly affect the quality of results as well as the necessary time for training. Thus, the effect of these parameters on the results needs to be explored.

Additionally, only the drum track was extracted from source audio, due to time and computing power limitations. For further work, we could investigate whether similar results arise for attempts to extract other source tracks from the mixed audio.

VI. CONCLUSIONS

Wave-U-Net has shown to be an effective strategy for end to end sound source separation in the time domain. Our experiments show that mean squared error is slightly more effective than mean absolute error for extracting a signal. Additionally, we introduce a loss metric that penalizes discrepancies in signal STFTs in the frequency domain, which shows promise as an effective method for speeding up training and providing cleaner sound source separation.

ACKNOWLEDGMENT

We would like to acknowledge the invaluable instruction, encouragement, and feedback of our instructor, Chris Curro. We also acknowledge the invaluable moral support of our peers and classmates in the Cooper EE classes of 2019 and 2020.

REFERENCES

[1] Andreas Jansson, Eric Humphrey, Nicola Montecchio, Rachel Bittner, Aparna Kumar, Tillman Weyde. Singing Voice Separation with Deep U-Net Convolutional Networks, *18th International Society for Music Information Retrieval Conference*, Suzhou, China, 2017.

[2] Aapo Hyvrinen and Erkki Oja. Independent Component Analysis: Algorithms and Applications. *Neural Networks, 13(4-5):411-430*, 2000.

[3] Daniel D. Lee and H. Sebastian Seung. Algorithms for Non-negative Matrix Factorization. *NIPS*, 2001.

[4] D. Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236243, 1984.

[5] Daniel Stoller, Sebastian Ewert, and Simon Dixon. Wave-U-Net: A multi-scale neural network for end-to-end audio source separation. *arXiv preprint arXiv:1806.03185*, 2018.

[6] Yi Luo, Nima Mesgarani. TasNet: Surpassing Ideal Time-Frequency Masking for Speech Separation. *arXiv preprint arXiv:1809.07454*, 2018.

[7] Emad M. Grais, Dominic Ward, and Mark D. Plumbley. Raw Multi-Channel Audio Source Separation using Multi-Resolution Convolutional Auto-Encoders. *arXiv preprint arXiv:1803.00702*, 2018.

[8] Olaf Ronneberger, Philipp Fischer, Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation *arXiv preprint arXiv:1505.04597*, 2015.

[9] Zafar Rafii, Antoine Liutkus, Fabian-Robert Stter, Stylianos Ioannis Mimilakis, and Rachel Bittner. The MUSDB18 corpus for music separation, 2017.

[10] Andreas Jansson, Eric J. Humphrey, Nicola Montecchio, Rachel Bittner, Aparna Kumar, and Tillman Weyde. Singing voice separation with deep U-Net convolutional networks. *In Proceedings of the International Society for Music Information Retrieval Conference (ISMIR), pages 323332.*, 2017.

APPENDIX

TABLE I

LISTING OF EACH DOWNSAMPLING LAYER IN OUR 22050 HZ MODEL.

Layer Num	**Operation**	**Output samples**	**Output channels**
1	Input	65523	2
2	Conv1D	65509	24
3	downsample	32755	24
4	Conv1D	32741	48
5	downsample	16371	48
6	Conv1D	16357	72
7	downsample	8179	72
8	Conv1D	8165	96
9	downsample	4083	96
10	Conv1D	4069	120
11	downsample	2035	120
12	Conv1D	2021	144
13	downsample	1011	144
14	Conv1D	997	168
15	downsample	499	168
16	Conv1D	485	192
17	downsample	243	192
18	Conv1D	229	216
19	downsample	115	216
20	Conv1D	101	240
21	downsample	51	240
22	Conv1D	37	264
23	downsample	19	264

See section III.A for a description of each operation.

TABLE II

LISTING OF EACH UPSAMPLING LAYER IN OUR 22050 HZ MODEL.

Operation	**Output samples**	**Output channels**
Upsample	38	264
Conv1D	33	240
Concatenate (20)	33	480
Upsample	66	480
Conv1D	61	216
Concatenate (18)	61	432
Upsample	122	432
Conv1D	117	192
Concatenate (16)	117	384
Upsample	234	384
Conv1D	229	168
Concatenate (14)	229	336
Upsample	458	336
Conv1D	453	144
Concatenate (12)	453	288
Upsample	906	288
Conv1D	901	120
Concatenate (10)	901	240
Upsample	1802	240
Conv1D	1797	96
Concatenate (8)	1797	192
Upsample	3594	192
Conv1D	3589	72
Concatenate (6)	3589	144
Upsample	7178	144
Conv1D	7173	48
Concatenate (4)	7173	96
Upsample	14346	96
Conv1D	14341	24
Concatenate (2)	14341	48
Conv1D	14341	2

Concatenation operators concatenate over the channels dimension. Each concatenation creates a skip connection from one specific layer in the downsampling portion of the model; this layer is identified in parentheses (using its number from Table 1). Further details are given in section III.A.