Differentiable All-pole Filters for Time-varying Audio Systems

$$\nabla_z \frac{1}{A(z)}$$

Chin-Yun Yu[1], Christopher Mitcheltree[1], Alistair Carson[2], Stefan Bilbao[2], Joshua D. Reiss[1], György Fazekas[1]

[1]C4DM, Queen Mary University of London

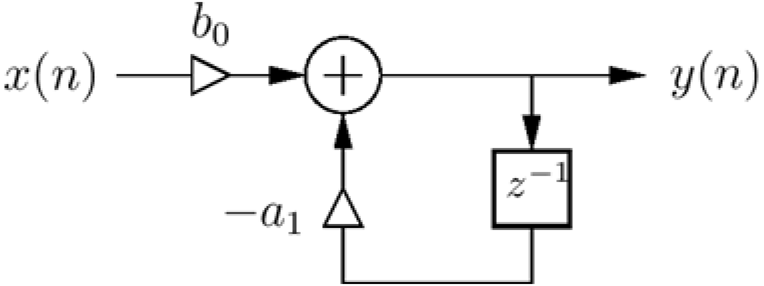
[2]AAG, University of Edinburgh

UK Research and Innovation · Queen Mary University of London · centre for digital music · The University of Edinburgh · Acoustics & Audio Group

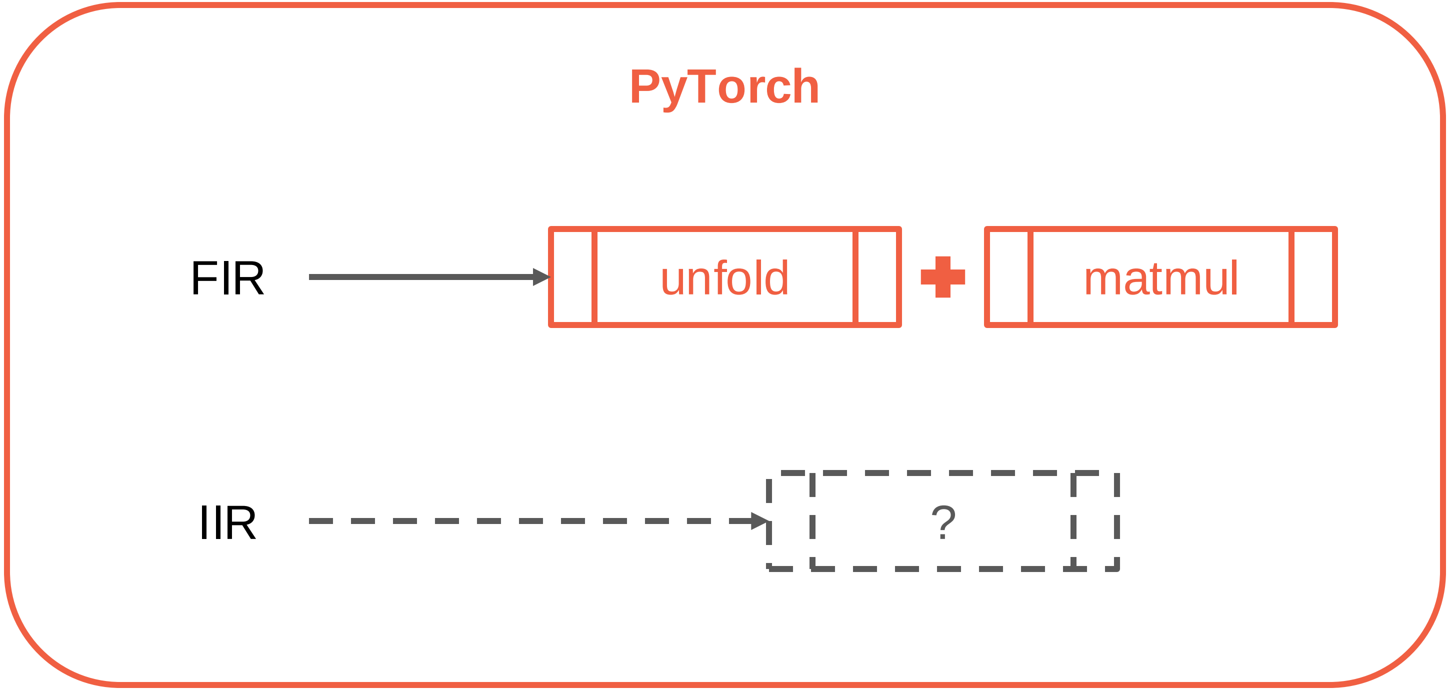
DAFx24

Introduction

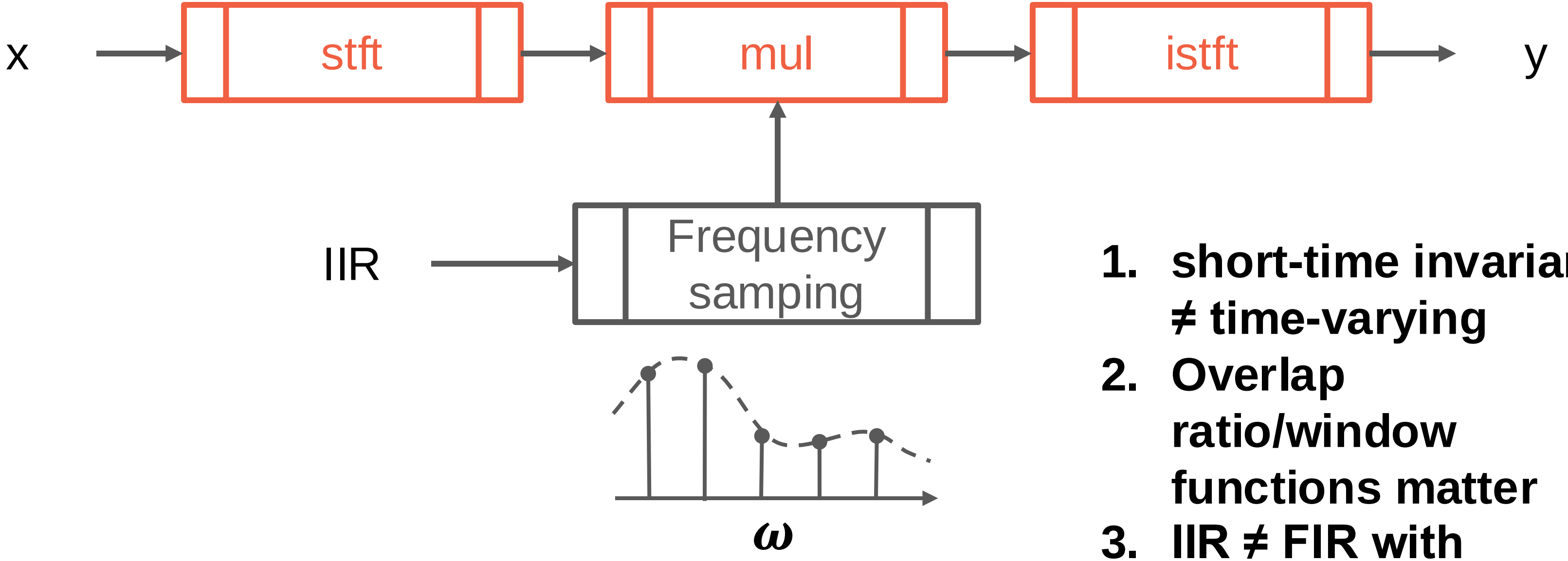
Many time-varying systems embed recursive filters (IIR)



Lack of low-level differentiable operators



Frame-wise frequency-sampling introduces mismatch



1. **short-time invariant ≠ time-varying**
2. **Overlap ratio/window functions matter**
3. **IIR ≠ FIR with circular convolution**

Contributions

1. Efficient time-domain training of time-varying recursive filters without approximation
2. No mismatch between training and real-time inference conditions
3. More accurate modelling of time-varying analog audio circuits

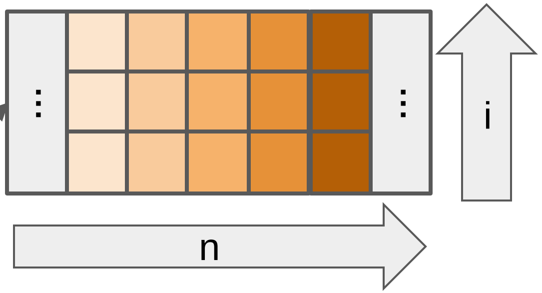
torchlpc
An all-pole filter and low-level operator

```
pip install torchlpc
```

The all-pole filter

The minimal form of a recursive filter.

[**a**(0), **a**(1),...,**a**(N)]



$$y(n) = f(\mathbf{a}(n), x(n))$$

$$= x(n) - \sum_{i=1}^{M} a_i(n) y(n-i)$$

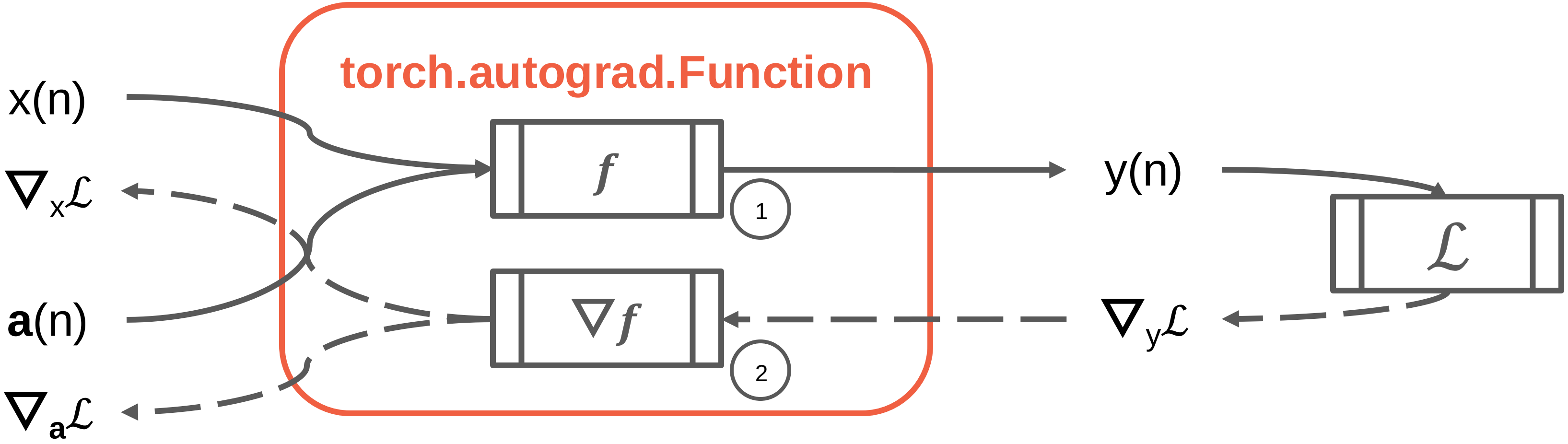
Current outputs

Time-varying filter coefficients

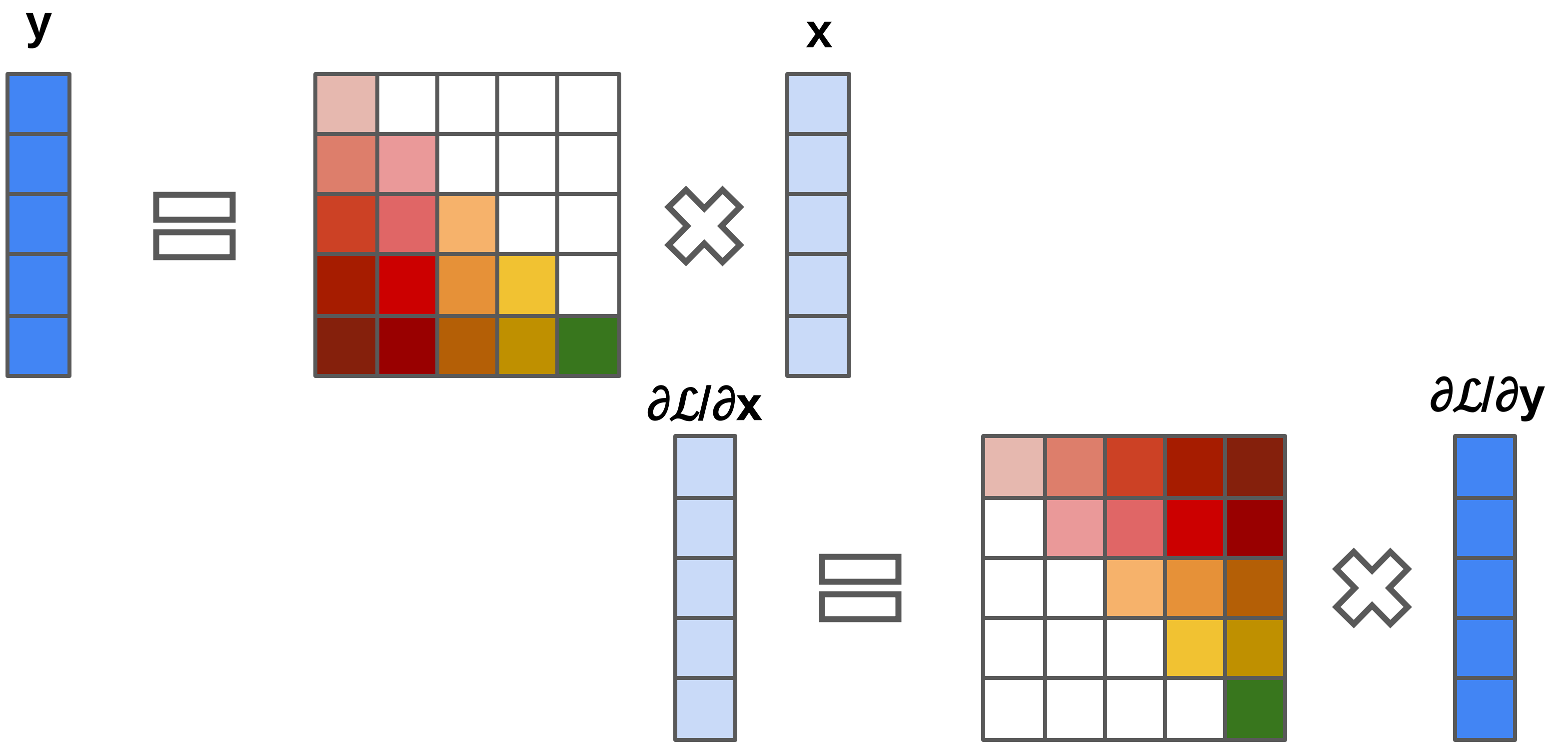
Previous outputs

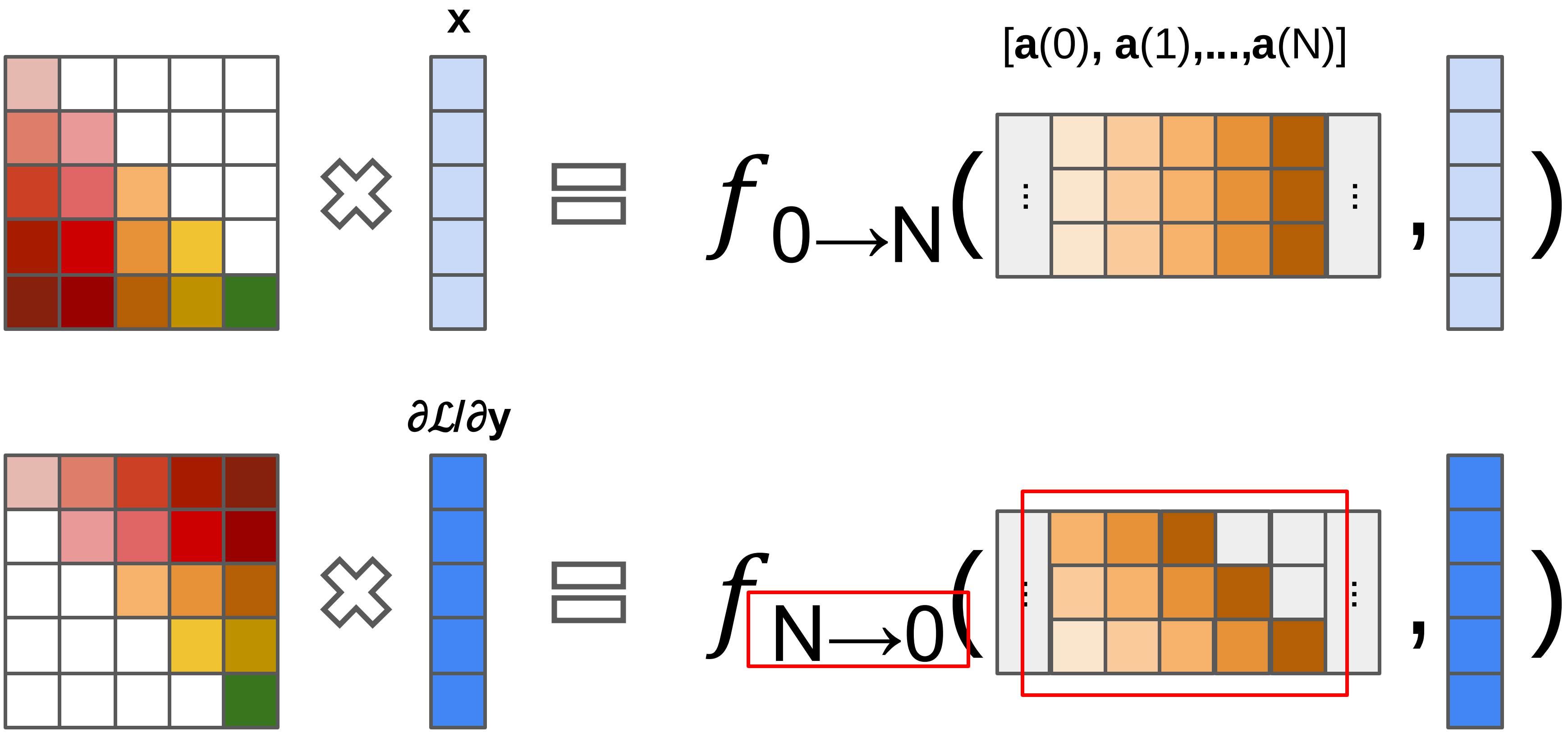
Efficient solution

1. Implement all-pole filter in compiled language using Numba
2. Derive equations for gradient backpropagation



Linear filters as matrix multiplications




x
[a(0), a(1),...,a(N)]
f0→N(,)
∂ℒ/∂y
fN→0(,)

Gradients of **a**(n)

$$\frac{\partial \mathcal{L}}{\partial a_i(n)} = -\frac{\partial \mathcal{L}}{\partial x(n)} y(n-i)$$

Bonus: Check out the github repository for gradients of the initial conditions $y(n)|_{n<0}$!

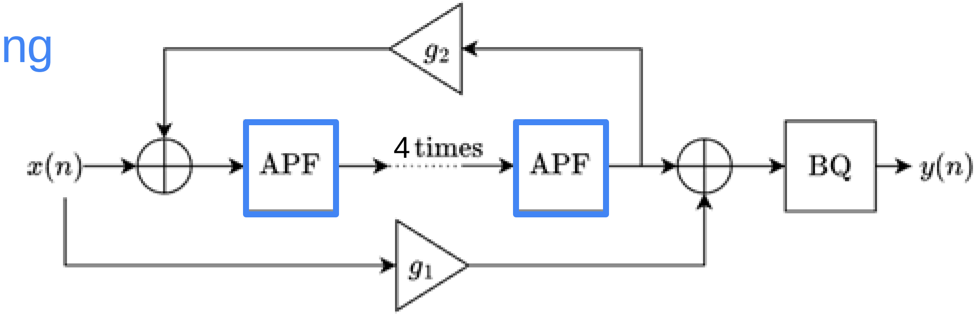
Virtual Analog modelling

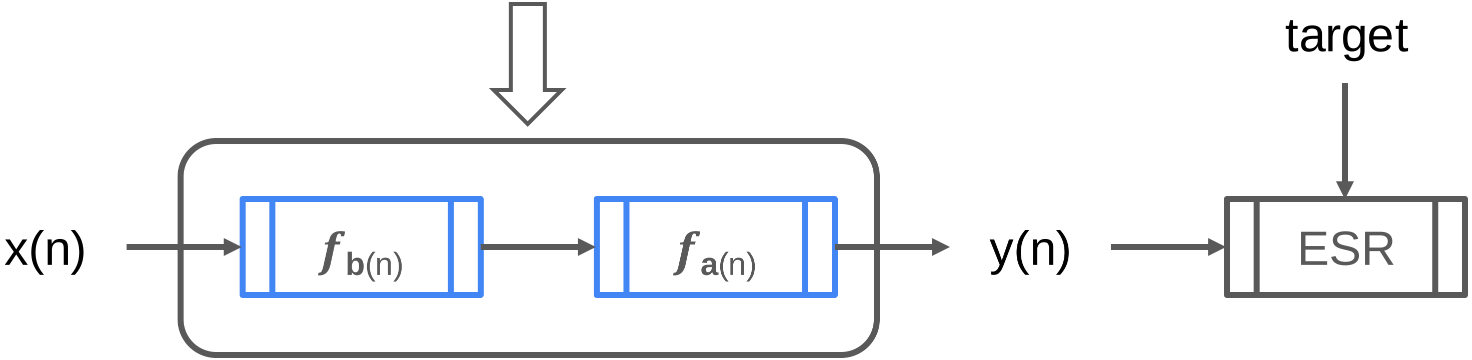
White-box DDSP approach

1. Phaser (Small Stone)
2. Synthesiser (TB-303)
3. Compressor (LA-2A)

Phaser (Electro-Harmonix Small Stone)

Time-varying





Phaser evaluation

- FS = frequency sampling
- TD = time-domain using torchlpc
- Rows = training
- Columns = inference

For most settings, TD outperforms FS no matter the inference condition.

Target

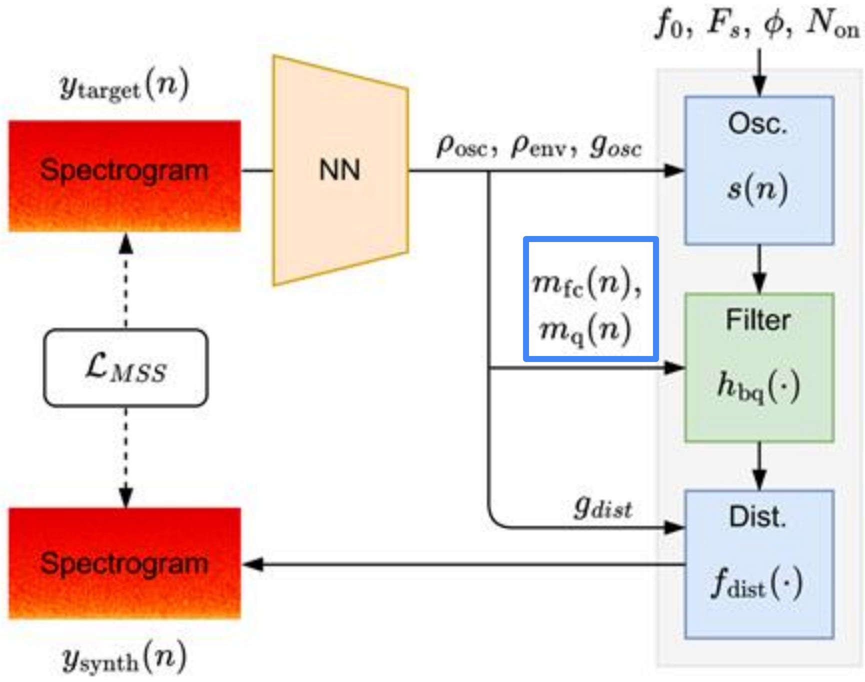
TD

Table 2: *Phaser evaluation results. "Method" refers to the training method; whereas the ESR was computed over the test dataset using both FS and TD at inference.*

Dataset	L/F_s	Method	ESR (%)	
			FS	TD
SS-A	10 ms	FS	1.46	1.53
		TD	**1.34**	**1.36**
SS-B	40 ms	FS	1.37	1.49
		TD	**1.35**	**1.34**
SS-C	160 ms	FS	**1.62**	**1.80**
		TD	2.56	2.23
SS-D	10 ms	FS	22.47	23.47
		TD	**21.64**	**23.33**
SS-E	40 ms	FS	15.43	16.69
		TD	**13.63**	**13.87**
SS-F	160 ms	FS	8.79	9.83
		TD	**7.83**	**8.79**

Roland TB-303 Acid Synth

- Sound matching
- 48 kHz and 32 sample hop size
- Modulating low-pass filter as time-varying biquad
- Configurations
  - TD (ours) vs. FS
  - Low-pass vs. general biquad coefficients
  - end2end LSTM



Synthesiser evaluation

- MSS
  - TD + Biquad Coeff. performs the best
- FAD
  - Biquad Coeff. + FS training + TD inferenc

Target Coeff. TD LSTM

Table 3: *Synth evaluation results (FS = frequency sampling, TD = time domain) with 95% confidence intervals for FAD scores.*

Filter	Method	N_{WIN}	MSS FS	MSS TD	FAD VGGish FS	FAD VGGish TD
Coeff.	FS	4096	1.66	1.78	2.62 ± 0.09	2.70 ± 0.13
		2048	1.64	1.65	$\mathbf{2.18 \pm 0.07}$	2.35 ± 0.11
		1024	1.53	1.58	2.57 ± 0.08	2.27 ± 0.12
		512	1.57	1.57	2.87 ± 0.10	2.46 ± 0.10
		256	**1.49**	1.48	2.25 ± 0.08	$\mathbf{1.98 \pm 0.06}$
		128	1.53	1.55	3.37 ± 0.14	2.73 ± 0.12
	TD	-	-	**1.38**	-	2.49 ± 0.21
LP	FS	4096	1.96	1.98	2.59 ± 0.06	$\mathbf{2.09 \pm 0.07}$
		2048	1.95	2.04	2.62 ± 0.07	4.52 ± 0.17
		1024	1.89	2.15	2.59 ± 0.08	4.18 ± 0.14
		512	1.83	2.92	$\mathbf{2.13 \pm 0.06}$	3.38 ± 0.08
		256	**1.82**	2.89	2.17 ± 0.06	3.36 ± 0.12
		128	1.84	2.70	2.34 ± 0.09	3.93 ± 0.12
	TD	-	-	**1.56**	-	2.51 ± 0.10
LSTM 64	TD	-	-	1.76	-	3.24 ± 0.07

LA-2A Leveling Amplifier

≈ feed-forward compressor from the DAFx textbook.

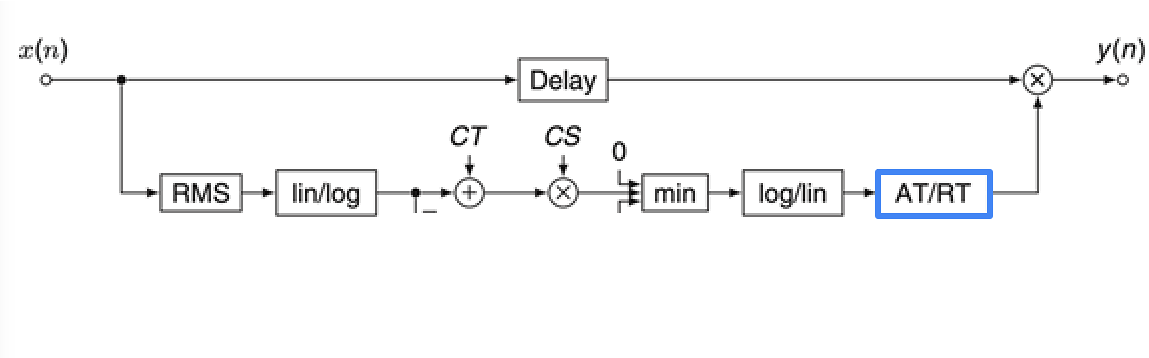
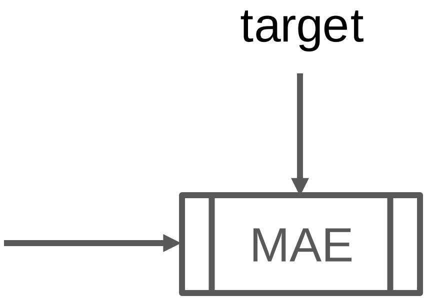


Figure 4.15 Block diagram of a compressor/expander [Zöl05].



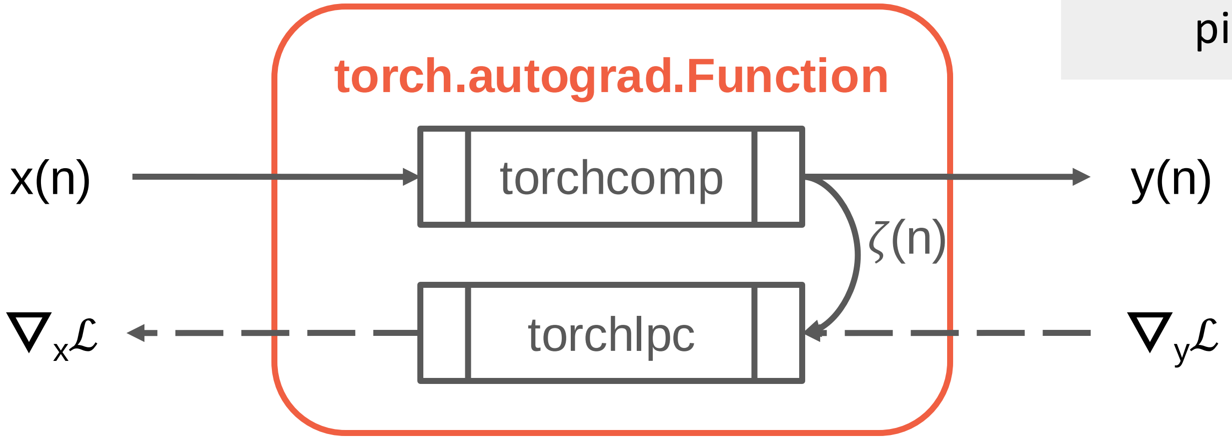
Differentiable attack/release

decision flag of attack/release phase

AT/RT ≡

$$\beta(n) = \alpha_{\mathrm{at}}^{\zeta(n)} \alpha_{\mathrm{rt}}^{1-\zeta(n)}$$
$$\hat{g}(n) = \beta(n) g(n) + (1 - \beta(n)) \hat{g}(n-1)$$

pip install torchcomp



Compressor evaluation

- Methods
  - FS (attack time = release time)
  - ∇FF (time-domain)
- Conditions
  - Feed-forward compressor (FF)
  - LA-2A (LA)
- 2-3 times faster

Table 5: *Summary of compressor ESR (%) evaluation.*

Method	FF-A	FF-B	FF-C	LA-D	LA-E	LA-F
FS	2.362	**0.00780**	4.649	11.29	9.485	7.783
∇FF	**0.015**	0.00785	**0.017**	**10.58**	**9.356**	**7.639**

Table 6: *The learned parameters for matching a LA-2A.*

Method	Data	R	CT (dB)	Attack	Release	α_{rms}	γ (dB)
FS	D	9.1	-11.66	489.43 ms		0.008	0.69
	E	231.1	-19.08	44.62 ms		0.606	0.34
	F	2.9	-26.00	0.06 ms		0.002	-0.81
∇FF	D	39.0	-26.58	99.41 ms	0.06 ms	0.703	0.74
	E	13.1	-12.41	5.68 ms	420.56 ms	0.978	0.54
	F	5.4	-20.14	2.24 ms	229.15 ms	0.973	-0.13

fast attack, slow release

Future Works

- Differentiable initial conditions
- Forward-mode automatic differentiation
- Higher-order gradients for advanced optimisation
- Extending to other effects, such as flanger, chorus, FDN, etc.

Q&A

Sound samples

Code

Neutone plugin