EMPIRICAL RESULTS FOR ADJUSTING TRUNCATED BACKPROPAGATION THROUGH TIME WHILE TRAINING NEURAL AUDIO EFFECTS

Yann Bourdin
Arturia
Montbonnot-Saint-Martin, France
Astral Inria Team, Inria Bordeaux
Talence, France
yann.bourdin@arturia.com

Pierrick Legrand
IMS, UMR CNRS 5218
ENSC, Bordeaux INP
Astral Inria Team, Inria Bordeaux
Talence, France
pierrick.legrand@ensc.fr

Fanny Roche
Arturia
Montbonnot-Saint-Martin, France
fanny.roche@arturia.com

ABSTRACT

This paper investigates the optimization of Truncated Backpropagation Through Time (TBPTT) for training neural networks in digital audio effect modeling, with a focus on dynamic range compression. The study evaluates key TBPTT hyperparameters – sequence number, batch size, and sequence length – and their influence on model performance. Using a convolutional-recurrent architecture, we conduct extensive experiments across datasets with and without conditioning by user controls. Results demonstrate that carefully tuning these parameters enhances model accuracy and training stability, while also reducing computational demands. Objective evaluations confirm improved performance with optimized settings, while subjective listening tests indicate that the revised TBPTT configuration maintains high perceptual quality.

1. INTRODUCTION

Audio effects are vital in shaping music, influencing timbre, dynamics, and spatial traits in both production and performance. Initially developed with analog circuitry, digital emulation is now important for its portability, flexibility, and lower cost. However, capturing analog devices' unique sonic traits digitally is challenging due to their nonlinear and time-dependent behaviors. Traditional modeling methods include white-box, gray-box, and black-box approaches [1]. White-box modeling uses precise mathematical descriptions of components, offering accuracy but requiring deep knowledge and high computational cost. Black-box methods replicate effects based on input-output data without detailed internal understanding. Gray-box models merge both approaches, integrating partial system knowledge with data-driven techniques. Deep learning has emerged as a promising black-box method for audio effect modeling, enabling neural networks to learn complex signal transformations only from input/output recordings, removing the need for handcrafted equations, extensive tuning, and in-depth hardware knowledge. Despite its potential, challenges remain in modeling effects with parameters and long time dependencies.

This work builds on [2], focusing on Dynamic Range Compression (DRC). A compressor modifies an audio signal's dynamics by applying time-varying gain reduction based on the input or sidechain signal's level. The nonlinear, time-dependent, time-invariant[1] nature and parameter conditioning of compressors make them relevant for this research. A compressor's behavior is mainly governed by four parameters: Threshold, defining the level above which gain reduction is applied; Ratio, determining attenuation degree; Attack time, controlling how quickly compression starts; and Release time, setting how gradually gain reduction is lifted when the signal drops below the threshold. Neural modeling of DRC has been explored using various architectures, including auto-encoders in the spectral domain [3], convolutional networks [4], recurrent networks [5], and convolutional-recurrent networks [6, 7, 2]. However, large attack and release times introduce long time dependencies, complicating modeling due to the need for handling extensive audio sequence lengths. In [2], the State Prediction Network (SPN) was introduced to efficiently train recurrent networks with long time dependencies. This convolutional neural network predicts the model's initial states, replacing the processing needed to warm up these states, thus reducing training times. However, state prediction can increase error in inference compared to training due to exposure bias [8], i.e., exposure to ground truth during training. This discrepancy motivates reducing reliance on the SPN through an approach derived from Truncated Backpropagation Through Time (TBPTT) [9], to be introduced in Section 2. While commonly used, TBPTT parameters are typically set empirically. In Section 3, our main contribution is to demonstrate, by training numerous models, the critical importance of properly setting these parameters alongside the batch size. Our results are validated by an objective evaluation and a subjective listening test.

2. METHODS

2.1. Model Architecture

The architecture we focus on, SPTMod (Series-Parallel Temporal Modulation), is a convolutional-recurrent network initially designed for DRC, as introduced in [2]. It is similar to the GCNT-FiLM model [7] but is lighter as it involves fewer nonlinear operations. As shown in Figure 1, SPTMod's backbone includes two paths: the modulation path and the audio path. The modulation path comprises blocks called ModBlock (typically 3 blocks, see Section 2.4.5) that calculate modulations for the audio path tensors via Feature-wise Linear Modulation (FiLM) operations [10]. These operations scale and shift tensor channels using modulation tensors. Since the target effect is a compressor, to ensure the final audio output is not degraded, the audio path performs only amplitude modulation through Temporal FiLM (TFiLM) operations.

[1] The compression parameters do not vary over time, but the gain reduction depends on the past of the input or sidechain signal.

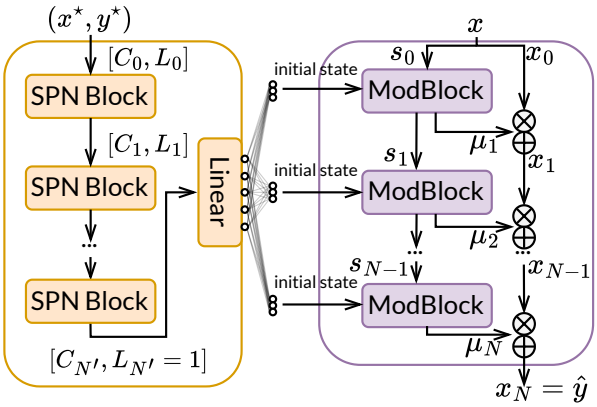


Figure 1: State prediction network (SPN) and SPTMod

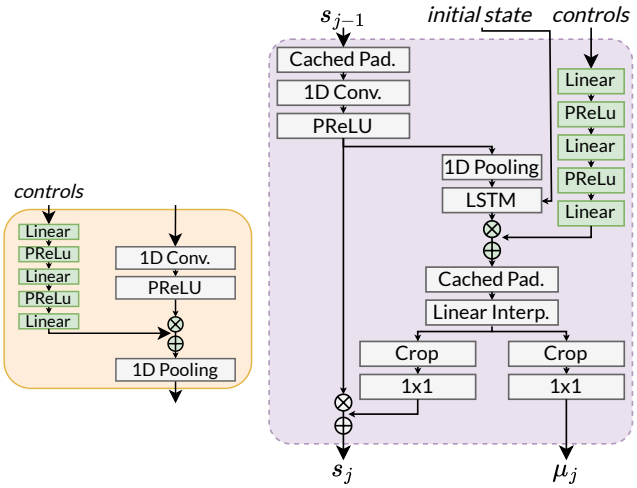


Figure 2: State Prediction Block (SPN Block, on the left side), and Modulation Block (ModBlock, on the right side).

Though the architecture avoids convolutional layers in the audio path, adding them could enable dynamic convolution for modeling other audio effects [7, 11]. Each modulation block, detailed in Figure 2, starts with a 1D convolutional layer followed by a Parametric Rectified Linear Unit (PReLU) activation. The output is processed by a TFiLM sub-block, which includes a pooling layer for downsampling, followed by a Long Short-Term Memory (LSTM) layer. This is complemented by a FiLM operation, transforming user controls via a two-layer neural network. A linear interpolation layer for upsampling concludes TFiLM to compensate for pooling. Two 1D convolutional layers with a kernel size of 1 ('1x1') adjust channel count before FiLM operations, yielding tensors s_j and x_j. Note that the LSTM layer, being recurrent, has an internal state whose initial value should be seen as an input to the modulation block. Cropping and cached padding layers are detailed in Section 2.4.1.

Proper initialization of the LSTM layers' initial states is crucial due to the effect's time dependency. We use the State Prediction Network (SPN), as introduced in [2] and shown in Figure 1, to address this. Typically, states are initialized with a warm-up method, processing a number of samples before gradients are registered. However, for effects with long time dependencies, even longer than the sequence length used for training, the warm-up length can become computationally expensive. The SPN suits these cases by efficiently reducing tensor size exponentially. The SPN receives inputs and reference outputs of lengths close to the effect's time dependency and predicts the state values the model would generate processing this input. Designed like a convolutional network performing classification, it consists of blocks, each with a 1D convolution, a PReLU activation, and a pooling layer, as seen in Figure 2. The SPN is also conditioned by user controls through the same FiLM method as used in the modulation blocks. After each pooling layer, the intermediary tensor's temporal length is divided by a factor, shrinking it exponentially with block depth. The final SPN block reduces the length to one, resulting in a batched vector. A final linear layer transforms this vector to match the sum of state sizes, allowing initial states of the model to be derived from slices of this vector.

An implementation of SPTMod is provided at the accompanying repository.[2]

2.2. Loss

The loss function used here mirrors [2], comprising two time-domain terms, Mean Absolute Error (MAE) and Error-to-Signal Ratio (ESR), a spectral-domain term, Multi-Resolution Short Term Fourier Transform (MR-STFT) spectral loss [12], and an energy-based term, Multi-Resolution Energy Error-to-Signal Ratio (MR-EESR) [2]. Equations (1) and (2) define the MAE and ESR:

$$\mathcal{L}_{\text{MAE}} = \frac{1}{L}\sum_{t=0}^{L-1} |\hat{y}[t] - y[t]| \quad (1) \qquad \mathcal{L}_{\text{ESR}} = \frac{\sum_{t=0}^{L-1}(y[t]-\hat{y}[t])^2}{\sum_{t=0}^{L-1} y[t]^2} \quad (2)$$

The STFT loss, given by (3), includes the STFT convergence and STFT magnitude terms:

$$\mathcal{L}_{\text{STFT}} = \frac{||\,|Y| - |\hat{Y}|\,||_F}{||\,|Y|\,||_F} + \frac{1}{L}||\log(|Y|) - \log(|\hat{Y}|)||_1 \quad (3)$$

where Y and $\hat{Y}$ are the STFTs of y and $\hat{y}$, and $||.||_F$ is the Frobenius norm. MR-STFT averages several STFT losses using different window sizes (512, 1024, 2048, i.e. 12, 23 and 46 ms with a sample rate of 44100 Hz). EESR is defined in (4):

$$\mathcal{L}_{\text{EESR}} = \frac{1}{K}\sum_{k=0}^{K-1} \frac{|\widehat{\text{E}_k} - \text{E}_k\,|}{\text{E}_k} \text{ with } \text{E}_k = \frac{1}{W}\sum_{\tau=kW}^{(k+1)W-1} y[\tau]^2 \quad (4)$$

where W is the window size and $K = \lfloor L/(W/4) \rfloor$, corresponding to a 75% overlap. MR-EESR averages several EESR losses with the same W values as MR-STFT. The final loss is the sum of $100\mathcal{L}_{\text{MAE}}$, $\mathcal{L}_{\text{ESR}}$, $\mathcal{L}_{\text{MR-STFT}}$, and $\mathcal{L}_{\text{MR-EESR}}$. The factor of 100 balances the magnitude of these terms at training start.

2.3. Dataset

2.3.1. Source Audio

The input audio files in our study consist of several sequences. Each file starts with a 1000 Hz pure tone, where the amplitude varies in 1 dB steps from -39 dB to 0 dB. These steps are organized into four groups of ten, each lasting 0.25 seconds, and separated by -40 dB steps lasting 0.5 seconds, creating a 16-second sequence. These quieter steps trigger the release phase of the compressor. Next, the file includes several 4-second excerpts randomly

[2] https://github.com/ybourdin/sptmod

selected from the Free Music Archive dataset [13], which contains diverse musical tracks. Each excerpt is normalized to a maximum peak of 0 dB, and this 40-second sequence's peak amplitude decreases gradually from 0 dB to -20 dB. The file concludes with two 20-second sequences of procedurally generated sounds. The first sequence has many sound events with few silence, while the second spaces the sound events with low-amplitude noise to capture the compressor's release phases. The sound events are created using randomized Attack-Decay-Sustain-Release envelopes modulating sounds among a white noise generator; an oscillator with three harmonics subject to random parameters and waveshaping; and finally linear and exponential chirps with randomized initial and target frequencies.

2.3.2. Parameter Sampling

Our work models the API-2500+ compressor, including most of its controls: Threshold, Attack, Ratio, Release, and Knee, plus the Thrust control, which enables high-pass filtering after gain detection. We focus on mono effects, so stereo controls are ignored. The compressor also has a 'tone type' control, which switches between feed-forward ('new') and feed-back ('old') configurations; our study only considers feed-forward. The parameters are discrete except the Threshold, continuous between -20 dB and +20 dB, which we discretize into 4 dB increments. We represent these controls as a vector of normalized values from 0 to 1.

Three datasets were recorded, with the described source audio but using different parameter sampling strategies. The first is a 'snapshot' dataset with 16 items recorded with a single configuration of controls. Here, all controls are set to their middle values except for the release control, which is set to maximum, posing a challenging task due to the long time dependency. The second dataset is a limited full-modeling set, where only the Threshold and Ratio parameters vary while other controls remain identical to those in the snapshot dataset. This dataset also contains 16 items, with Threshold values at 4, 0, -4, and -8 dB, and Ratio values at 3, 4, 6, and 10. A third dataset is the full-modeling dataset considered in [2], containing 160 items. Parameters in this dataset were sampled using Latin Hypercube Sampling, to facilitate a unique train-validation-test split, ensuring that each subset was representative of the control parameter distributions.

The recordings were made using an Arturia AudioFuse sound card, at a sample rate of 44100 Hz. The output of the sound card, which is the compressor input, is looped back to the sound card. The slight DC biases of the card's inputs are removed in post-processing by substracting the mean out of each recording.

2.3.3. Cross-Validation

In supervised learning, the goal is to train a model that performs accurately on unseen data from the same distribution as the training dataset, minimizing the expected risk $\mathbb{E}_{(x,y)\sim\mathcal{D}}[e(h(x), y)]$, where $\mathcal{D}$ is the data distribution, e is a metric, and h is the model [14]. The standard practice is dividing the dataset into three subsets: training, used by the optimization algorithm; validation, to monitor generalization and prevent overfitting; and test, for final evaluation. Selecting these subsets is essentially a realization of a random variable, as it involves sampling from the full data distribution $\mathcal{D}$. The training pipeline also faces variance from sources like initial model weights and batches constitution and order. Thus, evaluating a single model instance should be seen as a realization of a random variable [14], and training a model once does not reliably estimate risk or allow meaningful model comparisons. Cross-validation offers a better risk estimate by training multiple instances on different splits. However, due to the high training cost of deep networks, most studies train models once on a single split. In this work, we extracted 10 train-validation-test splits from each dataset by shuffling items before splitting. For the snapshot and Threshold-Ratio datasets, each with 16 items, we shuffled and split the items into 8 training, 4 validation, and 4 test items, repeating 10 times. For the third dataset (full-modeling) with 160 items, the list was split into 128 training, 16 validation, and 16 test items.

2.4. Training

2.4.1. Temporal Operations and Cached Padding

In convolutional neural networks, operations like convolutions and pooling alter the temporal length of inputs. Convolutions with kernel size k (dilation and stride 1) reduce input length by $k - 1$ samples, while pooling divides it by a factor. Though shorter outputs help in tasks like classification, audio effect modeling needs the model output to match the input length. Zero-padding is a common solution to this mismatch but can complicate inference, causing discontinuities at the start of output buffers when processing sequential sample buffers. To solve this, [15] suggests cached padding, storing the last samples of an input tensor in cache before applying a temporal operation, so these samples can be used for padding in subsequent operations.

Cached padding is effective with consecutive buffers but still relies on zero-padding when the cache is not initialized, e.g. at the start of a training batch. Using zeros instead of actual sample values can introduce bias, especially when large numbers of zeros are used relative to the input length. Moreover, models like Temporal Convolutional Networks (TCNs) [4] with large receptive fields may compute outputs predominantly from zeros rather than true signal values, necessitating large sequence lengths in training batches. In [2], zero-padding was avoided by calculating input samples required by a model to achieve the desired output length.

Models with binary operations involving 1D tensors, like skip connections, need identical tensor lengths. In SPTMod, this occurs with TFiLM operations following each ModBlock. Without padding, tensors often differ in length, necessitating cropping layers to trim the longer tensor (on its left-hand side, for causality) to match the shorter one. Determining input lengths and cropping sizes presents an optimization challenge. The goal is to minimize cropping sizes (which must be non-negative) while ensuring that the input tensor lengths of the binary operations are equal, and that the input lengths of pooling layers are multiples of the pooling size. An optimization algorithm can solve this for any architecture and fixed hyperparameters. For architectures like SPTMod, we derived an optimal solution, detailed at our accompanying repository, depending on the pooling size and the convolution hyperparameters.

2.4.2. Windowed Target and Streamed Target

The use of models with recurrent layers varies between training and inference. During training, models often process nonconsecutive batches of sequences, as datasets typically consist of long sequences that are sliced and shuffled during training. Consequently, internal states may not persist across iterations. The Windowed Target (WT) error [3] measures a metric (e.g., loss) like during training. In contrast, during inference, models still process inputs in a windowed fashion, yet states are not reset between batches,

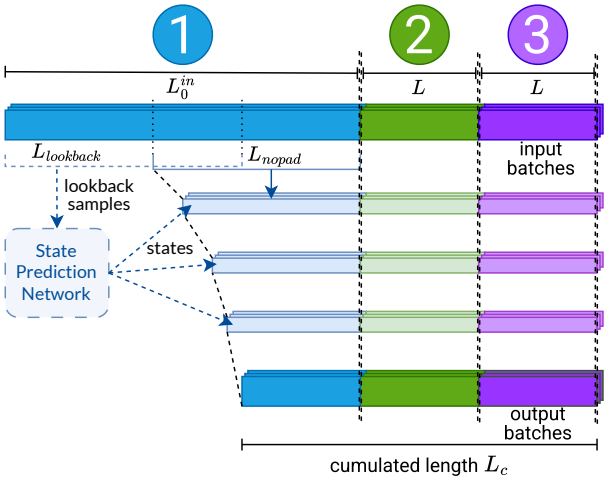


Figure 3: Diagram of intermediary tensor lengths for consecutive (non-overlapping) sequence batches in our TBPTT-based approach with $N = 3$. In the first iteration, no padding is applied, so the input length includes the samples needed for temporal operations. In subsequent iterations, states and caches are retained, but their gradients are detached from the computational graph.

corresponding to the Streamed Target (ST) setting. The ST error provides a more accurate user-observed metric. Ideally, minimizing WT error during training should generalize to minimizing ST error during inference, especially via techniques like warm-up or SPNs. However, previous work [2] showed ST error can increase compared to WT error when using state prediction. It was hypothesized this is due to exposure bias: models do not learn to recover from erroneous state values. Two strategies are possible to reduce reliance on the SPN, mitigating exposure bias: resetting states less often, and/or avoiding SPN overfitting. The first strategy can be done by increasing sequence length, though this raises computational costs and reduces weight update frequency. The second strategy uses regularization techniques like dropout to prevent overfitting state predictions.

2.4.3. Truncated Backpropagation Through Time

Recurrent networks can be trained by updating weights after each processed time step, but this is computationally expensive and limits the loss function to sample-to-sample differences. More commonly, recurrent networks process full sequences before updating weights with Backpropagation Through Time (BPTT) [16]. However, very long sequences result in infrequent weight updates.

Truncated BPTT (TBPTT) [9] improves efficiency by splitting long sequences into smaller ones, updating weights after processing each smaller sequence while preserving recurrent layer states.

We propose an optimization strategy based on TBPTT, detailed in Figure 3. This strategy introduces two hyperparameters: N, number of sub-sequences, and L, sub-sequence length, allowing us to define the cumulative length L_c as $L_c = N \cdot L$. The strategy divides training into groups of N iterations, processing a cumulative sequence with TBPTT. In the first iteration, padding is not used, so the input length L_{nopad} must be adjusted to achieve an output length equal to L. The SPN initializes states, requiring L_{lookback} samples. Thus, the input buffer length for the first iteration is $L_{\text{in}}^0 = L + \max(L_{\text{nopad}} - L, L_{\text{lookback}})$. The SPN receives the first L_{lookback} samples, while the effect processor network receives the last L_{nopad} samples and generates L output samples. In subsequent TBPTT iterations, the SPN is not used and cached padding is employed, making input length equal to output length L.

To form batches, the dataset is divided into long sequences of length $L_{\text{in}}^0 + (N-1) \cdot L$ with a step of L_c. At the start of each epoch, sequences are shuffled and batches formed. These batches, containing long sequences, are further sliced into N batches of shorter consecutive sequences of lengths $(L_{\text{in}}^0, L, ..., L)$. Every N iterations, states and caches are reset, since a new batch from a different long sequence is processed.

2.4.4. Training Procedure

The models and their training are implemented using the PyTorch deep learning library. We use the Adam optimizer with its default parameters and a learning rate set to 5×10^{-4}. Due to varying batch sizes and sequence lengths in this work, the number of batches per epoch also varies. At the end of each epoch, we measure the validation loss in both WT and ST modes. We use early stopping, halting training if the ST validation loss does not improve over 76800 iterations, with a cap of 1 million iterations.

2.4.5. Hyperparameters

SPTMod involves hyperparameters which are the number of modulation blocks and, within each block, parameters related to the convolutional layer: the number of channels, kernel size, and dilation size. Additional hyperparameters include the pooling size, LSTM hidden size, and the number of hidden neurons in the FiLM conditioning layer.

In [2], a hyperparameter search led to a setup with 3 modulation blocks. The convolutional layer in each block has 21, 19, and 32 output channels respectively, with kernel sizes of 9, 29, and 25. The pooling size is set to 95, the LSTM hidden size is 31, and the FiLM hidden layers have 26 neurons. The SPN has 7 blocks with 16 channels, a kernel size of 38, a pooling size of 4, and a hidden layer size of 8 in FiLM, achieving a lookback length of ∼5 seconds. This model configuration is called SPTMod24. Note that within our framework, the pooling size must be a divisor of the sequence length; thus, we set the pooling size to 64 instead of 95 in SPTMod24.

In this study, we explore different hyperparameters in a model called SPTMod25. This model has 4 modulation blocks, each with 15 output channels and a kernel size of 3 in the convolutional layers, and 32 neurons in the FiLM hidden layers for both modulation and SPN blocks. The other hyperparameters are identical to those used in SPTMod24. This configuration was chosen to investigate the impact of model depth over width, hypothesizing that increasing the number of blocks to 4 with simpler hyperparameter choices could offer similar or improved performance without the computational cost of an extensive hyperparameter search. The shrinkage in model width (number of channels) led to roughly a 10x reduction in multiply/add operations.

3. EXPERIMENTS AND RESULTS

3.1. Adjusting Truncated Backpropagation Through Time

TBPTT highlights two key hyperparameters: number of sequences N and sequence length L. The choice between BPTT or TBPTT, and specific values for N and L, is empirical and varies across studies modeling audio effects with recurrent layers. For instance,

[17] and [18] use TBPTT with warm-up lengths of 1000 and 4096 samples, and sequence lengths of 2048 and 8128 samples, achieving cumulative lengths of 0.5s and 2.5s, $N = 10$ and 13. Similarly, [19] uses TBPTT with both warm-up and sequence lengths of 1024 samples, resulting in a cumulative length of 1s and $N = 42$. In contrast, [6] and [7] do not use TBPTT or warm-up, with sequence lengths from 1024 to 8192 samples in [6] and set to 112640 in [7].

In our preliminary testing, we assessed a grid of N and L values, maintaining a fixed batch size of 16 and using the snapshot dataset. We found that training a neural effect in snapshot modeling led to faster and more stable convergence, with less variance than in full modeling. For smaller L values (2048-8192), the loss was notably high, likely due to low information content in a batch. Increasing the batch size by a factor of four resulted in a final loss closer to what was observed with larger L values. Based on this, we included batch size B as a hyperparameter in further experiments. The TBPTT experiment presented here includes both the snapshot and Threshold-Ratio datasets, to determine if the effects of these hyperparameters differ between snapshot and full modeling. We considered the Cartesian product of $N \in \{1, 2, 3\}$, $B \in \{8, 16, 32, 64, 128\}$, and $L \in \{4096, 8192, 16384, 32768\}$, excluding $(B, L) = (128, 32768)$ due to excessive memory requirements that could not be met by all nodes of our heterogeneous computing platform. For each dataset and each combination of N, B, and L, we trained 10 model instances, one for each train-validation-test split described in Section 2.3.3. Due to a substantial 570 training runs per dataset, we limited the experiment to the first two datasets, excluding the large full-modeling one.

Results

The main results of this experiment are shown in tables (1-6), which detail statistics on validation losses and training times for both the snapshot and Threshold-Ratio datasets across all considered hyperparameter configurations. Tables 1 and 2 present median validation loss values to estimate expected model accuracy under given (N, B, L) hyperparameters. Generally, median validation loss improves as N, B, or L increase. On both datasets, increasing batch size has a strong impact at low L_c values, but this effect fades as L_c rises, especially on the snapshot dataset when $L_c \geq 32768$. On the Threshold-Ratio dataset, batch size has more impact. Increasing L consistently enhances accuracy, while N tends to do so. Sometimes, $N = 1$ (i.e., without TBPTT) results in a lower median validation loss than higher N values.
Tables 3 and 4 show the Median Absolute Deviation (MAD) values (defined as the median of the absolute deviations from the data's median) of validation losses to estimate model training stability under specific (N, B, L). Although trends are harder to discern due to limited values for proper MAD evaluation, using TBPTT $(N > 1)$ typically improves MAD values. However, for high L (thus so is L_c), small MAD values are achieved with $N = 1$.

Tables 5 and 6 provide median training times for each configuration. Training duration is calculated by determining the number of iterations needed to reach the minimum validation loss within a 5% margin, then multiplying by the seconds per iteration factor. This factor was pre-evaluated in all configurations on a unique computer with an NVIDIA L40 GPU. Higher B and L values generally increase training time due to more computation per batch, but convergence can occur in fewer iterations and less time, as shown when $B = 8$ and L rises in Table 6. Increasing N often significantly improves training times because the SPN is invoked only at one iteration out of N.

Discussion

The choice of (N, B, L) greatly impacts the training pipeline performance, affecting accuracy, time, and stability. As predicted in [20], larger batch sizes surpassing a critical value yield diminishing returns, and the more complex the modeling task, the higher the critical batch size. For the snapshot dataset, this critical size is low, while it's high in full modeling – more than expected – as shown by the Threshold-Ratio dataset with 2 control parameters. Thus, one might expect even higher ideal batch sizes for full modeling datasets with more control parameters, like ours with 7 controls. A relevant compromise may be using large batch sizes with low sequence lengths and high N values. Training variance, measured by MAD values, is generally greater on the Threshold-Ratio dataset than the snapshot one, likely due to increased data diversity with more control parameters. Lastly, and unexpectedly, training time is not significantly shorter with snapshot than full modeling.

3.2. Objective Evaluation

Following the TBPTT experiment, SPTMod24 and SPTMod25 were trained and assessed on the large full-modeling dataset, under 3 specific hyperparameter sets for comparison: (N, B, L) = (1, 16, 16384), mirroring [2], and two top-performing configurations, (N, B, L) = (1, 64, 32768) and (N, B, L) = (3, 64, 32768).

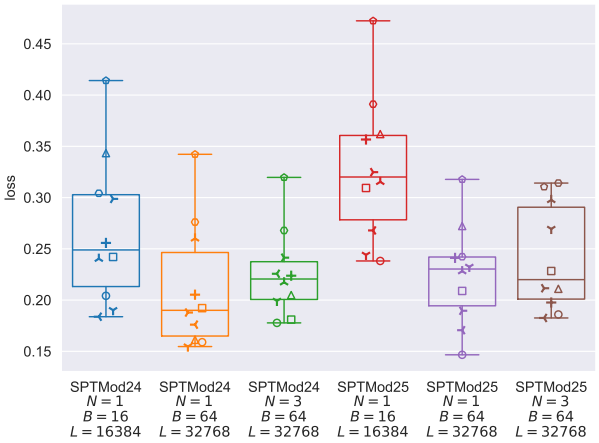


Figure 4: Loss after training the 6 models on 10 splits each, evaluated on their respective test subsets, depicted by markers.

Figure 4 shows all test loss values for the models considered. Under the same (N, B, L) values as in [2], where (N, B, L) = (1, 16, 16384), SPTMod25 performs worse than SPTMod24. However, with new training hyperparameters, SPTMod24 and SPTMod25 perform similarly, though SPTMod24 with (N, B, L) = (1, 64, 32768) appears to outperform the other models. We conclude from this objective evaluation that training hyperparameters N, B and L are at least as crucial as architectural ones.

3.3. Subjective Evaluation

A listening test was designed to assess the perceptual quality of the models. For the subjective evaluation, we selected (1) SPTMod24

$B \setminus L$ / L_c	4096 / 4096	8192 / 8192	16384 / 16384	32768 / 32768
8	3.70	3.18	2.20	1.46
16	3.48	2.36	1.61	1.10
32	4.77	2.92	1.32	1.18
64	3.09	2.49	1.33	1.21
128	2.58	2.32	1.55	

(a) N = 1

$B \setminus L$ / L_c	4096 / 8192	8192 / 16384	16384 / 32768	32768 / 65536
8	4.70	2.52	1.49	1.23
16	3.03	2.44	1.42	1.27
32	2.51	2.10	1.46	1.07
64	2.76	1.86	1.25	1.09
128	2.67	1.65	1.34	

(b) N = 2

$B \setminus L$ / L_c	4096 / 12288	8192 / 24576	16384 / 49152	32768 / 98304
8	3.54	2.67	1.64	1.28
16	2.68	1.90	1.29	1.14
32	2.67	1.85	1.27	1.27
64	2.83	1.81	1.25	1.13
128	2.47	1.96	1.25	

(c) N = 3

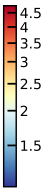


Table 1: Median values of the validation loss (x10) on the snapshot dataset.

$B \setminus L$ / L_c	4096 / 4096	8192 / 8192	16384 / 16384	32768 / 32768
8	6.44	5.20	4.37	3.60
16	6.54	4.83	4.05	2.68
32	5.91	4.80	2.75	1.55
64	5.73	3.83	3.31	1.53
128	5.86	3.39	1.99	

(a) N = 1

$B \setminus L$ / L_c	4096 / 8192	8192 / 16384	16384 / 32768	32768 / 65536
8	5.47	4.53	3.72	3.18
16	5.33	4.15	2.93	2.12
32	4.76	4.00	2.47	1.78
64	4.36	3.92	1.96	1.81
128	4.33	2.71	1.98	

(b) N = 2

$B \setminus L$ / L_c	4096 / 12288	8192 / 24576	16384 / 49152	32768 / 98304
8	4.83	3.75	3.41	2.52
16	4.17	3.64	2.96	1.87
32	4.49	3.77	2.52	1.89
64	4.30	3.65	2.06	1.38
128	4.20	2.45	1.92	

(c) N = 3

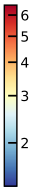


Table 2: Median values of the validation loss (x10) on the Threshold-Ratio dataset.

$B \setminus L$ / L_c	4096 / 4096	8192 / 8192	16384 / 16384	32768 / 32768
8	0.79	0.53	0.64	0.27
16	0.65	0.52	0.18	0.10
32	1.75	0.75	0.14	0.04
64	0.44	0.61	0.12	0.16
128	0.15	0.42	0.30	

(a) N = 1

$B \setminus L$ / L_c	4096 / 8192	8192 / 16384	16384 / 32768	32768 / 65536
8	1.37	0.26	0.12	0.05
16	0.50	0.73	0.08	0.08
32	0.28	0.61	0.19	0.03
64	0.42	0.29	0.14	0.05
128	0.40	0.29	0.03	

(b) N = 2

$B \setminus L$ / L_c	4096 / 12288	8192 / 24576	16384 / 49152	32768 / 98304
8	0.55	0.85	0.17	0.08
16	0.41	0.18	0.08	0.03
32	0.40	0.37	0.12	0.06
64	0.22	0.13	0.04	0.08
128	0.16	0.36	0.06	

(c) N = 3

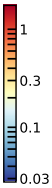


Table 3: Median Absolute Deviation (MAD) values of the validation loss (x10) on the snapshot dataset.

$B \setminus L$ / L_c	4096 / 4096	8192 / 8192	16384 / 16384	32768 / 32768
8	0.79	0.24	0.44	0.48
16	0.80	0.74	0.80	0.59
32	0.96	0.55	1.03	0.27
64	1.11	1.61	0.69	0.30
128	1.90	1.33	0.50	

(a) N = 1

$B \setminus L$ / L_c	4096 / 8192	8192 / 16384	16384 / 32768	32768 / 65536
8	0.44	0.59	0.42	0.27
16	0.62	0.51	0.51	0.38
32	0.52	0.51	0.54	0.49
64	0.47	0.56	0.28	0.24
128	0.54	0.56	0.25	

(b) N = 2

$B \setminus L$ / L_c	4096 / 12288	8192 / 24576	16384 / 49152	32768 / 98304
8	0.41	0.27	0.71	0.41
16	0.37	0.30	0.55	0.21
32	0.30	0.38	0.26	0.35
64	0.47	0.29	0.35	0.15
128	1.31	0.39	0.33	

(c) N = 3

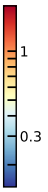


Table 4: Median Absolute Deviation (MAD) values of the validation loss (x10) on the Threshold-Ratio dataset.

$B \setminus L$ / L_c	4096 / 4096	8192 / 8192	16384 / 16384	32768 / 32768
8	0.40	0.28	0.74	1.16
16	0.46	0.60	1.03	2.83
32	0.50	0.82	2.84	3.11
64	1.20	1.98	8.05	9.82
128	3.95	2.71	4.36	

(a) N = 1

$B \setminus L$ / L_c	4096 / 8192	8192 / 16384	16384 / 32768	32768 / 65536
8	0.53	0.47	0.64	0.82
16	0.45	0.46	0.87	1.15
32	0.87	0.91	1.27	2.82
64	1.97	1.32	1.96	5.94
128	1.61	3.79	3.40	

(b) N = 2

$B \setminus L$ / L_c	4096 / 12288	8192 / 24576	16384 / 49152	32768 / 98304
8	0.31	0.61	0.53	1.26
16	0.44	0.57	1.06	1.13
32	0.49	0.78	1.14	1.39
64	0.88	1.87	3.63	4.51
128	2.20	3.44	8.02	

(c) N = 3

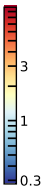


Table 5: Median values of the training time (in hours) on the snapshot dataset.

$B \setminus L$ / L_c	4096 / 4096	8192 / 8192	16384 / 16384	32768 / 32768
8	0.36	0.52	0.73	0.42
16	0.45	0.57	0.77	0.90
32	0.89	1.15	2.13	3.22
64	1.50	1.67	3.88	9.74
128	1.94	4.76	11.12	

(a) N = 1

$B \setminus L$ / L_c	4096 / 8192	8192 / 16384	16384 / 32768	32768 / 65536
8	0.40	0.62	0.46	0.35
16	0.43	0.42	0.88	1.10
32	0.58	0.99	1.35	1.89
64	1.20	0.95	2.47	3.93
128	2.13	2.62	6.14	

(b) N = 2

$B \setminus L$ / L_c	4096 / 12288	8192 / 24576	16384 / 49152	32768 / 98304
8	0.55	0.52	0.68	0.54
16	0.72	0.48	0.44	1.20
32	0.57	0.81	0.81	1.36
64	0.62	0.84	2.26	2.92
128	1.00	2.90	3.83	

(c) N = 3

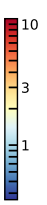


Table 6: Median values of the training time (in hours) on the Threshold-Ratio dataset.

with $(N, B, L) = (1, 16, 16384)$, as featured in [2]; (2) SPTMod25 with $(N, B, L) = (1, 64, 32768)$; and (3) a variant of a Temporal Convolutional Network (TCN) model [4], shown in Figure 5, with $(N, B, L) = (1, 64, 32768)$. This TCN variation computes a gain reduction signal applied to the input. It serves as a non-recurrent baseline, thus with no WT/ST difference, and is more comparable to our model.

We followed the MUSHRA method (MUlti Stimulus test with Hidden Reference and Anchor) [21], a listening test where participants rate the perceptual quality of multiple excerpts on a scale from 0 to 100, relative to a reference: the target effect's output. The test was conducted online via a webMUSHRA instance [22].[3] It includes outputs from the three models, a hidden reference identical to the provided reference, and an anchor processed through a rough compressor implementation. The hidden reference and anchor assess participant reliability and their ability to distinguish excerpts. The test featured 3 input audio excerpts: drumming, guitar strumming, and gypsy guitar playing. We evaluated 4 compressor configurations: low attack, low attack and release, high release, and high 'thrust'. Each audio and control combination was evaluated twice: once on a 4-beat excerpt (about 4 seconds) and once on a 1-beat segment (about 1 second) isolated from the excerpt. This approach ensures participants make decisions based on the same content. In total, 24 evaluation items were assessed. To reduce test duration, items were split into two groups of 12, with each participant receiving a random group and order, except that a 4-beat excerpt and its 1-beat version remain consecutive. Though the test was designed to last 15 minutes, participants completed the test with unlimited time and repetitions, and the median total rating time is 26 minutes. Afterward, they reported their audio device and answered questions about their expertise, particularly whether they had used a dynamic range compressor or had mixing or mastering skills. Of the 29 participants, 14 met the selection criteria for final results: positive responses to at least one expertise question, use of headphones or monitoring speakers, no more than three ratings below 90 for the hidden reference, and no more than two ratings above 90 for the anchor. This yielded 168 item evaluations.

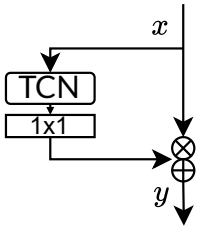


Figure 5: TCN variant in the listening test

Results

Figure 6 shows statistics on scores given to each model. The median score of SPTMod24 is 100, indicating this model, trained with the hyperparameters in [2], achieves excellent perceptual quality. SPTMod25 is slightly worse, with a median score between 90 and 95, which is satisfying given its smaller size and straightforward tuning. The TCN variant is the worst, still providing good quality but being easier to discern. As expected, the scores assigned to the hidden references among shorter loops (in orange) have lower variability than longer ones. Thus, more confidence should be put in the scores given on shorter loops. In Figure 7, scores are grouped by the parameter configurations described earlier, aligning with the overall mean results. On the high attack time configuration (7a), SPTMod24 achieves outstanding scores, while in the other configurations (7b, 7c, 7d) they are very high and similar to SPTMod25. Although TCN performs worse overall, it still achieves relatively high scores across most configurations, particularly in the high release setting, though it struggles with the high thrust configuration.

[3]Exposed at `https://ybourdin.github.io/sptmod/`

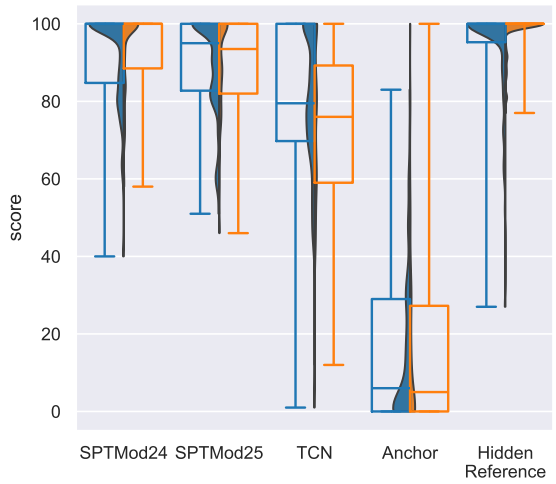


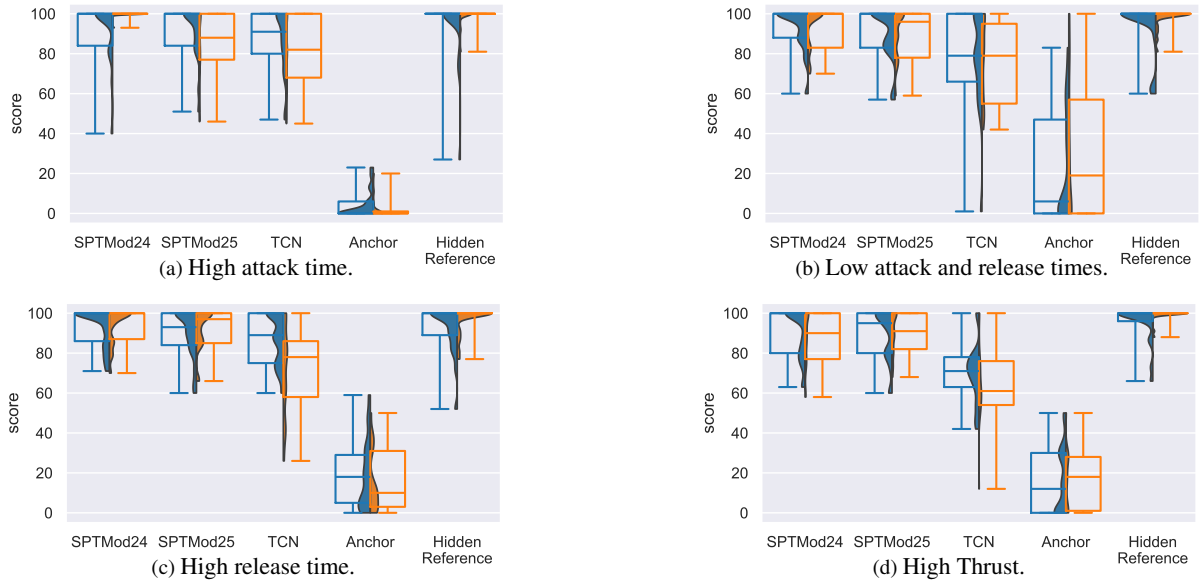
Figure 6: Scores of the listening test, split into the blue (longer loops — 4 beats) and the orange (shorter loops – 1 beat) groups.

4. CONCLUSION

This study investigated the parameters of TBPTT in the context of training a convolutional-recurrent network for modeling digital audio effects, focusing on DRC. Our main contribution is the empirical evaluation of TBPTT hyperparameters – namely, the number of sequences, batch size, and sequence length – and their effects on model accuracy, training stability, and computational efficiency. Extensive tests with both snapshot and full-modeling datasets show that increasing these hyperparameters generally improves model accuracy and reduces training variability, though it raises computational demands. Notably, TBPTT lowers validation loss variance and speeds up training, potentially allowing for larger batch sizes, which is beneficial for modeling multiple control parameters. These findings suggest that those hyperparameters are as crucial as architectural choices for optimal performance. Objective evaluation confirms that models trained with newly optimized TBPTT parameters perform better than prior setups. However, a subjective listening test revealed that even previously, SPTMod24 had excellent perceptual quality, indicating the revised TBPTT setup did not significantly impact the model's perceptual quality. For future work, implementing the model in C++ will allow for a precise estimation of its computational efficiency and real-time viability, as well as enabling multi-objective hyperparameter search that considers both model accuracy and computational cost.

5. ACKNOWLEDGMENTS

This work is part of a Cifre PhD project funded by ANRT. It benefited from access to the computing resources of the 'PLaFRIM' and 'CALI 3' clusters. PLaFRIM is supported by Inria, CNRS (LABRI & IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine. CALI 3 is operated by the University of Limoges and is part of the HPC network in the Nouvelle-Aquitaine Region, financed by the State and the Region.



(a) High attack time.

(b) Low attack and release times.

(c) High release time.

(d) High Thrust.

Figure 7: Scores of the listening test, grouped by parameter configuration, split into the blue (longer loops – 4 beats) and the orange (shorter loops – 1 beat) groups.

6. REFERENCES

[1] M. Comunità, C. J. Steinmetz, and J. D. Reiss, "Differentiable black-box and gray-box modeling of nonlinear audio effects," *arXiv preprint arXiv:2502.14405*, 2025.

[2] Y. Bourdin, P. Legrand, and F. Roche, "Tackling Long-Range Dependencies in Dynamic Range Compression Modeling via Deep Learning," in *Int. Conf., Evolution Artificielle*, 2024.

[3] S. Hawley, B. Colburn, et al., "Profiling audio compressors with deep neural networks," in *Audio Engineering Society Convention*, Oct 2019.

[4] C. Steinmetz and J. Reiss, "Efficient neural networks for real-time analog audio effect modeling," in *Audio Engineering Society Convention*, 2022.

[5] R. Simionato and S. Fasciani, "Fully conditioned and low-latency black-box modeling of analog compression," in *Proc. of the Int. Conf. on Digital Audio Effects (DAFx)*, 2023.

[6] M. Ramírez, *Deep Learning for Audio Effects Modeling*, PhD Thesis, Queen Mary University of London, 2021.

[7] M. Comunità, C. Steinmetz, et al., "Modelling Black-Box Audio Effects with Time-Varying Feature Modulation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

[8] A. Peussa et al., "Exposure Bias and State Matching in Recurrent Neural Network Virtual Analog Models," in *Proc. of the Int. Conf. on Digital Audio Effects (DAFx)*, 2021.

[9] J. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[10] E. Perez et al., "FiLM: Visual reasoning with a general conditioning layer," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[11] Y. Chen et al., "Dynamic convolution: Attention over convolution kernels," in *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, 2020, pp. 11030–11039.

[12] C. Steinmetz and J. Reiss, "Auraloss: Audio focused loss functions in PyTorch," in *Digital Music Research Network One-day Workshop*, 2020.

[13] M. Defferrard, K. Benzi, et al., "FMA: A dataset for music analysis," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2017.

[14] X. Bouthillier, P. Delaunay, et al., "Accounting for variance in machine learning benchmarks," *Proceedings of Machine Learning and Systems*, vol. 3, 2021.

[15] A. Caillon et al., "Streamable Neural Audio Synthesis With Non-Causal Convolutions," in *Proc. International Conference on Digital Audio Effects (DAFx)*, 2022.

[16] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[17] A. Wright et al., "Real-time black-box modelling with recurrent neural networks," in *Proceedings of International Conference on Digital Audio Effects (DAFx)*, 2019.

[18] A. Wright and V. Valimaki, "Grey-box modelling of dynamic range compression," in *Proceedings of International Conference on Digital Audio Effects (DAFx)*, 2022.

[19] O. Mikkonen et al., "Sampling the user controls in neural modeling of audio devices," *EURASIP Journal on Audio, Speech, Music Processing*, vol. 2024, no. 1, pp. 26, 2024.

[20] S. McCandlish, J. Kaplan, D. Amodei, and O. D. Team, "An empirical model of large-batch training," *arXiv preprint arXiv:1812.06162*, 2018.

[21] B. Series, "Method for the subjective assessment of intermediate quality level of audio systems," *Int. Telecommunication Union Radiocommunication Assembly*, vol. 2, 2014.

[22] M. Schoeffler et al., "webmushra—a comprehensive framework for web-based listening tests," *Journal of Open Research Software*, vol. 6, no. 1, 2018.