Université de Paris
45 Rue des Saints-Pères
75006 Paris

ENS Paris-Saclay
4 Avenue des Sciences
91190 Gif-sur-Yvette

Steelseries France
19 Rue de la Ladrié
59491 Villeneuve-d'Ascq

INTERNSHIP REPORT[1]

Learning rate scheduling and gradient clipping for audio source separation

Student :

Mateo Vial
Université de Paris
mateo.vial@etu.u-paris.fr

Supervisors :

Nathan Souviraà-Labastie
Steelseries France
nathan.souviraa-labastie@steelseries.com

Roland Badeau
Télécom Paris
roland.badeau@telecom-paris.fr

Held on September 16, 2022

[1]modified version for HAL publication

Contents

Introduction

This internship focuses on audio source separation with deep neural networks. After a phase of bibliography at the beginning of the internship, particularly on the state of the art of audio source separation, we mostly investigated and experimented methods allowing to make the learning phases of the models more efficient, as much on the learning speed than on the performances of the trained models.

The Torch custom learning rate scheduling code is available on GitHub [1].

1 About the company

1.1 General presentation

Steelseries is a Danish company created in 2001 and initially specialised in gaming equipment (headsets, keyboards, peripherals). Steelseries also produces softwares and audio solutions meant for gamers, in particular since the acquisition of the french company **A-Volute** in 2020. The latter has then been renamed to **Steelseries France** and integrated to the Business Unit Software, the software branch of Steelseries.

1.2 Use of audio source separation

The project for which the company uses audio source separation is a project of speech enhancement. This task consists in removing the surrounding noises which could be detrimental to the quality of the speech. Typically, but not exclusively, keyboards and mouse noises, because the product is mostly meant for gamers.

2 Theory and state of the art of audio source separation

The following section talks about the theoretical aspect of audio source separation and its state of the art. We will also define the essential keywords about deep learning.

2.1 Bases of audio source separation

2.1.1 Notions of mixture and separation

When N_s audio sources are simultaneously recorded with a number C of sensors, we assume that the sources signals $(\mathbf{s}_i)_{i \in [\![1,N]\!]}$ add up to form the mixture $\mathbf{x}$.

$$\mathbf{x}^c(t) = \sum_{i=1}^{N_s} \mathbf{s}_i^c(t) \tag{1}$$

It is worth noting that though there are N_s sources, the $\mathbf{s}_i^c$ correspond to the spatial image of the source s_i on sensor c, which is the theoretical recording of the isolated source s_i as detected by the sensor c. The spatial image of a given source may vary depending on the distance to the sensor, the acoustics of the room, or the reverberation. A source $\mathbf{s}_i$ thus corresponds to a number C of signals $(\mathbf{s}_i^c)_{c\in[\![1,C]\!]}$, one for each sensor.

In the case of a monophonic mixture ($C = 1$), we free ourselves from this notation by directly denoting the spatial image of source i on the sensor by $\mathbf{s}_i$:

$$\mathbf{x}(t) = \sum_{i=1}^{N_s} \mathbf{s}_i(t) \tag{2}$$

In order to simplify the notations, we will only consider monophonic mixtures because the internship did not focus on multi-channel separation.

Audio source separation is the task which consists in isolating one or several of the sources $(\mathbf{s}_i)$ given a mixture $\mathbf{x}$. Audio source separation is an essential task in a vast majority of audio processing applications: denoising, speech enhancement, speech recognition, etc.

2.1.2 Complexity of audio source separation

From a mathematical point of view, we can distinguish two cases, depending on the number of sources and the number of channels (sensors).

- When the number of channels is greater or equal to the number of sources, the problem of audio source separation corresponds to an overdetermined system, because the (linear) transformation corresponding to the mixture is injective.
- On the contrary, if the number of channels is less than the number of sources, the problem is an underdetermined system because the transformation is no longer injective: several different sets of sources could theoretically output the same mixture. It is especially the case when dealing with monophonic separation.

For those reasons, the task is much more complex in the underdetermined case, and statistical methods using neural networks have won their spurs (cf. Section 2.2).

2.1.3 Intuition and time-frequency representation

While audio separation presents no difficulty for a human brain, it is much less convenient for a machine to separate sources in a pure signal (waveform). The brain has a strong tendency to visualise audio signals in a time-frequency representation (e.g., a sheet music). When a person performs a mental audio separation, they proceeds in the same way: they perceive different frequencies and are able to isolate them. Indeed, unless it is noise, it is common that two sources of different natures have different frequencies.

This intuition is also valid in signal processing, which is why a large majority of audio separation algorithms use a time-frequency representation as an intermediate for separation. To separate sources, it is very intuitive to estimate their respective contributions at each time-frequency bin of the mixture, then mask them accordingly.

The most widely used time-frequency representation in many areas of audio processing is the STFT (Short-Time Fourier Transform). It is a Fourier transform operated on a small signal segment multiplied by a fast decay window W to compensate for discontinuities.

$$\mathbf{X}(f,n) = \sum_{t \in T} \mathbf{x}(t) W(n-t) \exp(-\mathrm{i}ft) \tag{3}$$

The linearity of the Fourier transform and of the STFT mean that the separation problem is formulated in a similar way in the time-frequency domain:

$$\mathbf{X}(f,n) = \sum_{i=1}^{N_s} \mathbf{S}_i(f,n) \tag{4}$$

where $\mathbf{X}$ and $\mathbf{S}_i$ denote respectively the STFTs of the mixture $\mathbf{x}$ and of the source $\mathbf{s}_i$.

2.1.4 A base model: Encoder-Separator-Decoder

In order to deal with source separation in the time-frequency domain, one form of models commonly used in separation consists of the following three blocks:

- Encoder: The encoder transforms a signal into its time-frequency representation. Most often, we use a STFT because it has the advantage of being simple to program and to execute. In this case, as the STFT returns a complex number, one extracts real-valued features of the STFT (for example, the magnitude) which will be used to do the separation. The encoder can also be a layer of neurons. In this case, as traditional neural networks deal with real values, the computation of the features will not be necessary.

- Separator: The separator takes the time-frequency representation of the signal as input and returns a vector of masks of same dimension. These masks are then multiplied to the time-frequency representation of the mixture to isolate the audio sources of interest and mask the unwanted sources.

  The separator is usually a neural network consisting of recurrent layers (cf. Section 2.2.3), and is designed to take real numbers as input.

- Decoder: The decoder takes as input the time-frequency representation to which the separation masks have been applied, and re-synthesises a signal from it. Similarly to the encoder, we use an inverse STFT. But as for the encoder, it is also possible to use a layer of neurons. In this case, it does not necessarily imply that the decoder corresponds to the exact inverse operation of the encoder. Additionally, the work of Luo and Mesgarani [5, 6] shows that keeping the inverse operator of the encoder as a decoder is less efficient than keeping free layers.

In general, the encoder and decoder are simple in terms of structure and computational cost. The separator has a significantly higher cost, because the task which it performs is mathematically more complex.

The STFT as an encoder/decoder has been adopted because of its simplicity of implementation and execution, however it poses a problem: as the masks are calculated only on the magnitudes of the STFT values, the phases of the STFT are not processed by the separator. Thus, during decoding, we synthesise a signal with the new modules masked by the separator, and the phases of the original signal. The reconstruction cannot be perfect if we are forced to keep the phase information of the mixture.

2.2 Audio source separation and neural networks

2.2.1 Introduction to deep learning

The usual components of a deep learning model are the following:

- Forward function: this is the function that we want to apply to our data, which is made of layers of neurons. This function corresponds to the task we want the model to learn. For example, in the case of audio separation, this function takes a mixture as input and hopes to return the same signal from which we have isolated the sources of interest. The forward function is parameterised by the parameters of the model, i.e. the parameters have an influence on the output of the function, without playing the role of an input.

- Database: the database contains all the files that will be used during the training phase. It is divided into two parts: the training files and the validation files.
- Loss function: the loss function is a function that takes as input all the parameters of the model and returns an integer value quantifying the quality of the parameters, i.e. how well the model does its task. To do this, it compares the outputs of the forward function with the ground truths, for a large number of files in the database. It is this function that will be optimised during the learning phase.

During the learning phase, the files of the database are divided into batches of same size. Each batch is processed individually to optimise the loss function.
The learning phase can be compared to an optimisation algorithm by gradient descent, with the difference that between two different batches, the loss is not the same in theory because it is not calculated on the same data. The parameters of the model are then updated at each batch. The step of optimising the loss function once on all the batches of the trainset is called an epoch.

2.2.2 TasNet as a reference DNN architecture

The encoder/separator/decoder structure described above has the advantage of being highly modular, in the sense that all three parts can be modified independently to create new separation models. A notable innovation has been the replacement of the encoder and decoder by small neural networks, composed of convolutional layers in 2017 by Luo and Mesgarani: this is the TasNet model (*Time-domain Audio Separation Network*).

In order to imitate the STFT as well as possible, the signal is first split into segments of constant size (which may overlap), then each segment is processed independently to produce a time-frequency representation. The segmentation operation is reversible and is performed during the synthesis at the output of the decoder.
Hence, one hyper-parameter to define is the dimension of the time-frequency representation (the size of the filterbank in the case of learned representation). It will be denoted N in all the following models derived from TasNet. We will use a default value of $N = 500$ for those models.

2.2.3 LSTM as a separator

The use of neural networks has showed to be effective in the field of audio separation, particularly thanks to recurrent neural networks (RNN). RNNs are a type of neural network where connections between nodes can create a cycle, i.e. nodes which output can affect their own input. RNNs are particularly well suited to data with a strong time dependency, which is the case for time-frequency representations of audio signals. One RNN structure in particular, the LSTM (Long Short-Term Memory, Figure 1) is widely used in many areas of audio processing: speech recognition [8, 4], source separation [5, 6, 11], etc.

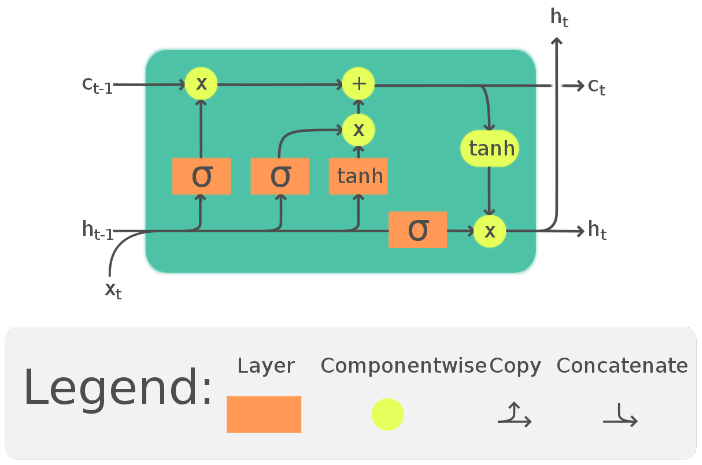


Figure 1: *Long Short-Term Memory. Illustration from [3].*

The LSTM block includes a cell (c_t), a hidden layer (h_t) as well as four gates (i_t, o_t, g_t, f_t: input, output, cell, forget). Without going into details, the cell remembers signal values during arbitrary time intervals, while the gates define the flow of information which goes in and out of the cell. Here is the detail of the operations performed by the LSTM block:

$$\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \otimes c_{t-1} + i_t \otimes g_t \\
h_t &= o_t \otimes \tanh(c_t)
\end{aligned} \tag{5}$$

where the variables in W and b correspond to the LSTM parameters (weights and biases), and $\otimes$ denotes the entrywise product. The memory of variable length and the different layers acting on this memory allow the LSTM to remember the important information as the signal progresses in time, and to ignore the less useful information, allowing a better processing of signals with a strong time dependency.

2.2.4 Learning rate scheduling

In an standard optimisation problem by gradient descent algorithm, at each iteration, the gradient of the objective function is computed, and then the solution of the algorithm is updated by adding the gradient multiplied by a quantity called step. The purpose of the step is to quantify the speed of progress of the algorithm.

While it is possible to use a constant step size to approach the solution, this strategy is generally not adopted, because a convergent sequence has infinitesimally small successive differences. Depending on the optimized function, with a constant step, there is often an oscillation between different values on either side of the true solution. This is even more noticed when the step size is too large.

A simple solution to this problem is to change the step size as the algorithm converges. Once the iterate is close enough to the true solution to the point where no improvement is seen over the iterations, the step size is decreased in order to approach the solution more finely.

Since a learning algorithm is simply a gradient descent algorithm, with some additional subtleties (the fact that the objective function theoretically varies between each iteration, as the data is divided into different batches), there is a parallel between these two concepts: the step is the learning rate, the iterates (variables) are the parameters of the model, and the objective function is the loss function of the model. Thus, the same reasoning can be intuited for neural networks, which introduces the notion of learning rate scheduling.

Learning rate scheduling is the action of making the learning rate vary over the course of the learning phase. The objective is not only to decrease the learning rate as mentioned before, but to vary it in a more general way, which includes the possibility of increasing it. The scheduling can be determined in advance, in which case the learning rate is defined as a function of the temporal progress of the learning (usually, as a function of the epochs). It can also be adaptive, which means that the schedule is not defined in advance, but rather that a new learning rate (after each *epoch*, for example) is computed as a function of the learning results, such as the value of the loss (typically on the loss computed on the cross-validation set).

Within the state of the art of audio source separation, a few publications mention learning rate scheduling [7, 2, 10], and extensive research on the subject is very rare.

In Section 3.3, we will present the work on learning rate scheduling that has been done during this internship.

2.2.5 Autoclip

A common problem encountered in deep learning is gradient explosion. When it occurs, the learning is unstable and the performance of the neural network is very limited. This phenomenon occurs when the computed gradients have values that are excessively large. This may be due to the complex geography of the loss function in high dimension, or to an inadequate model.

An often used way to solve this problem is to perform a gradient clipping, i.e., to cut the gradient as soon as it goes over a predefined threshold. The advantage of this method is that it requires almost no effort to implement, but on the other hand, this threshold needs to be manually chosen and remains fixed during the whole learning phase.
An inappropriate threshold will not give good performances: a too small threshold will cut the gradient too often, and the updates of the model parameters will be of low amplitude, so the network will learn too slowly. On the contrary, a too large threshold will be close to a learning phase without gradient clipping because the gradient will be very rarely cut, and the effect of too large gradients will be often felt.

Autoclip [9] is an automatic and adaptive gradient clipping algorithm, with which there is no longer the need to define the clipping threshold manually, and the algorithm may adapt it automatically according to the measured gradients. We will confirm later (cf. Section 3.4) that this implementation indeed produces improvement in performances, as already depicted in [9] (all other things being equal).
Autoclip works as follows: the history of measured gradients[2] is kept in memory. At each iteration (epoch), the new clipping threshold is set to the p-percentile of the gradient history, where p is a variable hyper-parameter from 0 to 100. This allows the threshold to adapt to the measured gradients amplitude.

The role of p is to quantify how strict the clipping mechanism is. The more the learning phase progresses, the more the proportion of clipped gradients converges to p %. With $p = 0$, the threshold is defined as the smallest measured gradient since the beginning of the learning. With $p = 100$, the threshold is defined as the largest measured gradient. Seetharaman et al. show empirically that the best choice is around $p = 10$.

[2]As a reminder, the gradients are calculated at each iteration (or batch), and not at each epoch.

Autoclip has the advantage of being insensitive to the size of the data and to the scale of the loss: the parameter p has a universal role for all tasks.

In Section 3.4, we will show the results obtained after re-implementing Autoclip.

3 Conducted experiments

This section contains all the experiments that were done during the internship.

3.1 Experimental setup

3.1.1 Evaluation

Objective evaluation The loss function that we use the most is the Signal to Distortion Ratio (SDR). The SDR is simply a way of comparing two signals quite similarly to an MSE (Mean Square Error).

Subjective evaluation While the loss function defined above quantify well enough the basic aspects of the quality of an audio separation, as the quality increases it is not possible to quantify it precisely by mathematical means, as the quality of an audio separation is very subjective. This is why we always use subjective listening to judge the quality of a mathematically validated model.

Ideally, a MUSHRA (Multiple Stimulus with Hidden Reference and Anchor) test is conducted, which is a cross-test where different results from different models are being played to a bunch of human listeners in a random order to eliminate bias. However, such a test is costly to implement at every experimental step.

3.1.2 Databases

For the experiments, we exclusively used a learning database, called *V3 Beta ulight*, sometimes shorten to *V3 ulight*, with speech and ambient noises (background noise, door noises, animals, keyboard noises, etc.) mixed at various SNR in order to perform speech enhancement. We exclusively use a batch size of 16 to avoid biases across experiments, which amounts to a total of 1384 batches with audio excerpts of 15 seconds.

3.1.3 Previous internal scheduler

Until now, the company's computers used to run a custom-coded scheduler that took into account the evolution of the measured gradient and divided the learning rate by 2 when the gradient did not decrease anymore, in addition to adding an automatic stopping criterion. This algorithm is used in the experiments of the Section 3.2.

This scheduler had defects, both in the reduction of the learning rate and in its stopping criterion, and was not easily modifiable.

From Section 3.3, we modify the functioning of the learning phase by generalizing the behavior of the learning rate and implementing a more convenient scheduler formalism.

Since the models (several million parameters) and the databases (hundreds of hours of audio sampled to 48 kiloHz) are very large, the training phases are long and can take a whole day for the smallest databases.

3.2 Re-implementation of a state-of-the-art separator: E3Net

At the beginning of my internship, a first task was to implement a structure from a paper that had just been published by Microsoft. This work allowed me to familiarise myself with the company's work environment and the neural network structures used in audio processing.
E3Net is a source separation model based on the encoder-separator-decoder architecture. E3Net is primarily intended for personalized speech enhancement, incorporating a component called spaker embedding. This component contains speaker-specific features and allows for better performance in separating the voice of that particular speaker.
We did not reproduce the custom enhancement part, but we re-implemented the separation model. We used it along with the encoder and decoder of TasNet.

We use an experiment already conducted on TasNet as a comparison. An initial learning rate of 2.5e$-$4 has been found after doing experiments on a grid of initial learning rates in order to find one that is suitable for this new model. However, we do not show the intermediate experiments in this report.

The results are shown in Figure 2 and more details are in appendix. We have performed experiments on the $E3Net$ model with $N = 500$ and $N = 2048$ as filterbank sizes for the encoder (see Section 2.2.2). In both cases, the performances are very close to the comparative experiment conducted on TasNet.
Subsequently, the experiments on E3Net will be done with the parameter $N = 500$ in order to homogenise the results with the experiments previously conducted by the company.

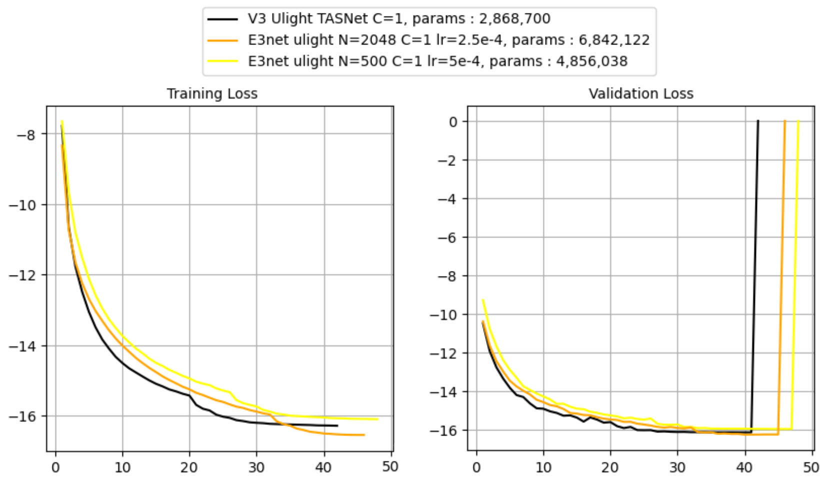


Figure 2: *Training and validation curves of the experiments on E3net with N=500 and N=2048 (yellow and orange respectively). Training curve on TasNet with N=500 for comparison in black. The experiment on TasNet was carried out before the beginning of the internship. The displayed loss functions are SDR.*

3.3 Learning rate scheduling

We present the two basic scheduling building blocks that we will use in this section.

3.3.1 Reduce on Plateau: ROP2 [3]

The first scheduler that we introduce is an adaptive scheduler that we call ROP2 for "Reduce On Plateau" (Figure 4). This elementary scheduler multiplies the learning rate by a factor strictly between 0 and 1 with the following rule. A counter is initialized to zero. After each epoch, the loss is evaluated. If its value is less satisfactory than that of the loss evaluated at the previous epoch, the counter is incremented by 1. If the counter reaches a fixed value named patience, the learning rate is multiplied by the factor, and the counter is reset to zero. The parameters of this scheduler are the factor and the patience.

From now on, in all the figures and for all the schedulers to come, we will adopt the notation $Scheduler(\cdot, \cdot, \cdots)$ where the first argument is the initial lr and the following arguments are the parameters of the scheduler as they will have been described and in that specific order. For example, $ROP2(x, y, z)$ corresponds to a ROP2 scheduler of initial learning rate x, with a factor y and a patience z.
Figure 3 gives a comparison of the two models (E3net and TasNet) with the ROP2 scheduler.

[3]Torch code available at : `https://github.com/SteelSeries/torch_custom_lr_schedulers`

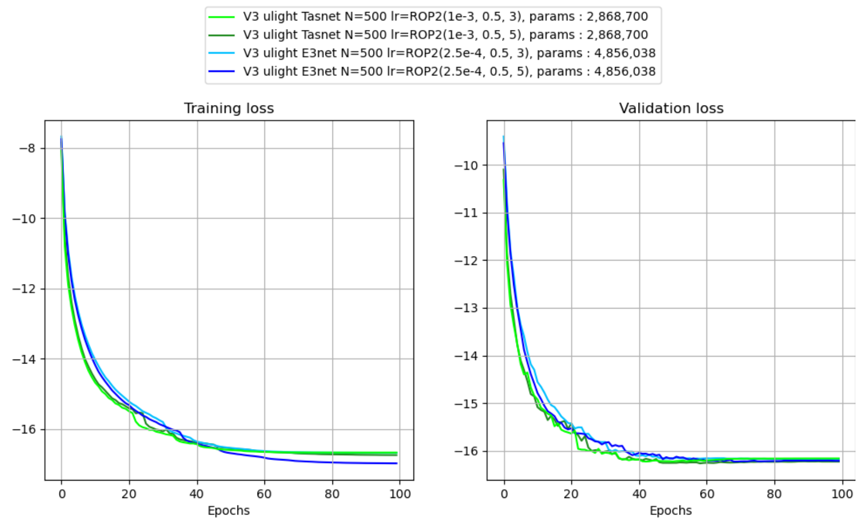


Figure 3: *Comparison of experiments carried out on TasNet (green curves) and E3Net (blue curves) with the ROP2 on two different sets of parameters. The two architectures use a different initial learning rate ($1\mathrm{e}{-3}$ for TasNet, $2.5\mathrm{e}{-4}$ for E3net). Apart from that, the schedulers have identical parameters two by two. The displayed loss functions are SDR.*

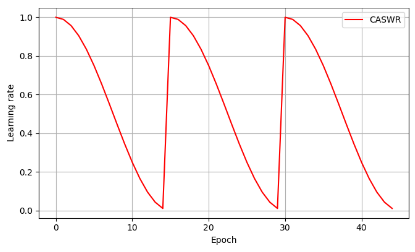
3.3.2 Cosine Annealing Scheduler with Warm Restarts: CASWR

We mentioned in the introduction to schedulers that the learning rate was not obliged to decrease. We call annealing the action of increasing the value of the learning rate at a given moment of the learning phase. The point of annealing is not to continuously increase the value for multiple epoch in a row, as this would be of no interest for the learning process, but to increase the value only for one epoch.

A gradient descent can only expect to approach a *local* minimum (*non-gobal*) of a function. In this case, the learning of a neural network can be compared to a gradient descent on a function which input space dimension is several million, and which is likely to present a large number of non-global minima.
Once the parameters of the model are in a non-global minimum of the loss function, it is very difficult, if not impossible, to leave that minimum by following the gradient with small steps. The only way to get out of it is to make a larger step, hence the interest of annealing.

The CASWR scheduler (Cosine Annealing Warm Restarts) defines the learning rate as a cosine function of time (Figure 4). "Warm restarts" refers to the fact that the function is discontinuous at the time of annealing. The periodic sequence of learning rates is precisely defined as follows:

$$\left(\cos\left(\frac{x}{period} \times \pi\right) + 1\right) \times \frac{lr\_initial - lr\_min}{2}, \qquad x = 0, 1, \ldots, period - 1 \tag{6}$$



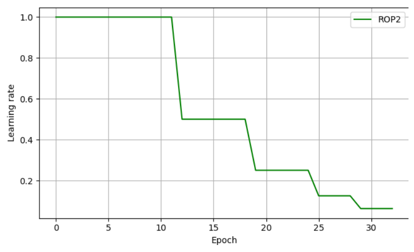


Figure 4: *Evolution of the learning rate with schedulers* $CASWR(1,0,15)$ *(left) and* $ROP2(1,0.5,3)$ *(right). Since the ROP2 scheduler is adaptive, its learning rates curve may to vary from one experiment to another.*

where lr_min and $period$ are the parameters of the scheduler.

We now show in Figure 5 training phases using both of our schedulers.

On the validation curves, we notice that the experiments conducted with the ROP2 stagnate very quickly: from about 50 epochs onwards, there is no further improvement, even when decreasing the learning rate. This is probably due to a convergence towards a non-global minimum. On the contrary, the validation curves of the CASWR experiments seem to continue to improve their performance even after a large number of epochs, which justifies experimentally the interest of annealing.
On the other hand, experiments conducted with ROP2 seem to be faster to reach their lowest point.

These two elementary building blocks (ROP2 and CASWR) both have disadvantages and advantages. Subsequently, we create other techniques of scheduling inspired by these two schedulers.

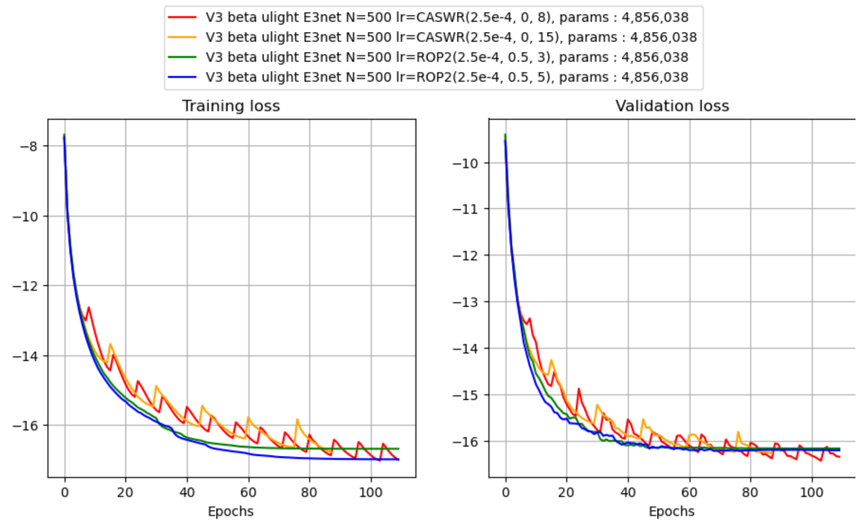


Figure 5: *Trainings on E3net with the ROP2 and CASWR schedulers, with two different sets of parameters for each scheduler. The green and blue curves represent respectively trainings with ROP2 with a patience of* 3 *and* 5 *respectively. The red and orange curves represent training with CASWR of periods* 8 *and* 15 *respectively. The displayed loss functions are SDR.*

3.3.3 Combining ROP2 and CASWR [4]

A first idea to combine these two schedulers is to first perform a learning phase with the CASWR in order to benefit from the annealing, then finish with a ROP2 phase to converge finely towards a minimum. For the sake of experiment, we made an experiment with the TasNet model and the CASWR scheduler for a duration of 300 epochs. We then continued it with the ROP2 scheduler at different times of the experiment (resp. at epochs 70, 120, 176, 239) (Figure 6).
The results are remarkable: as soon as the ROP2 takes over, the loss immediately show values well below the red curve. This can be seen in particular on the yellow and green curves on the validation graph.

Thereafter, in order to harmonise the comparisons, the experiments will all be conducted on TasNet.

[4]Torch code available at : `https://github.com/SteelSeries/torch_custom_lr_schedulers`

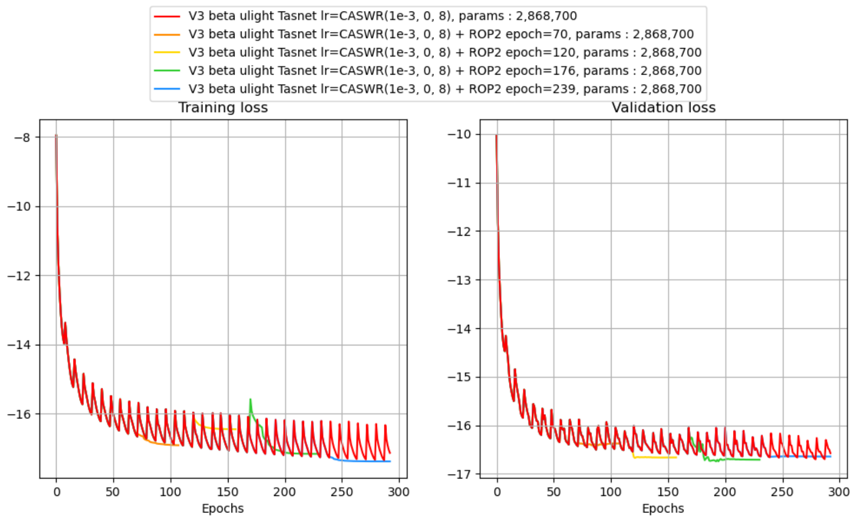


Figure 6: *Experiments on a TasNet model consisting of a first CASWR phase and a second ROP2 phase. The ROP2 phases of the orange, yellow, green and blue curves are started respectively after 70, 120, 176 and 239 epochs of the first CASWR phase. The displayed loss functions are SDR.*

3.4 Gradient clipping and other schedulers

We re-implement Autoclip [9], an adaptive gradient clipping algorithm (see Section 2.2.5). In the rest of this section, we continue to create new schedulers used in conjunction with Autoclip.
We first compare (Figure 7) the use of Autoclip with the ROP2 scheduler presented in the previous section. On the training curve, we can see that the green and blue curves, corresponding to the experiments with Autoclip, are below the black curve, and this from the beginning of the experiments, which validates the theoretical superiority of Autoclip. The validation curves show the same trend. However, similar experiments on the CASWR scheduler (Figure 8) did not show any notable improvements compared to the previous one. From now on, we will always use Autoclip in the upcoming experiments.

We also present other experiments with ROP2 schedulers of different parameters (again with Autoclip), but without any learning benefits (Figure 9).

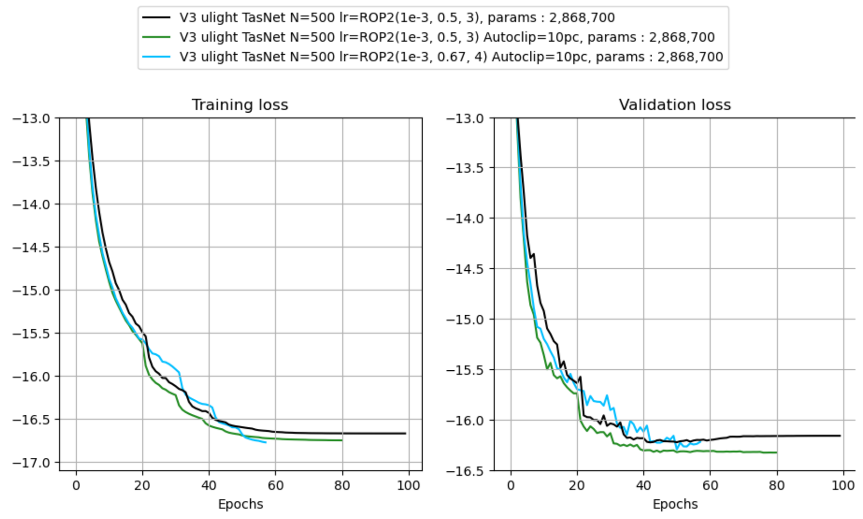


Figure 7: *Comparison of a few training phases with scheduler ROP2, with and without Autoclip. The displayed loss functions are SDR.*

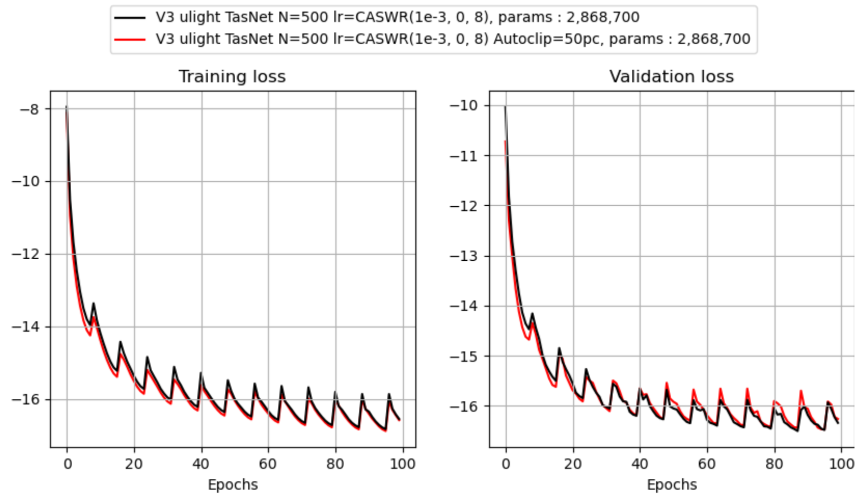


Figure 8: *Comparison of a few training phases with scheduler CASWR, with and without Autoclip. The displayed loss functions are SDR.*

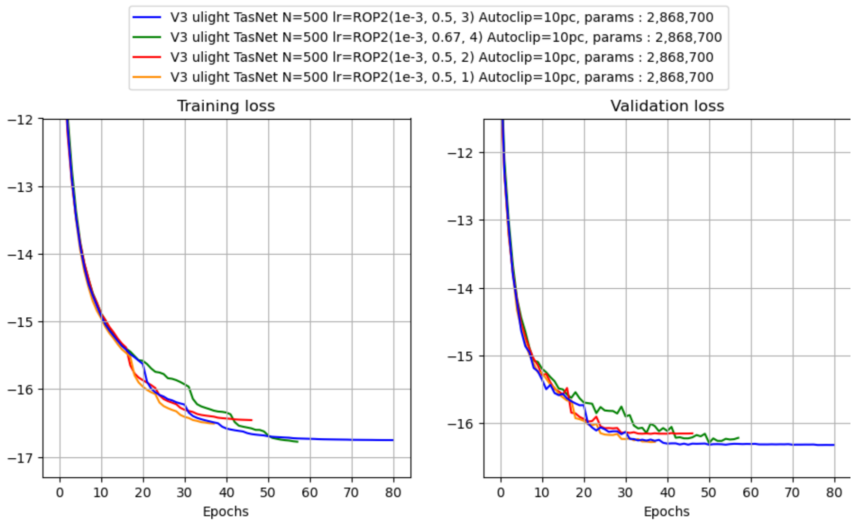


Figure 9: *Other experiments on TasNet with schedulers ROP2 of various sets of parameters. The displayed loss functions are SDR.*

We now present other experiments with the last scheduler mentioned in the previous section, i.e. a training composed of a CASWR phase followed by a ROP2 phase (Figure 10). The three curves (resp. green, blue and orange) use schedulers consisting of a first CASWR phase followed by a second ROP2 phase. The periods of the CASWR phases of the three experiments are different (resp. 8, 5, 10). The parameters are adapted so that the three CASWR phases last approximately 50 epochs. The results are not showing improvement compared to the ROP2

In the previous section (Figure 6), we trained such a scheduler (CASWR then ROP2) over an unrealistic duration of 300 epochs for the sake of the experiment, and found interest in mixing these two schedulers. Here, on more realistic durations (50 to 100 epochs), the interest is less obvious because the curves do not benefit from enough annealing to be sufficiently different from an experiment without annealing. This is all the more validated as we had already noticed that the CASWR joint with Autoclip was not particularly more efficient.

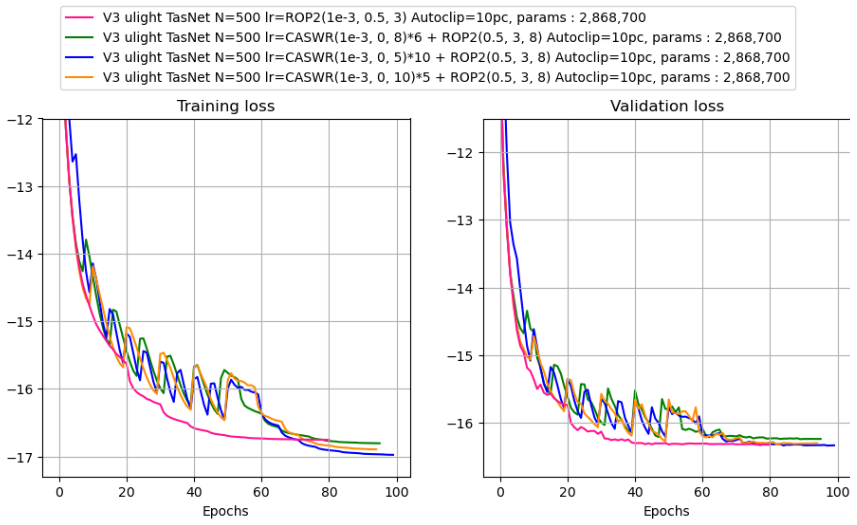


Figure 10: *Various experiments on the TasNet architecture comparing the ROP2 scheduler with the CASWR+ROP2 scheduler. The displayed loss functions are SDR.*

Subsequently, we abandon the cosine scheme in the annealing phase because the CASWR scheduler does not seem to benefit from Autoclip.

Annealing with ROP2: ChainROP [5]

We implement a way of doing annealing without using the cosine scheme, but rather a scheme similar to the tried and tested ROP2. We call this new scheduler ChainROP (Figure 11). It starts with a classic ROP2 phase, then once the lr has been reduced a certain number of times, it is reset to its starting value and a new ROP2 phase with the same parameters begins. In short, the ChainROP performs a "chain" of ROP2 schedulers, hence its name

This scheduler does annealing, because the learning rate is reset to its starting value after being reduced a certain amount of times.

Such a scheduler requires additional parameters, in addition to those required by ROP2: the number of reductions of the learning rate before which it must be reset, and the number of chained ROP2 phases, this last parameter playing the role of stopping criterion.
For example, $ChainROP(x, y, z, n, p)$ represents a ChainROP scheduler based on a $ROP2(x, y, z)$ where the learning rate will be reset to x after n reductions, for a total of p times.

The experiments (Figures 12 and 13) show a total of 4 experiments with ChainROP schedulers of different parameters. The results are satisfactory: one of the ChainROP parameterization manages to perform slightly better than a training with ROP2 in a *reasonable* number of epochs.

[5]Torch code available at : `https://github.com/SteelSeries/torch_custom_lr_schedulers`

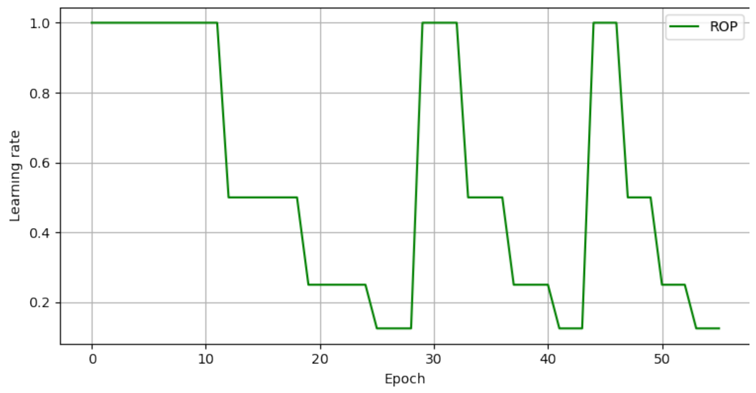


Figure 11: *Evolution of the learning rate with scheduler ChainROP(1, 0.5, 3, 4, 3). This scheduler is made up of three successive ROP2 phases. The first phase is always longer than the others, because it is at the beginning of the learning process that the network learns the most. Thereafter, improvements in the loss over the epochs are rarer, so the learning rate is more frequently reduced.*

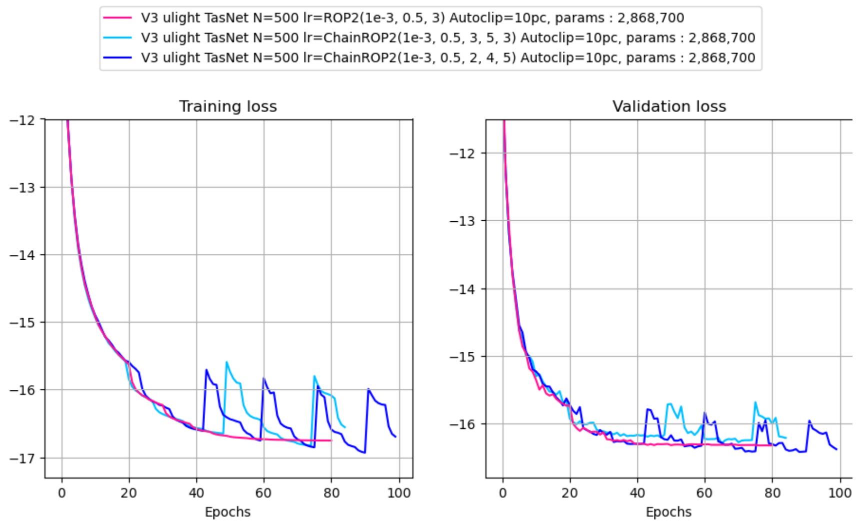


Figure 12: *Comparing of an experiment with scheduler ROP2 (pink curve) and two experiments with scheduler ChainROP (blue curves). The displayed loss functions are SDR.*

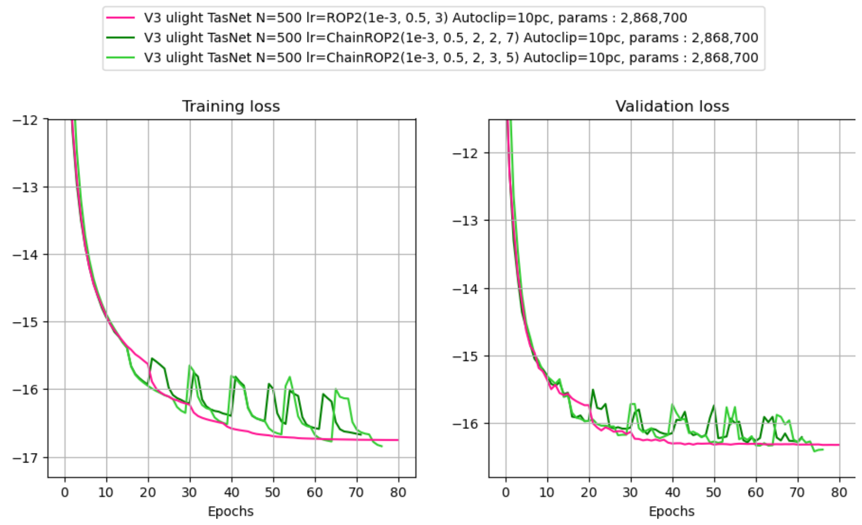


Figure 13: *Comparing of an experiment with scheduler ROP2 (pink curve) and two experiments with scheduler ChainROP (green curves). The displayed loss functions are SDR.*

Conclusion

We explored a range of learning rate schedulers and created a few new ones. The use of schedulers is not new, but we have tried to go further in the study of that topic. We distinguish two main classes of elementary schedulers: the schedulers which make the learning rate converge towards 0 (this is the case of ROP2), and the periodic schedulers which introduce annealing (this is the case of CASWR). We have seen experimentally that it is interesting to combine these two classes of schedulers to obtain the best performances.

The Torch custom learning rate scheduling code is available on GitHub [1].

Empirical take away

- TasNet and E3Net perform equally
- Autoclip improve training for ROP2 scheduler
- Autoclip does not improve training for CASWR scheduler
- ROP2 is suited for short training
- CASWR is suited for long training and an additionnal ROP2 phase can improve the results if it is launch at the very end
- We proposed the chainROP scheduler of which one parameterization slightly improves the results on middle term training compared to ROP2

Appendix

Minimum loss and corresponding epoch per experiments

* denotes experiments not present in the Figures.

		- SDR	Epoch
Figure 2	V3 ulight TasNet N=500 lr=1e-3	-16.15	39
	V3 ulight E3net N=2048 lr=2.5e-4	-16.27	39
	V3 ulight E3net N=500 lr=2.5e-4	-16.02	45
Figure 3	V3 ulight TasNet N=500 lr=ROP2(1e-3,0.5,3)	-16.22	42
	V3 ulight TasNet N=500 lr=ROP2(1e-3,0.5,5)	-16.26	58
	V3 ulight E3net N=500 lr=ROP2(2.5e-4,0.5,3)	-16.17	56
	V3 ulight E3net N=500 lr=ROP2(2.5e-4,0.5,5)	-16.22	68
Figure 5	V3 ulight E3net N=500 lr=CASWR(2.5e-4,0,8)	-16.45	119
	V3 ulight E3net N=500 lr=CASWR(2.5e-4,0,15)	-16.27	87
	V3 ulight E3net N=500 lr=ROP2(2.5e-4,0.5,3)	-16.17	56
	V3 ulight E3net N=500 lr=ROP2(2.5e-4,0.5,5)	-16.22	68
Figure 6	V3 ulight TasNet N=500 lr=CASWR(1e-3,0,8)	-16.7	279
	V3 ulight TasNet N=500 lr=CASWR(1e-3,0,8)+ROP2(1e-3,0,3) (epoch 70)	-16.42	78
	V3 ulight TasNet N=500 lr=CASWR(1e-3,0,8)+ROP2(1e-4,0,3) (epoch 120)	-16.67	121
	V3 ulight TasNet N=500 lr=CASWR(1e-3,0,8)+ROP2(1e-4,0,3) (epoch 176)	-16.74	192
	V3 ulight TasNet N=500 lr=CASWR(1e-3,0,8)+ROP2(5e-4,0,3) (epoch 239)	-16.65	239
	V3 ulight TasNet N=500 lr=CASWR(1e-3,0,4)*	-16.65	211
	V3 ulight TasNet N=500 lr=CASWR(1e-3,0,4)+ROP2(1e-3,0.5,3) (epoch 60)*	-16.34	85
	V3 ulight TasNet N=500 lr=CASWR(1e-3,0,4)+ROP2(2.5e-4,0,3) (epoch 60)*	-16.36	68
Figure 7	V3 ulight TasNet N=500 lr=1e-3 Autoclip=10pc*	-16.41	153
	V3 ulight TasNet N=500 lr=ROP2(1e-3,0.5,3)	-16.22	42
	V3 ulight TasNet N=500 lr=ROP2(1e-3,0.5,3,10) Autoclip=10pc	-16.33	76
	V3 ulight TasNet N=500 lr=ROP2(1e-3,0.67,4,4) Autoclip=10pc	-16.29	50
Figure 8	V3 ulight TasNet N=500 lr=CASWR(1e-3,0,8)	-16.7	279
	V3 ulight TasNet N=500 lr=CASWR(1e-3,0,8,1) Autoclip=50pc	-16.58	174
Figure 9	V3 ulight TasNet N=500 lr=ROP2(1e-3,0.5,3,10) Autoclip=10pc	-16.33	76
	V3 ulight TasNet N=500 lr=ROP2(1e-3,0.67,4,4) Autoclip=10pc	-16.29	50
	V3 ulight TasNet N=500 lr=ROP2(1e-3,0.5,2,7) Autoclip=10pc	-16.16	36
	V3 ulight TasNet N=500 lr=ROP2(1e-3,0.5,1,7) Autoclip=10pc	-16.28	36
Figure 10	V3 ulight TasNet N=500 lr=ROP2(1e-3,0.5,3,10) Autoclip=10pc	-16.33	76
	V3 ulight TasNet N=500 lr=CASWR(1e-3,0,8,1)*6+ROP2(1e-3,0.5,3,8) Autoclip=10pc	-16.24	89
	V3 ulight TasNet N=500 lr=CASWR(1e-3,0,5,1)*10+ROP2(1e-3,0.5,3,8) Autoclip=10pc	-16.34	100
	V3 ulight TasNet N=500 lr=CASWR(1e-3,0,10,1)*5+ROP2(1e-3,0.5,3,8) Autoclip=10pc	-16.32	82
Figure 12	V3 ulight TasNet N=500 lr=ROP2(1e-3,0.5,3,10) Autoclip=10pc	-16.33	76
	V3 ulight TasNet N=500 lr=ChainedROP2(1e-3,0.5,3,5,3) Autoclip=10pc	-16.3	65
	V3 ulight TasNet N=500 lr=ChainedROP2(1e-3,0.5,2,4,5) Autoclip=10pc	-16.42	88
Figure 13	V3 ulight TasNet N=500 lr=ROP2(1e-3,0.5,3) Autoclip=10pc	-16.33	76
	V3 ulight TasNet N=500 lr=ChainedROP2(1e-3,0.5,2,2,7) Autoclip=10pc	-16.29	58
	V3 ulight TasNet N=500 lr=ChainedROP2(1e-3,0.5,2,3,5) Autoclip=10pc	-16.42	74

References

[1] The torch$_{c}ustom_{l}r_{s}chedulersgithubrepository.$.

Andreas Bugler, Bryan Pardo, and Prem Seetharaman. A study of transfer learning in music source separation. *CoRR*, abs/2010.12650, 2020. URL https://arxiv.org/abs/2010.12650.

Guillaume Chevalier. LARNN: linear attention recurrent neural network. *CoRR*, abs/1808.05578: 3, August 2018. URL http://arxiv.org/abs/1808.05578.

Xiangang Li and Xihong Wu. Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. *CoRR*, abs/1410.4281, 2014. URL http://arxiv.org/abs/1410.4281.

Yi Luo and Nima Mesgarani. Tasnet: time-domain audio separation network for real-time, single-channel speech separation. *CoRR*, abs/1711.00541, 2017. URL http://arxiv.org/abs/1711.00541.

Yi Luo and Nima Mesgarani. Tasnet: Surpassing ideal time-frequency masking for speech separation. *CoRR*, abs/1809.07454, 2018. URL http://arxiv.org/abs/1809.07454.

Gerard Roma, Emad M Grais, AJ Simpson, Iwona Sobieraj, and Mark D Plumbley. Untwist: A new toolbox for audio source separation. In *Extended abstracts for the late-breaking demo session of the 17th international society for music information retrieval conference, ismir*, pages 7–11, 2016.

Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *CoRR*, abs/1402.1128, 2014. URL http://arxiv.org/abs/1402.1128.

Prem Seetharaman, Gordon Wichern, Bryan Pardo, and Jonathan Le Roux. Autoclip: Adaptive gradient clipping for source separation networks, 2020. URL https://arxiv.org/abs/2007.14469.

Daniel Stoller, Sebastian Ewert, and Simon Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation. *CoRR*, abs/1806.03185, 2018. URL http://arxiv.org/abs/1806.03185.

Manthan Thakker, Sefik Emre Eskimez, Takuya Yoshioka, and Huaming Wang. Fast real-time personalized speech enhancement: End-to-end enhancement network (e3net) and knowledge distillation, 2022. URL https://arxiv.org/abs/2204.00771.