Optimization of fast deep learning models for audio analysis and synthesis

Alexandre Défossez

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Institut National de Recherche en Informatique et Automatique

Optimization of Fast Deep Learning Models for Audio Analysis and Synthesis

Soutenue par
Alexandre Défossez
Le 7 juillet 2020

École doctorale n°ED 386
École Doctorale Sciences Mathématiques de Paris Centre

Spécialité
Mathématiques

Composition du jury :

Rachel Ward	University of Texas, Austin	*Rapporteure*
Axel Roebel	IRCAM	*Rapporteur*
Emmanuel Dupoux	EHESS/PSL/INRIA/FAIR	*Président du jury*
Francis Bach	INRIA/PSL	*Directeur de thèse*
Léon Bottou	FAIR	*Co-directeur de thèse*
Nicolas Usunier	FAIR	*Co-directeur de thèse*

MINES ParisTech | PSL

Contents

1. Introduction **9**

1.1. A short primer on sound 11
1.1.1. From the physical world to the digital waveform 12
1.1.2. The time-frequency representation of sound 13
1.1.3. The vocoder: an analysis/synthesis framework 14
1.1.4. The content of sound: periodicity and noise 15
1.1.5. Conclusion 17
1.2. Neural audio synthesis of music instruments 17
1.2.1. From concatenative models to end-to-end generation 17
1.2.2. Using spectrograms for invariant audio synthesis 20
1.2.3. Contributions 21
1.2.4. Conclusion 22
1.3. Music source separation in the waveform domain 23
1.3.1. Blind source separation 24
1.3.2. Supervised source separation 25
1.3.3. Contributions 28
1.3.4. Conclusion 30
1.4. Stochastic Optimization 31
1.4.1. Empirical Risk Minimization 31
1.4.2. Stochastic gradient descent 32
1.4.3. Adaptive optimization methods 35
1.4.4. Non convex optimization 36
1.4.5. Contributions 38
1.4.6. Conclusion 39
1.5. Summary of contributions 40

2. SING: deep neural generator for music notes **41**

2.1. Introduction 41
2.2. Related Work 43

2.3. The spectral loss for waveform synthesis . . . 45
2.3.1. Mean square regression on the waveform . . . 45
2.3.2. Spectral loss . . . 45
2.4. Model . . . 47
2.4.1. LSTM sequence generator . . . 47
2.4.2. Convolutional decoder . . . 48
2.4.3. Training details . . . 48
2.5. Experiments . . . 49
2.5.1. NSynth dataset . . . 49
2.5.2. Generalization through pitch completion . . . 50
2.5.3. Human evaluations . . . 51
2.6. Conclusion . . . 54

3. Demucs: music source separation in the waveform domain 55
3.1. Introduction . . . 55
3.2. Related Work . . . 58
3.3. Adapting Conv-Tasnet for music source separation . . . 59
3.4. The Demucs Architecture . . . 61
3.4.1. Convolutional auto-encoder . . . 61
3.4.2. Loss function . . . 64
3.4.3. Weight rescaling at initialization . . . 64
3.4.4. Randomized equivariant stabilization . . . 65
3.5. Experimental setup . . . 66
3.5.1. Evaluation framework . . . 66
3.5.2. Baselines . . . 66
3.5.3. Training procedure . . . 67
3.6. Experimental results . . . 67
3.6.1. Comparison with baselines . . . 70
3.6.2. Ablation study for Demucs . . . 71
3.7. Conclusion . . . 72

4. Adabatch: an efficient gradient aggregation rule for sparse optimization 73
4.1. Introduction . . . 73
4.1.1. Problem setup . . . 76
4.1.2. Related work . . . 77
4.2. AdaBatch for SGD . . . 78
4.3. Convergence results . . . 80

4.4. Wild AdaBatch . . . 83
4.5. AdaBatch for SVRG . . . 84
4.6. Experimental results . . . 85
4.7. Conclusion . . . 87

5. On the Convergence of Adam and Adagrad **91**
5.1. Introduction . . . 91
5.2. Setup . . . 93
5.2.1. Notation . . . 93
5.2.2. Adaptive methods . . . 93
5.2.3. Assumptions . . . 94
5.3. Related work . . . 95
5.4. Containment of the iterates . . . 96
5.4.1. Containment of the last layer . . . 97
5.4.2. Containment of the previous layers . . . 98
5.5. Main results . . . 99
5.5.1. Convergence bounds . . . 99
5.5.2. Analysis of the bounds . . . 101
5.5.3. Optimal finite horizon Adam is Adagrad . . . 101
5.6. Proofs for $\beta_1 = 0$ (no momentum) . . . 102
5.6.1. Technical lemmas . . . 103
5.6.2. Proof of Adam and Adagrad without momentum . . . 106
5.7. Conclusion . . . 107

6. Future directions **109**

A. Supplementary results for Demucs **127**
A.1. Audio samples . . . 127
A.2. Results for all metrics with box plots . . . 127

B. Proofs of convergence of Adabatch **133**
B.1. Algorithms . . . 133
B.2. Expectation and variance of the AdaBatch update . . . 133
B.3. Proof of convergence of AdaBatch and mini-batch SGD . . . 137
B.3.1. Constant step size SGD with mini-batch . . . 137
B.3.2. Convergence of reconditioned SGD . . . 141
B.3.3. Convergence of AdaBatch . . . 142
B.3.4. Sparse linear prediction . . . 145

B.4. AdaBatch for SVRG . . . 146
B.4.1. Mini-batch SVRG . . . 146
B.4.2. AdaBatch SVRG . . . 149
B.4.3. Comparing the effect of regular mini-batch and AdaBatch for SVRG 150
B.5. Experimental results . . . 151
B.5.1. Experimental results for AdaBatch Wild . . . 151
B.5.2. Experimental results for SVRG . . . 155

C. Convergence of adaptive methods with heavy-ball momentum 157
C.1. Setup and notations . . . 157
C.2. Results . . . 158
C.3. Analysis of the results with momentum . . . 159
C.4. Technical lemmas . . . 160
C.5. Proof of Adam and Adagrad with momentum . . . 166

Abstract

Neural networks improved the state of the art for many classical signal processing tasks: audio source separation, speech enhancement, speech synthesis, synthesis of instruments etc. Existing models working directly in the waveform domain already exists, but their computational complexity does not allow to run them on CPU, and even training them on GPU can be a slow process. In this thesis, we aim at developing an architecture able to process waveforms, that is both fast and accurate. Our approach uses recurrent layers along with transposed convolutions, thus reducing the number of auto-regressive steps which would otherwise slow down the model. For pure generation, we avoid the use of a costly classification loss, relying instead on a differentiable loss on the log power spectrum. We obtain state-of-the-art results for the task of instrument modeling and music source separation. The training of such models requires the use of stochastic optimization techniques. We introduce a rule for aggregating gradients within a batch which speeds up convergence for sparse convex problems. Finally, we provide strong guarantees of convergence for the Adam and Adagrad algorithms when applied to non convex optimization.

Résumé

Les réseaux de neurones ont permis d'améliorer l'état de l'art pour de nombreuses tâches classiques de traitement du signal: séparation de sources audio, débruitage, synthèse de voix ou d'instruments. Des modèles fonctionnant dans le domaine temporel existent déjà, mais demandent trop de capacité de calcul pour pouvoir fonctionner sur CPU, et même leur entraînement sur GPU peut s'avérer coûteux. Dans cette thèse, nous nous proposons de développer une architecture capable de traiter l'audio dans le domaine temporel tout en étant rapide et précise. Notre approche utilise des couches récurantes suivies de convolutions transposées, ce qui permet de limiter le nombre de pas auto-régressifs qui pourraient sinon ralentir le modèle. Pour la génération pure, nous évitons l'usage d'un objectif de classification qui serait coûteux, lui préférant un objectif differentiable sur le spectre en log-puissance. Nos modèles atteignent l'état de l'art pour les tâches de modélisation d'instruments de musique et la séparation de sources musicales. L'entraînement de ces modèles nécessite l'utilisation de techniques d'optimisation stochastique. Nous présentons une nouvelle règle d'aggrégation des gradients au sein d'un mini-batch qui permet d'accélerer la convervence pour des problèmes convexes et sparses. Finalement, nous établissons des garanties de convergences pour deux algorithmes d'optimisation classiques, Adam et Adagrad, quand ils sont appliqués à un problème non convexe.

Remerciements

I would like to thank the rapporteur-e-s of this manuscript, Axel Roebel and Rachel Ward, for taking the time to read it and provide me with feedback and fixes. It was an honor to have you both on my jury and follow your steps on the exciting path of research in digital signal processing and optimization. Thank you Emmanuel Dupoux for also being on my jury, and giving me the chance to present you my work. Thank you again Axel for taking the time to answer my many questions when I first started working on audio. I learnt a lot and I hope I didn't leave too many inaccuracies in what follows.

Un grand merci à toi Francis, pour toutes les opportunités que tu m'as offertes. Grâce à toi j'ai pu tout autant découvrir la mythique Silicon Valley qu'explorer les contrées de l'optimisation théorique. Merci pour ton soutien en toutes circonstances, et de m'avoir aidé tout au long de mes recherches, même quand celles-ci ont divergées des tiennes. Thank you Hussein Mehanna for trusting me with exciting and impactful projects in California and London. I learnt and grew a lot trough this experience and I really enjoyed working with you. Merci à Florent Perronnin, Antoine Bordes et Patricia Le Carré de m'avoir permis d'accéder à une CIFRE malgré une situation complexe. Merci Antoine pour la salle de musique. Merci Léon pour toutes nos discussions, elles m'ont aidées à forger ma compréhension du domaine, de la psycho-acoustique à l'optimisation de chaque petit neurone. Merci Nicolas, tu m'as transmis ta rigueur et ton exigence. Tu t'es plongé avec intérêt dans chacun de mes projets et ton aide a été cruciale pour les mener chacun à terme.

Merci à Facebook et tout particulièrement toute l'équipe de FAIR Paris. Merci au PhD crew, pour toutes les folies, les baby-foots, et le soutien jusqu'au bout de la deadline. Merci tout particulièrement à Neil Zeghidour, Guillaume Lample, Pierre Stock, Timothée Lacroix et Louis Martin. Merci à l'INRIA et à l'équipe SIERRA de m'avoir accueilli. Merci à Thomas Kerdreux, Thomas Eboli, Alex Novak et Yana Hasson pour toutes nos discussions, couscous et autres façons d'oublier qu'il reste un article à écrire.

Merci à mes amis pour votre soutiens, les moments de joies et de folies. Merci à la team Nowhere et la team Pharmas. Merci à Sarah d'avoir enchanté la fin de ma thèse, de m'avoir soutenu face à l'incertitude et pour les très belles illustrations de ce manuscript. Merci à mes frères et soeurs, Aurélien, Audrey et Aude. Merci à mes parents qui ont tout fait pour que je puisse réaliser mon rêve d'enfant: me voilà devenu un vrai chercheur.

Merci à tous mes professeurs qui m'ont peu à peu révélé les mystères du monde physique, mathématique et spirituel qui nous entoure. Merci en particulier à Thierry Sageaux pour m'avoir guidé dès le lycée vers le chemin qui me mènerait à cette thèse.

“Des nains sur des épaules de géants”, Bernard de Chartres.

1. Introduction

In the present work, we wish to tackle the task of fast audio synthesis with deep learning models. By fast we mean a model that can produce audio in real time on a consumer level CPU. There is a growing divide between the computational power available in research clusters and high end devices, and the democratized market of computers, phones and on-board devices. While audio synthesis has seen solid improvements, for instance with WaveNet (see Section 1.2.1), the training of such models is computationally expansive and evaluation of those models is slow on consumer hardware. Furthermore, audio generation is likely to be only a part in a complex pipeline. By making it as lightweight as possible, we free up resources for the more complex tasks that would command or complete the synthesis process (conversation, planning, locomotion etc.). As the rate of the transistor miniaturization has almost halted, it would be risky to ignore this problem by waiting for more powerful hardware. Besides, designing new models to run on existing low end devices will help deploy them to the widest possible audience, while limiting planned obsolescence. Henderson et al. [2020] support accounting early on for the environmental impact of deep learning models in order to build sustainable technologies[1]

A key difficulty of audio generation is that a single perceptual experience can be caused by any signal sampled from a complex distribution with infinite support, while for images, there exist an almost bijective mapping between RGB pixel values and what we see. Playing twice the same note on an organ will likely result in two completely different realizations, which forbids the use of a regression loss directly on the waveform. The approach taken by state-of-the-art deep synthesis models is to model the entire distribution of waveforms. This requires performing conditional classification over at least 256 classes per time step (16,000 times per second of audio for speech, 44,100 for music), and at evaluation time, each step is predicted in an autoregressive manner. As this step alone cannot scale on CPU[2] it is interesting to investigate alternative architecture and

[1] While we did not track the exact impact of the present work, we estimate its CO2 emissions from powering the GPUs to be equivalent to 5 to 10 Paris/New York round trips.

[2] Taking a simple non trivial model, an LSTM with 2 layers, 32 inputs and 256 outputs, it takes 3.3 seconds to generate 1 second of audio at 44.1 kHz.

losses. We focus on music tasks for different reasons: because they cover a wide diversity of timbre and pitches, because they require a high sampling rate for commercial applications (which is a good way to assert the scalability of a model) and finally by mere personal preference.

The driving idea for this work is that it should be sufficient for the model to chose a canonical member of the waveform distribution, rather than modelling the entire distribution probability. In Chapter 2, we present SING, a universal synthesizer capable of modeling a thousand instruments with a subjective quality significantly higher than that of a WaveNet based auto-encoder for a fraction of its computational resources, at train or evaluation time. We introduce a simple architecture based on a LSTM followed by transposed convolutions that transform the internal representation from being high dimensional at a low sample rate, to the output waveform which is low dimensional at a high sampling rate. The change in sampling rate allows to have the LSTM auto-regressive layer scale, while still being able to model long range dependencies. In order to remedy the problem of doing regression over a distribution problem, we use a regression loss over the log-power spectrogram of the outputted waveform. This allows to partially collapse the entire distribution of waveforms to a single point in the quotient space. While promising, spectrogram losses still suffer from limitations: they do not penalize some irregularities in the phase and are harder to optimize due to their many local minima (inconsistent choices of phase over multiple segments). Besides, for sounds containing noise, the power spectrogram is itself stochastic, and we fall back on the same regression over a distribution problem.

In Chapter 3, we leave aside the problem of the loss to focus only on building an efficient architecture by studying the problem of audio source separation for music. Audio source separation is different from pure synthesis: as we have access to an input mixture, there is often a single possible output waveform. Music source separation benchmarks are sampled at 44.1 kHz, which reinforces the importance of having fast models with a limited number of auto-regression steps. So far, the state-of-the-art was achieved with models predicting a mask over the input spectrogram. Those models scale well but have specific limitation, and in particular make the attack of instruments sound dull and hollow, and cannot model well drums. We present Demucs, a U-Net based recurrent and convolutional architecture that improved on spectrogram methods while still being able to separate audio faster than real time on CPU. A convolutional encoder takes the input signal and transform it to a high dimensional representation with a low sampling rate. We then apply the same recipe as with SING: a recurrent layer with few time steps is applied and upsampled back to the output waveform with transposed

convolutions. The U-Net connections allows to directly pass fine grained details from the encoder to the decoder without making them go through the bottleneck.

In order to train any deep learning model, one need optimization methods that scales well with the number of parameters of the model and the size of the dataset, while being relatively robust to the choice of hyper-parameter. In Chapter 4, we introduce AdaBatch, a simple aggregation rules of gradients for mini-batch stochastic gradient descent that allows to speed up convergence for sparse problems by improving the conditioning of the problem. On standard sparse datasets, it matches the performance of Adagrad, but requires no extra memory storage. While not directly related to the task of audio modeling, this work allows to better understand stochastic optimization. The reconditioning performed is a limited second order method that only accounts for the sparsity of the data (not its scale or covariance), but is stable in the stochastic setting. In Chapter 5 we provide a bound on the convergence of two widespread adaptive algorithms, Adagrad and Adam, for non convex optimization. Adam is one of the most popular methods to train deep learning network so that understanding its behavior is important to derive practical guarantees. We provide a unified proof for Adagrad and Adam and in the absence of heavy-ball momentum, it is relatively short. There is currently limited theoretical results on why momentum helps. In fact, all the known bounds for SGD or adaptive methods with momentum in the non convex settings worsen as we add momentum. We provide a longer proof for this case, while improving the dependency in the momentum parameter, but still not showing a theoretical benefit.

In Chapter 6, we develop our ideas for future work on deep audio synthesis, both in terms of architecture and loss. We also present research direction for developing novel optimization methods that would behave like second order methods while being stable in the stochastic setting.

In the rest of this chapter, we cover the basic background on signal processing in Section 1.1, the historical context and our contribution to the task of pure audio synthesis in Section 1.2, before moving on to the problem of music source separation in Section 1.3. Finally, we present the important aspects of stochastic optimization along our contribution to the field in Section 1.4.

1.1. A short primer on sound

This section provides a short background on Digital Signal Processing (DSP) along with context on the complexity of the audio processing tasks we introduce in the next sections. We introduce in Section 1.1.1 and 1.1.2 the basics of DSP, and then present the vocoder

in Section 1.1.3, a now standard family of audio models able to perform tempo stretching and pitch shifting. We finish with a discussion on the limitations of vocoders due to their assumptions on the content of sound in Section 1.1.4.

1.1.1. From the physical world to the digital waveform

At the physical level, sound is the evolution of pressure in the air at a given point through time. For this evolution to be heard by the human hear, pressure needs to oscillate around its average value at a frequency between 20 Hz and 20 kHz. A microphone consist in a membrane attached to a device able to translate the mechanical movement induced by the change of pressure into an electrical signal. Such a device can consist in an induction coil moving in the magnetic field of a magnet: the movement of the coil in the magnetic field creates an electrical currant whose tension follows the oscillations of the sound wave reaching the microphone. On the other hand, a loud speaker is the opposite device: the same electrical currant sent through the coil will lead back to the same movement of the membrane, which will induce a change of pressure in the air.

Thus, we can restrict ourselves to the study of a function $y(t) \in \mathbb{R}$ with $t \in \mathbb{R}$, where t is the time (measured in seconds for instance) and y one of the quantity of interest (pressure, current location of the membrane, electrical tension). As all recording or playback devices have a limited range, $y(t)$ will be restricted to $[-1, 1]$ where -1 and 1 represent the most extreme displacements that the speaker or microphone can achieve. Still, computers can only deal with discrete data, thus it is required to discretized the values $y(t)$. This can be achieved using standard IEEE 754 float encoding over 32 bits, or linear coding over 16 bits integers as done in most WAV audio files. More compressed representations such as the μ-law logarithmic encoding over 8 bits have also been used for voice encoding in telecommunications. While the latter lead to a significant reduction in quality, it has been popularized again with the work of Oord et al. [2016] using deep learning to model waveforms, as the coding limited cardinality allows to use a classification loss. It is also necessary to discretize the time axis; the Nyquist-Shannon sampling theorem [Shannon, 1949] tells us that for any signal containing no frequency higher than $f_s/2$, it is possible to reconstruct it perfectly from equally spaced samples, with at least f_s of them per second. We have already noticed that the human hear cannot perceive frequencies beyond 20 kHz, so a sampling rate of 40 kHz should be sufficient for our needs. In practice, it is required to have the signal go through a low pass filter to prevent aliasing. It is not possible to build a perfect low pass filter with a finite impulse response [Smith, 2007], which means the filter cannot have a perfect cutoff at 20 kHz and one need a bit more capacity to allow for the attenuation band of

the filter, thus leading to the choice of the standard 44.1 kHz sampling rate, used by audio CD, most streaming services and digital audio libraries. While music makes use of the entire range of the human hear, it is not the case for all applications: telephony typically uses a sampling rate of 8 or 16 kHz, which is sufficient for conveying meaning but alters the texture of the sound in a way that is recognizable.

For a given sampling rate f_s in Hz depending on the application at hand, we define a monophonic waveform as $Y \in [-1, 1]^{Tf_s}$, with T the duration of the signal in seconds, chosen so that the number of samples $S = Tf_s$ is an integer. A stereophonic waveform will consist in two monophonic ones, i.e., $Y \in [-1, 1]^{2\times Tf_s}$. While audio with any number of channels could be studied, this setup will be sufficient for our needs.

1.1.2. The time-frequency representation of sound

A small chunk of an audio signal can be analysed thanks to the Discrete Fourier Transform [Smith III, 2011]. In order to prevent the apparition of artificial high frequencies, the chunk is made periodic by multiplying it with a window function that goes to zero on the edge, like the Hann window (a.k.a. the raised cosine window). By splitting the audio signal into overlapping chunks, and then taking their DFTs, one obtains a time-frequency representation called the Short-Time Fourier Transform (STFT). It is then possible to re-synthesize the audio from the STFT by computing the inverse DFT over each chunk and using the overlap add method [Allen, 1977]. This last operation is called the Inverse Short-Time Fourier Transform (ISTFT). The output of the STFT is complex valued, but while its modulus (the amplitude spectrogram) contains most of the meaningful information for discriminative purposes, it is not possible to synthesize realistic audio without its phase.

Note that one can see the STFT as a convolution of the input signal with a filter bank. Other filter banks have been developed, for instance the gammatone filter banks which is used as a model of the human auditory system and the cochlea [Wang and Brown, 2006]. The wavelet transform, as popularized among others by Mallat [1999] is another alternative. While the STFT has a fixed time resolution for all frequencies, the wavelet transform time resolution increases with frequency[3]. The downside of this varying time resolution is that one would need to sample different frequency coefficients at different rates. In order to remedy this problem, Andén and Mallat [2014] introduced a scattering transform, which pools high frequency features over time, but retains their variations through a recursive scattering up to a certain order. However, this transformation

[3]Due to the Heinsenberg-Gabor limit, the product of the time and frequency resolution is lower bounded so that any time-frequency representation has to decide on a specific trade-off.

is not explicitly invertible, which makes it better suited for classification or detection tasks rather than synthesis. When reconstruction is not a requirement, it is possible to further transform the STFT time-frequency representation, for instance by reducing the frequency resolution (while keeping the time resolution constant) through averaging, like for the mel-spectrogram [Mermelstein, 1976], thus reducing the number of coefficients while retaining roughly the same discrimination capacity as the human hear. This is particularly popular for speech recognition [Davis and Mermelstein, 1980]. Even if the mel-spectrogram is not mathematically invertible to a waveform, natural sounds have enough regularity that they can be convincingly inverted by deep learning models [Shen et al., 2018, Prenger et al., 2019]. Finally, rainbowgrams were introduced by Engel et al. [2017]. They conveniently represent the log-power magnitude for a specific frequency, but also the derivative of the phase, i.e. the instant frequency. The latter is color coded (explaining the name) while the amplitude is represented as the intensity of the color (black when the frequency is absent, full color when it is present). They allow to visually spot phase artifacts when modeling audio. Examples of mel-spectrograms and rainbowgrams are given on Figure 1.1.

1.1.3. The vocoder: an analysis/synthesis framework

A *vocoder* is a tool made of an analyser and a synthesizer, so that the output of the analyzer can be modified before being fed to the synthesizer. The name is the contraction of *voice coder* as vocoders were primarily developed for applications related to telephone communications. The quality of a vocoder is evaluated by two criteria: (i) the ease of interpretation and modification of the analyser output, (ii) the quality and absence of artifacts of the synthesizer output, especially after complex transformations of the analyser output. For instance, the identity function is a vocoder which would grade perfectly with respect to (ii) but very poorly with (i).

As first noticed by Flanagan and Golden [1966], the DFT provides a prime tool to build a vocoder. Their initial approach had too many artifacts, in particular because they did not use overlapping chunks. The refinement of the STFT/ISTFT by Allen [1977] allowed to develop a higher quality vocoder that could perform a change of tempo while leaving the pitch unchanged [Portnoff, 1981], changing the pitch while leaving the tempo unchanged [Seneff, 1982], or both at the same time [Moulines and Laroche, 1995]. Not all artifacts were removed though and large transformation deteriorated the texture of the sound, adding a characteristic called *phasiness*. Laroche and Dolson [1997] studied this issue and suggested some fixes. For instance, due to the windowing, a single stationary cosine would have non zero coeffecients over multiple frequency bins of the STFT, which

would be treated as multiple weaker cosines, each at one of the neighbouring frequencies. With the change of pitch/tempo, the phase of each cosine would drift away, leading to a "hollow" sound[4]. Part of the solution requires detecting peaks on the frequency axis and performing a phase locking of the nearby frequencies. Further trouble arises when one realises that the formula used to transform the phase from one tempo/pitch to another is only valid for a roughly stationary process. The attack of an instrument acts as a reset of the phase to a new random distribution. Using the transformation formula over an attack will dull it. Thus, peaks must also be detected on the time axis in order to detect transients and perform a phase reset at this point [Roebel, 2003].

1.1.4. The content of sound: periodicity and noise

The underlying assumption behind the STFT or phase locking vocoders is that an audio signal is composed of the sum of a few, mostly stationary, periodic cosine-like functions. This is especially true for the phase locked vocoder developed by Laroche and Dolson [1997]. This assumption is so powerful that it formed the basis for a large part of the early development of electronic music[5] and in particular synthesisers [De Wilde, 2016]. As summarized by Shepard [2013], the majority of synthesizers start from a simple periodic function (cosine, triangular, sawtooth) at the right fundamental frequency and shape it through the use of filters and envelopes. While it would be naive today to assume that this approach can recreate any sound, it is possible to achieve rather convincing imitations of a piano, violin, bass guitar or even basic vocalization with them [Engel et al., 2020].

Despite the success of electronic synthesizers and vocoders, it is not possible to reduce sound to a sparse combinations of cosines. Take the calming sound of the rain falling on the roof of a building. The time of impact of each drop of water can be approximated as independent from one another. The combinations of the many impacts will form a specific stochastic texture. Other examples of stochastic textures can be found in the sound of a snare drum, or in some phonemes in speech (say "sh" for instance). A simple but artificial example of stochastic texture is given by white noise, which is characterized by coefficients of equal modulus on average over all frequencies and with a random phase. The formula used for updating the phase when changing the tempo or pitch relies on the stationary sparse sum of cosines assumption and will not work when applied to white

[4] Note that this effect is sometime desired. The compositor Vangelis makes heavy use of two independent cosines with neighbouring frequencies for its chorus like quality.

[5] Another large part is based on sampling, i.e., recording a short audio waveform before filtering it and playing it back with a given pattern.

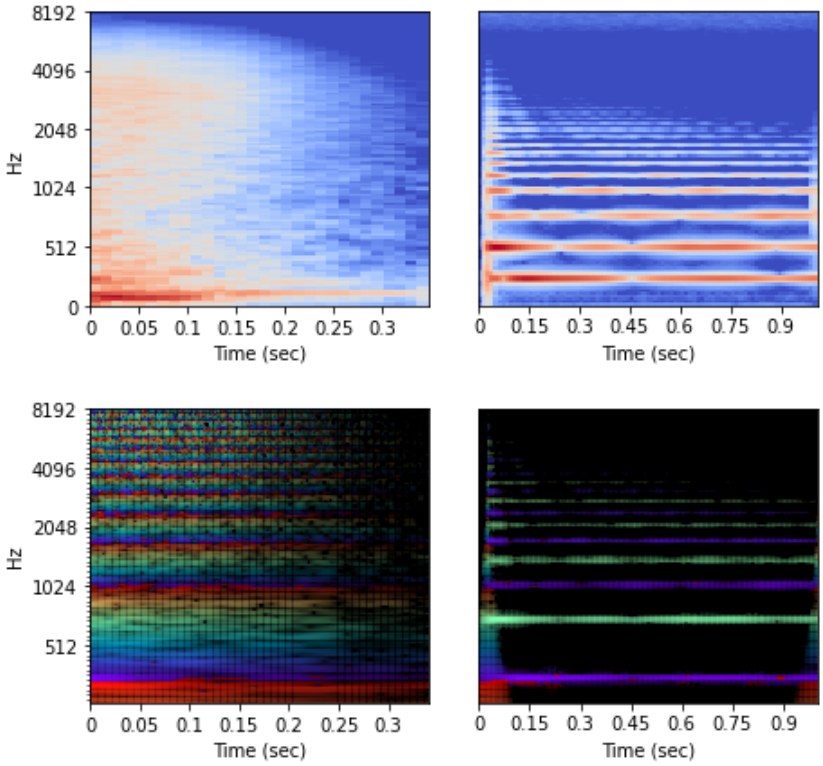


Figure 1.1.: Different time-frequency representations for a snare drum sample (left) and an FM synth note (right). The same audio is represented as a log-power mel-spectrogram (top), and a rainbowgram (bottom). Notice the harmonic structure of the FM synth, while the snare covers a wide range of frequencies.

noise. It will add a structure to the phase that was not present in the original audio, the same being true for more complex stochastic textures. Phase locking will detect spurious maxima along the frequency axis and bind together random variables that were independent. In order to account for noise-like components, more complex vocoders will try to separate the contribution of stationary cosines and of noise [Morise et al., 2016]. We show on Figure 1.1 different time-frequency representations introduced in Section 1.1.2 for two sounds of very different nature: a snare drum which contains high levels of stochasticity and a note played by a Digitone FM synthesizer, with a strong harmonic structure.

1.1.5. Conclusion

In this section, we have studied the nature of sound, from the underlying physical process to the sampled waveform stored on a computer. We have introduced the basic tools for the analysis, modification and synthesis of sound, namely the time-frequency representation given by the STFT and its inverse transform the ISTFT. We have seen how those tools allows to perform complex transformation of audio signals such as tempo stretching or change of pitch. We have also learnt that those approaches work best for a stationary sparse sum of cosines. For real world sounds, more complex rules are required in order to account for the frequency resolution of the STFT, the preservation transients, or the presence of stochastic textures. Those problems illustrate the difficulty of the modeling of audio processes, which can be somewhat simplified by the use of time-frequency representations, but only under certain assumptions.

1.2. Neural audio synthesis of music instruments

We will now turn to the task of pure audio synthesis. We first provide context on the development of audio synthesis models for Text-To-Speech (TTS), before presenting modern deep learning audio generation models and their application to music instruments modeling. Finally we give a short overview of our contribution, a universal instrument synthesizer, with improved perceived quality for a fraction of the training and evaluation time.

1.2.1. From concatenative models to end-to-end generation

Concatenative TTS Historical approaches to TTS leveraged a database of utterances along with their context, i.e. phonemes perfectly annotated with the preceding and

followings phonemes. In order to generate a specific sentence, one would first translate it to a sequence of phonemes and search in the database for the most appropriate samples [Hunt and Black, 1996] and concatenate them. While this lead to realistic and artifacts free audio with a database sufficiently large, the uterrances collection process must be repeated entirely to model new voices. As an alternative, statistical models were developed, based on two parts: (i) a model able to estimate the statistical characteristics of the speech, (ii) a synthesizer or vocoder able to turn those characteristics into a voice [Airaksinen, 2012]. This lead to more compact TTS systems and with a more natural flow through the sentence but more artifacts at the phoneme level due the limitation of the synthesizer or the vocoder [Ze et al., 2013]. Statistical characteristics will typically be the fundamental F0 frequency for voiced speech and the spectral envelope. Indeed, one of the simplest model of speech is the source-filter model [Fant, 1970]: an impulse train at a frequency F0 produced by the vocal chords goes through the vocal tract that acts as a linear filter. This filter can be approximated by a Finite Impulse Response (FIR) filter whose coefficients are optimized to reproduce the spectral enveloped on observed data. In practice, some phonemes require not an impulse train as input but white noise (voiced vs unvoiced speech) which is an extra characteristic the statistical model has to infer. Many more vocoders were developed, and a comparison was performed by Hu et al. [2013].

While the concatenative approach suffers from the prohibitive size of the dataset required to obtain a convincing synthesis (and a collection process that must be entirely repeated for every new voice), the statistical approach is quickly limited by the necessity of a deep understanding of the way speech is produced by humans as well as a computable model, as a model too simple would not sound natural. Deep learning models offer the capability to model complex phenomena in an accurate but black-box fashion [Goodfellow et al., 2016]. Regarding TTS, they were first put to use in order to improve the accuracy of the statistical estimation models [Ze et al., 2013, Zen et al., 2016]. The bottleneck for the quality of the generated speech then moved to the vocoder part.

Deep learning based audio synthesis A small revolution happened when Oord et al. [2016] found a way to combine the naturalness of concatenative model with the compactness of model-based speech synthesis. They developed WaveNet, the first deep learning network able to directly generate a waveform. When conditioned on the speech characteristics, like the one obtained from speech statistical models, WaveNet is able to output realistic audio for any of the 109 speakers it was trained on. However, WaveNet is fun-

damentally sequential: for every time step of the waveform to generate, it predicts a probability distribution given the previous samples and conditioning variables. It then sample from this distribution before feeding the chosen value back to the system to predict the next time step. Given that a waveform is typically sampled between 16 kHz (for research purpose) or up to 44.1 kHz (for commercial quality), this process cannot be achieved in real time, not even with batching and on accelerated hardware [Kalchbrenner et al., 2018]. During training, WaveNet uses the ground-truth as input which allows to parallelize the computation of the output for different time steps which mitigate the issues. Oord et al. [2017] introduced Parallel WaveNet which removes the need for autoregression, leading to faster inference but it requires distillation from a previously trained WaveNet model, which means even longer training times.

Other methods were developed like SampleRNN, [Mehri et al., 2016] made of multiple RNNs operating at different time scale and conditioned on the RNNs at coarser time scales. This multiscale approach allows to overcome the difficulty of handling very long sequences with RNNs, both in terms of gradient stability and speed, however, it did not significantly improve on WaveNet. Prenger et al. [2019] offered a faster alternative, with WaveFlow: a flow based approach to modeling audio at the waveform level. A flow based approach consist in transforming a known distribution, like a standard Gaussian, into a target distribution, like a dataset of audio samples. The transformation must be explicitely invertible and its Jacobian easy to compute which allows the likelihood of the generated audio to be derived from the likelihood of the matching Gaussian samples. The authors introduce specific operators that verifies those condition, as this rules out the use of standard deep learning components (convolutions etc.). The authors report a speed of generation on GPU of 520 kHz against 0.11 kHz for WaveNet, matching the speed of Parallel Wavenet at 500 kHz without the need for an extra distillation step.

One reason for the slowness of WaveNet is that it requires modeling an entire distribution for a given input. Indeed, we have seen in Section 1.1.4 that audio is usually made of a sparse combination of cosines with given frequencies mixed with stochastic textures. Both can have an infinite number of waveform representations, all perceptually equivalent (cosines can be shifted, stochastic textures can be drawn again from its distribution). To model those, WaveNet predicts the distribution of each time step, conditioned over the previous ones. This approach is fundamentally sequential: at train time, this can be alleviated by using batching and teacher forcing Bengio et al. [2015], but not at evaluation time. Parallel WaveNet [Oord et al., 2017] solve this issue by replacing the conditioning over previous time steps by an sampled gaussian noise. The model must be trained by distillation of a WaveNet model, leading to a long and com-

plex training procedure. The representation of the conditional probability is in itself another problem. WaveNet uses 256 possible outcomes using μ law encoding, but with a degradation of audio, while Parallel WaveNet uses a logistic mixture model. Given that each time step is typically encoded on two bytes, e.g. 65,536 possible outcomes, Kalchbrenner et al. [2018] presents WaveRNN, which splits the classification into two 256 classes predictions, one for the higher bits and the other for the lower bits. They achieves a generation speed of 96 kHz on GPU, the speedup being explain in part by the efficient representation of the conditional probability, as well as shallower architecture than WaveNet. Those approaches all require extensive work to scale to real time when running on CPU[6].

1.2.2. Using spectrograms for invariant audio synthesis

We now discuss an alternative to the exact distribution modeling approach used by WaveNet or WaveRNN, in order to still be able to output the predicted waveform directly through regression rather than classification using a differentiable loss over amplitude spectrograms.

Taking the modulus of the time-frequency representation given by the STFT introduced in Section 1.1.2 gives a representation that is mostly invariant to the different waveform realizations. This fact is leveraged for speech recognition by first transforming the input audio to an amplitude spectrogram or one of its variants such as the mel-spectrogram [Mermelstein, 1976]. Predicting an invariant representation is more convenient as it can be done by regression rather than distribution estimation. However what is missing is a way to transform this unique representation to a specific waveform. One possibility is to use the Griffin-Lim algorithm [Griffin and Lim, 1984], which consist in a fix point optimization over an initial guess of the phase in order to match the given spectrogram. However, the optimisation problem having many local minima, it is unlikely to perfectly match it. Shen et al. [2018] developed a TTS engine called Tacotron which uses a deep learning model in order to predict the amplitude spectrogram for a given input text, and use the Griffin-Lim algorithm in order to generate audio. For music note synthesis, Engel et al. [2017] also made use of the Griffin-Lim algorithm along with a convolutional autoencoder trained on amplitude spectrograms. In both case, significant artifacts were observed, with superior quality obtained when using a WaveNet vocoder. Shen et al. [2018] introduced Tacotron 2 which replaces Griffin-Lim with a

[6] A recent refinement of WaveRNN combined with heavy sparsification scale to real time on CPU, using all available cores: https://ai.facebook.com/blog/a-highly-efficient-real-time-text-to-speech-[...]-on-cpus/.

WaveNet conditioned over the input mel-spectrogram, with a significant gain in quality as measured with Mean Opinion Scores (MOS).

While it would be possible to try to predict the phase directly with a machine learning model, this would be almost equivalent to predicting the waveform directly. Instead Engel et al. [2019] suggest to predict the instant frequency, i.e., the derivative of the phase, which has the advantage of being unaffected by the shift of any of the sparse cosine components. The instant frequency is then integrated over time to obtain one possible choice of phase. While this approach has proven successful for the NSynth dataset, where each sample is made of a single well aligned note, with mostly harmonic instruments, it is more difficult to use on real life audio. In particular, the same problem occurring for the phase vocoders we introduced in Section 1.1.3 still apply: (i) the instant frequency for stochastic textures is still random (ii) during the attack of an harmonic instrument, a phase reset happens, leading to a random instant frequency.

Finally, it is possible to optimize the output waveform in order to match the characteristics of a given target amplitude spectrogram using gradient descent as pioneered in the work of Caracalla and Roebel [2017] for audio texture synthesis. In the following section, we reuse this approach as part of the end-to-end training of a deep learning model using the automatic differentiation of the log-amplitude spectrogram operator.

1.2.3. Contributions

In Chapter 2 we present SING, a Symbol-to-Instrument Neural Generator, achieving state-of-the-art synthesis quality at a fraction of the training and inference time of other neural vocoders. SING takes as input a one hot encoding of the instrument, pitch and velocity to generate and output the complete waveform sampled at 16 kHz. It is trained on the NSynth [Engel et al., 2017] dataset, made of 300,000 notes sampled at 16 kHz from over 1,000 instruments, covering each instrument entire chromatic range with 5 velocities, i.e., intensities, for each note. The modeling of music instruments can be done using similar technology as the one developed for speech synthesis, but this task also presents specific challenges: the diversity of timbre and wide frequency range covered.

SING uses an LSTM followed by convolutions and a transposed convolution that transform the high dimension but low sampling rate output of the LSTM into a low dimension but high sampling rate waveform. Remember that WaveNet could generate audio at 0.11 kHz, WaveGlow and Parallel WaveNet at about 500 kHz for speech. SING generates audio at 8.2 MHz on GPU, or 512 seconds of audio for every second of processing time. On CPU, SING works at 188 kHz, or 12 seconds of audio per second of processing time.

In order to train the SING universal note synthesizer, we chose to output directly a waveform, but compute its amplitude spectrogram in order to derive a loss in the spectrogram domain, which we call a *spectral loss*. This allows the model to make to most convenient choice of phase. The main difference with using Griffin-Lim is that the recovery of phase is done as part of an end-to-end loss. In contrast, using Griffin-Lim is a post processing step, whose limitations are not taken into account during training. For instance, as not all amplitude spectrogram match a real waveform signal, it is possible for the model to output an invalid spectrogram which will be projected onto the set of real spectrograms, a difference not accounted for in the training loss. Finally, Griffin-Lim is a relatively slow algorithm which requires many iterations to converge. Once trained, SING perform only a single feed-forward pass which directly output the waveform.

Our approach allowed us to obtain a music note synthesizer that outperformed both the Griffin-Lim and WaveNet approach on NSynth in terms of perceived quality and fidelity, for a fraction of the training and inference cost. However, some limitations remain: the use of the spectral loss still suffers from some issues common to any method optimizing over an amplitude spectrogram. As noted by Sturmel et al. [2011], if a signal x minimizes this loss, so does $-x$. It is then possible for the model to make two inconsistent choice of phases in two parts of the signals, one predicting x and the other $-x$. At the boundary betwen the two, the signal will cancel out. Because each part is pushing to opposite directions, their gradients cancel and we have a local optimum that cannot be escaped with regular gradient methods. This effect can be heard when listening to the generated samples for a flute, which is particularly stationary, where the inconsistent choices of phase can be heard as a slight beating.

1.2.4. Conclusion

We have studied pure audio synthesis, i.e., without any input waveform. The first methods for TTS used pre-recorded utterances that are then selected and concatenated into a final sentence. However, this methods will lead to unnatural transitions between phonemes and requires recording a lot of speech for any new voice to synthesize. Statistical models can estimate parameters for specifically designed vocoders. They don't need as much data and are quite compact once the model is trained. However, vocoders are hand designed and cannot cover the complexity of real life sound making. Thus, those models lack naturalness at the phoneme level. Deep learning based models such as WaveNet, WaveRNN or WaveGlow were developed as universal vocoders. Compared with hand-crafted systems, they can model audio in a black-box but complete manner. Because such models try to estimate the entire probability distribution of the output

waveform, they can be quite costly to run. Using an amplitude spectrogram based loss and a simple recurrent and convolutional architecture, we developed SING which allows for real time synthesis even on CPU. On the task of instrument modeling, SING achieved higher performance than a WaveNet based autoencoder for a fraction of the training and evaluation time. It also outperforms a Griffin-Lim based spectrogram autoencoder, which shows the interest of using a spectral loss over the waveform rather than doing spectrogram inversion as a post-processing step. While the approach is promising, the NSynth datasets consist mainly of harmonic instruments for which the amplitude spectrogram is mostly constant over realizations of the same sound. However, speech contains quite a bit of stochastic textures, for which even the amplitude spectrogram is stochastic and using our loss as is might result in a robotic voice. In Chapter 6, we explore potential directions to solve this and open the way to fast speech synthesis on CPU.

1.3. Music source separation in the waveform domain

We now turn to a task requiring both analysis and synthesis: music source separation. Source separation was first introduced in the context of speech, when Cherry [1953] defined the "cocktail party effect", namely the ability of the brain and auditory system to focus on a single conversation in a noisy environment, such as a cocktail party. This problem was later extended to the task of Audio Source Separation (ASS) for speech and music. Unlike TTS, or music note synthesis, a source separation model has access to an input signal, in particular it doesn't have to model the phase structure from scratch. This led to the development of different techniques compared with pure synthesis. In particular, it is possible to learn a mask over the input spectrogram, i.e., using the input phase when generating the output, which achieves higher quality than doing phase reconstruction with Griffin-Lim. This makes spectrogram methods more appealing than for TTS. Besides the input audio is sampled at 44.1 kHz for the standard music source separation datasets, which emphasizes the scaling issues of methods like WaveNet. Those differences explains that unlike for TTS and pure synthesis, waveform domain methods have been so far less successful for this task. The presence of the input waveform mostly reduces the distribution of outputs to a single point, so that we can focus more on the architecture, while using a simple regression loss over the waveforms.

In this section, we first cover blind source separation methods, before detailing supervised approaches, both in the spectrogram and waveform domain. We finish with presenting our contribution Demucs, the first waveform model to surpass spectrogram

models for the separation of music sources.

1.3.1. Blind source separation

Bregman [1990] formalized the ability of the brain to process auditory signals and identify relevent sources as Auditory Scene Analysis (ASA). As the understanding of the human psychoacoustic machinery improved, efforts were made to transpose such capabilities to computers, i.e., enabling Computational Auditory Scene Analysis. This initial goal was different from audio source separation, and mostly considered the extraction of semantic features from mixed sources, for instance multiple pitch estimation, speech transcription etc [Wang and Brown, 2006]. New techniques allowed to aim for a more ambitious goal: the recovery of waveform for each source.

The first of those techniques is Independent Component Analysis (ICA), introduced in the seminal paper of Jutten and Herault [1991]. ICA consist in finding components that are as independent as possible, for a specific non linear measure of independence. In order for the model to be identifiable, ICA requires at least as many mixtures (so that the collection of mixing weights are linearly independent) as the number of sources. For the cocktail party problem, one application would be when multiple microphones are located in the room, with as many of them as people in the room. ICA is composed of a rich family of approaches, with different independence criteria or optimization algorithms [Comon, 1994, Bell and Sejnowski, 1995, Hyvarinen, 1999]. Each criteria measures independence, and their diversity is explained by the fact that while independence is theoretically well defined, there is no single way to measure the level of independence of two distributions for which we only have access to samples.

While ICA showed promising results and allowed to recover individual waveforms, the constraint on the number of mixtures is only met in practice when dedicated microphone arrays are available. In most practical applications, such as phone calls or visio-conference without dedicated hardware, only monophonic data is available. For music applications, stereophonic audio is the standard, but a song contains more than two instruments. Even if stereo provides useful cues, humans are still able to perceive the different instruments with monophonic playback. Thus, the task of monoral source separation presents a significant interest. In such cases, the problem of source separation is necessarily ill-posed without further assumptions. One such assumption is that real life audio comes from a specific distribution rather than being a mixture of any possible waveforms. By modeling the distribution of waveforms from individual sources, one can extract a likely combination of sources. This approach was first developed by Roweis [2001]: for two sources, the amplitude spectrogram for each one is modeled by a sepa-

rate HMM. For each time step and frequency bin, whichever HMM explains the best the observed coefficient "wins" the bin. This provides a masking of frequency bins varying through time for each source, which can then be used to filter the mixture waveform and recover estimates of the sources.

Another approach consists in performing segmentation, like in computer vision, over amplitude spectrograms [Bach and Jordan, 2005]. In order to perform the clustering, cues are adapted from psychoacoustic observation. For instance, a continuous horizontal line (i.e., the same frequency present over a period of time) is likely to come from a single source. Besides, multiple horizontal lines starting at the same time are also likely to come from the same source, especially if they are harmonically related. Although appealing, one limitation of this approach is the need to hard code those cues rather than infer them automatically from the data.

Power spectrograms are non-negative, which allows for the usage of a specific factorization called Non-negative Matrix Factorization (NMF), discovered by Paatero [1997] and Lee and Seung [2001]. The idea is to represent the amplitude spectrogram P as a product WD where W and D are themselves non-negative. D represents a dictionary of spectral patterns and W is the weighting over time of the dictionary entries that come closest to P. Unlike regular matrix factorization, the non-negativity means that there cannot be any cancellation between two dictionary entries: like in real life, different sounds always add up[7]. [Smaragdis, 2004] noticed that this representation provides a natural separation of different music sources. However, complex sources cover more than one dictionary entry, showing the limits of monoral blind source separation. When supervised data is available, i.e., recordings of individual sources, it is possible to train a dictionary per source with NMF before computing the weights for a given amplitude spectrogram over the union of those dictionaries. Separation is then easily performed by remixing the contributions from a single dictionary. In order to obtain an estimate of the individual waveforms, either the separated amplitude spectrogram can be multiplied by the original signal phase and transformed with the ISTFT or a phase reconstruction algorithm like Griffin-Lime can be used, although with more artifacts [Virtanen, 2007].

1.3.2. Supervised source separation

We have seen that blind source separation is a difficult problem when there is less mixtures than the number of sources. It is possible to achieve some results with the use of psychoacoustic inspired cues but such approaches are quickly limited by our own under-

[7] Sounds can sometimes cancel out, but only for carefully designed and synchronized sources, such as with noise canceling headphones, or complex reflection patterns.

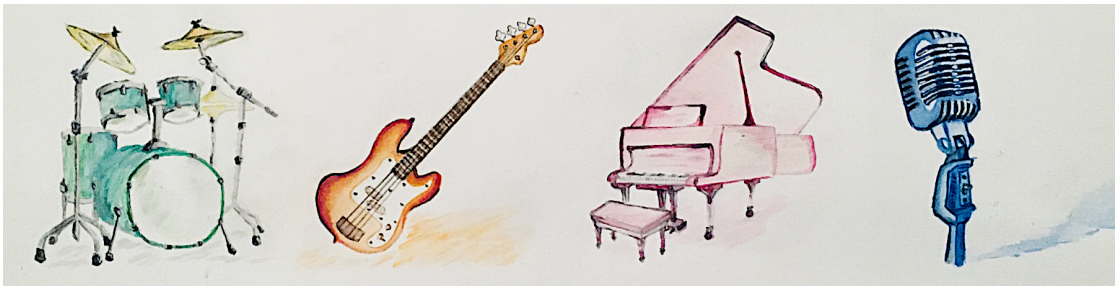


Figure 1.2.: The standard benchmark MusDB [Rafii et al., 2017] for music source separation consist in separating instruments grouped in four categories: drums, bass, other accompaniments and vocals.

standing of the human hearing and our ability to model it. Supervised source separation assumes the availability of individual sources which can then be mixed to obtain training examples along with their ground truth. This allows the definition of objective metrics, even if subjective evaluation remains the gold standard. Work on providing such metrics started by Schobben et al. [1999], and later Vincent et al. [2006] provided the framework used in Chapter 3, in particular the Signal to Distortion Ratio (SDR) which measures the overall quality of the reconstruction, and the Signal to Interference Ratio (SIR) which quantifies the amount of leakage from the other sources. Vincent et al. [2007] then organized the first audio separation evaluation campaigns, which continued until the last one as of writing [Stöter et al., 2018]. Those campaigns provides a photographic record of the evolution of the field: while in 2007, blind source separation with ICA or NMF was the main focus, it shifted to supervised source separation using deep learning in 2018. The last campaign was evaluated on the MusDB dataset, collected by Rafii et al. [2017], which consists in about 10 hours of commercial level music sampled at 44.1 kHz with the ground truth for four different sources presented on Figure 1.2: drums, bass, other accompaniments, and vocals.

Yilmaz and Rickard [2004] noted that applying a binary mask to an amplitude spectrogram and synthesizing the waveform from each source using the input phase yields very convincing results when the ideal mask is known. As a refinement, it is also possible to use a ratio mask with coefficient in $[0, 1]$, which can better model sources with conflicting frequencies [Reddy and Raj, 2007]. This allows to transform the separation problem into a classification problem, where each frequency bin is labeled with the source it belongs to. This is of course reminiscent of the seminal work of Roweis [2001] we presented earlier.

Deep Neural Network (DNN) are perfect candidates for estimating the ideal mask,

and they were quickly put to the task for music source separation [Uhlich et al., 2015] or speech separation [Huang et al., 2015]. Besides, most music tracks being stereophonic, Nugraha et al. [2016] improved prediction accuracy by estimating the spatial covariance matrices for the power spectrum and deriving a multichannel Wiener filter [Wiener, 1964]. Uhlich et al. [2017] later studied the impact of different data augmentation strategies, which is critical given the limited amount of data in standard datasets like MusDB. Standard data augmentation includes swapping of audio channels, shuffling of sources between tracks, and applying random gains on sources. More complex data augmentation can be used, such as randomly changing the pitch or tempo of the sources, using the vocoders introduced in Section 1.1.3 [Cohen-Hadria et al., 2019]. As part of the SiSec 2018 evaluation campaign, MMDenseNet [Takahashi and Mitsufuji, 2017] and MMDenseLSTM [Takahashi et al., 2018] proved to be the best spectrogram domain models. They consist in multiple multiscale dense net blocks [Huang et al., 2017] with a U-Net structure [Ronneberger et al., 2015], with different weights for different frequency bands. MMDenseLSTM further adds LSTMs operating at different time scales. While their authors did not provide the source code for their models, Stöter et al. [2019] released OpenUnmix, an open source spectrogram based model matching the performance of the MMDenseLSTM, but with a simpler architecture based on a bidirectional LSTM with extra fully connected layer and Wiener filtering.

In the waveform domain, Lluís et al. [2018] developed a WaveNet inspired architecture, but regression-based and non auto-regressive. Around the same time, Stoller et al. [2018] introduced Wave-U-Net, a U-Net based architecture, with significantly better accuracy than the WaveNet approach. Wave-U-Net was inspired by a U-Net architecture operating on amplitude spectrograms capable of separating the vocals from a song [Jansson et al., 2017]. However, Wave-U-Net performance did not match existing spectrogram methods, with an overall SDR over all sources of 3.2, against 5.3 for Open-Unmix. In the field of speech source separation however, a different approach allowed to surpass the Ideal Ratio Mask (IRM) oracle. The model called ConvTasnet [Luo and Mesgarani, 2019] consists in a simple convolutional encoder and decoder which is masked by a more complex model. The masking sub-model consists in a succession of convolutional blocks with skip connections and increasing dilations but a stride of 1, similar to the WaveNet model. The fact that a waveform domain model can achieve higher performance than an oracle on the spectrogram is a strong indicator of the intrinsic limitations of spectrogram methods. One limitation is that the phase contains important information on the texture of the sound. For instance, small variations of the frequency will be encoded in the phase, as well as information on the localization of transients in time. If two sources

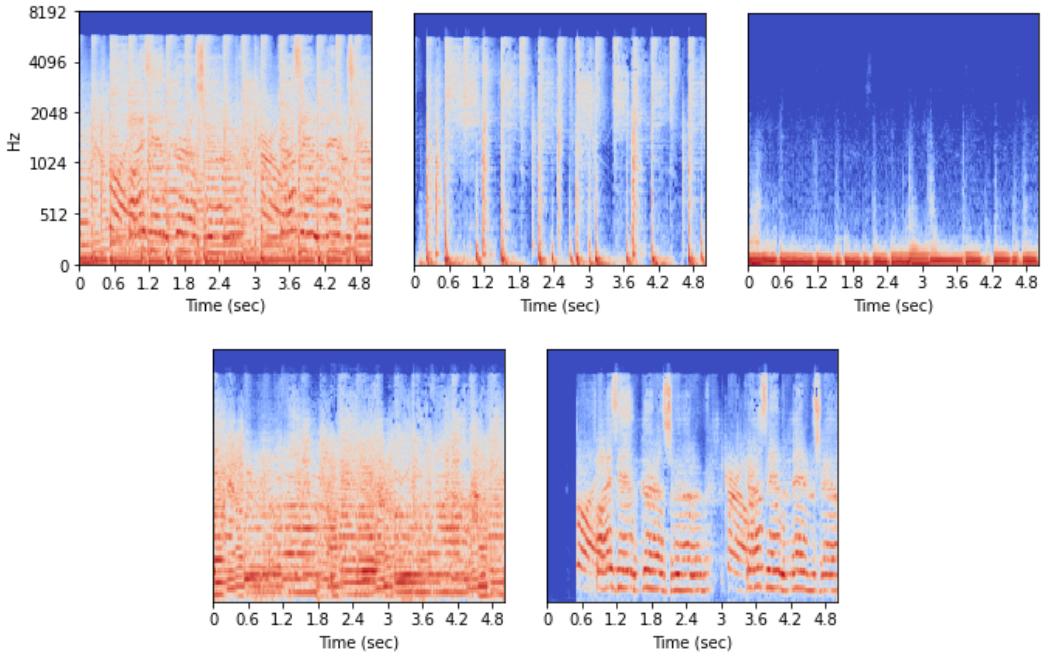


Figure 1.3.: Mel-spectrograms (limited to frequencies under 8 kHz) for an extract of the song "Androgyny" by Garbage and individual stems obtained with Demucs. From left to right: original, drums, bass, other, and vocals. Click on a spectrogram to listen to the audio.

overlap in frequencies, the sharing of the phase will lead to unavoidable bleeding of those characteristics and a deterioration of the transients. Finally, masking methods are subject to another limitation: the mixing of a loud source with a quiet one might lead to a loss of information. Imagine a strong crash cymbal overtaking the end of a light piano note. Masking method might not be able to recover the weak signal that was lost in the mix, while a model with sufficient capacity could "imagine" the end of the note, which we conjecture is a process likely to happen in the brain itself.

1.3.3. Contributions

In Chapter 3, we present Demucs, a time domain model for music source separation inspired by both the Wave-U-Net and SING architectures. The model consist in a convolutional encoder that gradually reduces the number of time steps while increasing the number of channels. Then a bi-directional LSTM provides contextual information

to improve the prediction of the model. Finally, a decoder based on transposed convolutions gradually increases the number of time steps while decreasing the number of channels. Each layer from the encoder is connected to the matching layer of the decoder through a U-Net Ronneberger et al. [2015] skip connection. The use of Gated Linear Units [Dauphin et al., 2017] in the encoder and decoder increased the capacity of the model by allowing masking-like operations (but not restricting it to masking only). In order to provide a strong baseline in the waveform domain, we adapted Conv-Tasnet [Luo and Mesgarani, 2019] to the task of music source separation.

When trained on MusDB, both Demucs and Conv-Tasnet surpass existing spectrogram or waveform domain methods in terms of the overall SDR. While Conv-Tasnet achieves a sligthly higher SDR than Demucs and less bleeding between sources, human evaluations show that it suffers from more artifact. We conjecture that this is due in part to its limited receptive field: because it uses a stride of 1 for all convolutions, it cannot handle long sequences without using excessive amounts of memory and is limited to a context of the order of a second at most, against 10 seconds for Demucs. When training Demucs, we did notice that a longer context improved performance. Finally, Conv-Tasnet predicts a mask over a learnt linear transformation of the input audio that has a kernel size of 20 samples only. On such a small window, it is possible for multiple sources to momentarily cancel out. The masking approach cannot produce any output audio if the input is zero on a given window of 20 samples, which could introduce extra artifacts.

Demucs particularly shines for the drum source: human listeners rated the naturalness of its samples at 3.8 out of 5, against 3.4 for Conv-Tasnet and 3.1 for OpenUnmix. This provides further evidence of the limitation of time-frequency representations for percussive sounds. On the other hand, the naturalness of vocals separated by a spectrogram method like OpenUnmix is rated at 3.0, against less than 2.6 for Demucs and Conv-Tasnet. Indeed, we noticed static noise in the vocal estimates of Demucs and Conv-Tasnet. This is likely because of the time domain regression loss which allows for small inconsistent mistakes to be made for each time step, while a spectrogram mask will make a consistent choice over an entire STFT window.

Finally, the Demucs model is quite big, 2GB[8], but can separate a song on CPU in about the same time as the duration of the song, while Conv-Tasnet requires about 3 second of processing time per second of audio. On Figure 1.3, we show the mel-spectrograms for the mixture and individual sources estimated by Demucs on the song "Androgyny" by Garbage. You can click on each spectrogram to listen to the corresponding audio.

[8]Early work using quantization can reduce this size to about 256MB.

1.3.4. Conclusion

We have studied the task of music source separation. While the field historically developed around blind separation, it shifted in the recent years to the task of supervised source separation. Deep learning models achieved great results, first in the spectrogram domain. While initial work on waveform domain models did not achieve the same level of performance, we showed that they can in fact outperform spectrogram domain methods, in particular for percussive sounds, which are not well represented with spectrograms because of their many transients and lack of harmonic structure. We introduced Demucs, a novel U-Net time domain architecture based on a convolutional encoder, a recurrent layer for modeling long term dependencies and a convolutional decoder. Demucs achieves a good balance between speed, separation quality and naturalness of the produced audio, which made it relatively popular with a mainstream audience [9], despite the unfriendliness of running research code.

We noticed that Demucs and other time domain methods produced artifacts especially when separating vocals. We conjecture that this is because they make inconsistent mistakes per time step, which introduces a kind of static noise. Combining the spectral loss presented in Section 1.2 with a time domain regression loss might help reducing those artifacts.

[9] See `https://www.youtube.com/watch?v=4_l31Vucrmo`.

1.4. Stochastic Optimization

So far, we have mostly studied how to "optimize" the architecture of deep learning models for audio tasks, in order to achieve high quality, fast evaluation on CPU and decent training time. We now turn to a different aspect: given an architecture, how to efficiently choose its weights to obtain the best possible metric.

1.4.1. Empirical Risk Minimization

The training of any machine learning model, and deep learning networks in particular, requires the fitting of its free parameters, or weights, in order to minimize the error on the task at hand [Bishop, 2006]. This error is defined as a loss function measuring the distance between the prediction of the model and the ground truth, as well as a distribution of inputs and labels. For instance, for the task of music source separation we introduced in Section 1.3, the true input distribution would be the set of all possible songs, the ground truth would be the separate waveforms for the drums, bass, vocals and the rest and the distance could be either the MSE, L1 or any other metric on the waveform. However, it is usually impossible to either characterize the entire set of inputs of interest (all possible songs) or obtain the ground truth for all. Therefore, we restrain ourselves to a finite dataset sampled from the true distribution, hoping that it is representative enough so that a model minimizing the training loss over it would perform well on unseen samples. This approach is called *empirical risk minimization* [Vapnik, 1992]. Formally, the empirical risk is given as a function $F(x)$ with $x \in \mathbb{R}^d$ the parameters of the machine learning model we consider. F can be decomposed as a sum of risks over individual training examples $(f_i)_{i\in[M]}$ (with $[M] = \{1, \dots, M\}$):

$$F(x) = \frac{1}{M} \sum_{i=1}^{M} f_i(x). \tag{1.1}$$

It is possible to characterize how well a model will generalize from its empirical training set to the true distribution [Vapnik, 2006]: larger models will typically have worse performance outside of their training set. The theory of generalization and empirical risk minimization allows to separate the problem of learning into two independent components: design of the objective function (dataset size, regularization, loss etc) and optimization of the chosen objective. This approach allows to consider optimization as a black box problem of its own although it hides the effect of a potential implicit regularization for certain optimization methods [Lee et al., 2016]. Note however, that while Vapnik's theory is well verified for linear models, its prediction contradicts observations for deep learning networks, where more parameters can lead to better generalization [Neyshabur et al.,

2017, Nakkiran et al., 2019]. In this work, we nonetheless leave aside the problem of generalization to focus mostly the problem of optimization, although it should be noted that when each training example is used exactly once, most of the results we present give a convergence bound on the true risk rather than its empirical approximation.

In order to minimize the empirical risk with respect to the model weights, a natural approach is to use a gradient method. Starting from a certain point $x_0 \in \mathbb{R}^d$, we iteratively compute the gradient of the risk with respect to the weights and update them making a small step in the direction of the gradient, i.e., for an iteration $n \in \mathbb{N}$ and step size $\gamma > 0$,

$$x_{n+1} = x_n - \gamma \nabla F(x). \tag{1.2}$$

This method, called Gradient Descent (GD), has been known for centuries [Cauchy, 1847], and in the case where the risk function is convex, its behavior is very well understood and easy to prove [Bubeck et al., 2015]. In fact, this method is even today the focus of research, as it allows for a simple setup to understand complex behaviors such as the implicit regularization of gradient descent [Li et al., 2017] or the avoidance of saddle points [Lee et al., 2016]. A first limitation of gradient methods, namely the computation of the gradient for complex architectures, was lifted with the back-propagation algorithm, formalized by Rumelhart et al. [1985], an application of the derivation chain rule.

The second limitation is the cost of computing the gradient when the number M of training samples is large. It also happens that for some applications, it is not possible to have access to the entire training set at once. Instead, samples are provided one by one and cannot (due to limited storage) or should not (legal requirements, change of distribution over time) be reused, for instance for ads click prediction [McMahan et al., 2013]. The setup of streaming examples is called *online* learning. For either large datasets or online learning, an alternative to gradient descent is needed.

1.4.2. Stochastic gradient descent

Robbins and Monro [1951] contributed a Stochastic Approximation (SA) algorithm able to find the root of a monotonic function from stochastic estimates. This algorithm was then extended to the multivariate case by [Blum, 1954]. For a given function, finding the root of its derivative is equivalent to finding a critical point. For convex functions, this critical point can only be a global minimizer of the function.

While SA was initially developed purely as a statistical problem, its application to machine learning, and the task of minimizing the empirical risk (or equivalently maximizing likelihood) were slowly uncovered [Ho, 1963, Kashyap and Blaydon, 1966]. Amari [1967]

showed the soundness of SA for the training of perceptrons in an online learning setting, even for non separable problems, although he makes no explicit mention of the Robbins-Monroe algorithm. SA provides a clear theoretical grounding to the update rules of the perceptron developed by [Rosenblatt, 1961], and initially inspired by the observation of biological neurons. In particular the SA framework, along with backpropagation, allows to extend those rules to any deep learning model architecture, loss function and training data distribution [Bottou, 1991]. When applied to machine learning, the SA algorithm is usually called Stochastic Gradient Descent (SGD). Let us define SGD formally when applied to minimizing the function F given by (1.1). Starting from a point $x_0 \in \mathbb{R}^d$ and given step sizes $\gamma_n > 0$, we sample a training example $j_{n+1} \in [M]$ for all iterations $n \in \mathbb{N}$ and update the parameters according to

$$x_{n+1} = x_n - \gamma +_n \nabla f_{j_{n+1}}(x_n). \tag{1.3}$$

The increase in storage capacity, and therefore the growth of the size of training datasets, gave a definite advantage to the stochastic approximation method the batch gradient descent [Bottou and Le Cun, 2005]. Indeed, GD iteration cost grow linearly with the size of the dataset while the iteration cost of SGD is independent of the number of training samples. When further accounting for the fact that the empirical risk is not the true metric we are interested in, but only an approximation, it is actually beneficial to be able to optimize over a larger dataset even with a worse rate [Bottou and Bousquet, 2008]. Beyond deep learning, Bordes et al. [2005] showed that SGD could also outperform classical optimizers for SVM, especially for large datasets.

Non asymptotic analysis Early work on SA/SGD mostly study the asymptotic behavior of stochastic optimization algorithms. The rate of convergence is known only for a large number of iterations, but how large this number needs to be to reach the asymptotic regime was unknown, i.e. one could not derive practical guarantees for the convergence of SGD. Nemirovski et al. [2009] were the firsts to provide non-asymptotic results when $\gamma_n \propto 1/n$ when the objective function is convex, with a better rate when it is strictly convex, assuming that the stochastic gradients are uniformly bounded and that F is smooth, i.e. its gradient ∇F is Liptchitz. The analysis was refined by Bach and Moulines [2011], assuming only the smoothness of the gradients and that the variance of the stochastic gradients is bounded by σ^2. Alternatively, Lacoste-Julien et al. [2012] obtained the same non asymptotic convergence rate for the strictly convex case with bounded gradients but with a much shorter proof. Needell et al. [2014] studied the case where γ is constant (which is often the case in practice) F is L-smooth and μ-strongly

convex. In that case, there exist a single minimizer $x_* = \arg\min_{\mathbb{R}^d} F$, and as long as $\gamma \leq \frac{L}{2}$, we have

$$\|x_n - x_*\|_2^2 \leq (1 - \gamma\mu)^n \|x_0 - x_*\|_2^2 + \frac{2\gamma\sigma^2}{\mu}. \tag{1.4}$$

While not strictly speaking convergent, using a large step size allows to quickly decay the first term on the left hand side. Once this term is negligible, one can reduce γ to achieve the desired sub-optimality. Non asymptotic bounds not only give strong practical guarantees but also make apparent the different regimes: the exponential term in (1.4) will initially be dominant when starting far from the optimum. Then, in the asymptotic regime, the iterates x_n will remain within a ball whose radius is proportional to the step size. Optimizing only for the asymptotic regime would lead us to take a step size as small as possible, but this would make the exponential term decay slower. Non asymptotic analysis is key to understanding the trade-offs between the fast forgetting of the initial condition and the asymptotic sub-optimality.

Using an optimal step size schedule, SGD converges to the optimum as $O(1/n)$ when F is μ-strongly convex and L-smooth, against $O((1 - \mu/L)^n)$ for GD. When F is only L-smooth, SGD converges as $O(1/\sqrt{n})$ while GD does so at a rate of $O(1/n)$ (see [Bottou et al., 2018] for SGD and [Bubeck et al., 2015] for GD). The rates of SGD are optimal when one only has access to stochastic gradients and without further assumptions on F than its smoothness [Agarwal et al., 2009].

Finite sum and variance reduction. The structure of F given by the empirical risk minimization (1.1) is specific though: it is a finite sum of functions. This structure can be leveraged in order to obtain much faster convergence in the convex case, matching exactly those of GD but with the a computational cost just a few times that of SGD. The first method that achieved those rates is SDCA [Shalev-Shwartz and Zhang, 2013], which consist in a stochastic coordinate ascent in the dual: at each iteration, a training example is sampled and the associated dual parameter updated. However, SDCA requires the optimization problem to have a dual formulation. In practice, this can be achieved for linear models with ℓ_2 regularization. [Schmidt et al., 2017] introduced SAG, the first primal gradient method that achieves the same rates as GD without requiring the existence of a dual problem or ℓ_2 regularization. In particular, the algorithm converges optimally whether the problem is strongly convex or not without any change. Other methods were then discovered, such as SVRG [Johnson and Zhang, 2013], which enjoys the same rate of convergence as SAG but with a significantly shorter proof but requires the tuning of an extra hyper parameter. SAGA [Defazio et al., 2014] allows to combine the simplicity of SVRG with the absence of extra hyper parameter of SAG.

SAG, SVRG and SAGA are expressed in the primal formulation of the problem and provide an intuitive explanation: for each training example, they store an old gradient computed sometime in the past. When the example is sampled again, the old gradient is replaced by the newly computed one. When the example is not sampled, the optimizer uses the old gradient stored. Thus, at every iteration, all the examples contribute to the update, either through a fresh or stored gradient. This can be implemented efficiently for linear models as the stochastic gradient is colinear to the input vector and a single scalar has to be stored per example. As the model gets closer to the optimum, one can show that the average of the stored gradients gets closer to the true gradient, thus speeding up convergence in a virtuous cycle. The use of the old gradients to estimate the true total gradient $\nabla F(x_n)$ allows to reduce the variance of the stochastic gradient estimates, explaining why this approach is usually called variance reduction. While extremely useful for solving convex problems that can be expressed as a finite sums, those methods do not work well for non convex objective for two reasons. First the storage requirements for non linear models explodes, as one must store $d \times M$ values instead of M in the linear case (or in the case of SVRG, store a second version of the model and perform two evaluations of the gradient per iteration rather than one). Second, it has been observed by Defazio and Bottou [2019] that the old gradient information becomes "outdated" too quickly for deep neural network and do not actually improve convergence.

1.4.3. Adaptive optimization methods

A limitation of (S)GD is its dependency to the model parametrization. Instead of optimizing $F(x)$, we could have made a different choice and instead chose to optimize $G(\tilde{x})$ with $\tilde{x} = \lambda^{-1}x$ with $\lambda \in \mathbb{R}$ and G defined by $G(\tilde{x}) = F(\lambda\tilde{x})$. When using (S)GD, optimizing G with respect to $\tilde{x}$ is equivalent to the original optimization but with a step size of λ^2 times the original one. The squared dependency in λ tends to amplify bad choices of scaling and requires choosing carefully both the step size and the parametrization of the model. Using second order methods such as Newton's method will alleviate the issue, as they are invariant to any affine change of coordinates. However, second order methods do not work well in the stochastic settings, as inaccuracies in the estimation of the Hessian of the objective will lead to updates in arbitrarily bad directions [Bottou et al., 2018].

Adaptive methods, such as Adagrad [Duchi et al., 2011], partially solves this problem. Adagrad uses a per coordinate step size which depends on the scale of past gradients. If we take a coordinate $j \in [d]$ and iteration $n \in \mathbb{N}$, we note $x_{n,i}$ the i-th coordinate of x_n and ∇_i the derivative with respect to the i-th coordinate. Then, given a starting point

$x_0 \in \mathbb{R}^d$ and a global step size $\alpha > 0$, the Adagrad update is given by

$$x_{n+1,i} = x_{n,i} - \alpha \frac{\nabla_i f_{j_{n+1}}(x_n)}{\epsilon + \sqrt{\sum_{k=0}^{n} \left(\nabla_i f_{j_{k+1}}(x_k)\right)^2}}, \tag{1.5}$$

with ϵ typically of the order of 10^{-10}, introduced for numerical stability. Adagrad updates are invariant to a global rescaling of the objective function, and doing a scalar rescaling of the parameters λ as introduced in the previous paragraph is equivalent to changing the step size by a factor λ. In fact, because they provide a step size per coordinate, the property is still verified if λ is a diagonal rescaling. While adaptive methods are not completely invariant to an affine rescaling, the impact of a diagonal re-parametrization is not amplified like with SGD. Finally, the rate of convergence of Adagrad matches that of SGD for non convex (see Chapter 5) and non strictly convex problems, but does not require knowing the smoothness L of the objective function.

While Adagrad proved effective for sparse optimization [Duchi et al., 2013], or tensor factorization [Lacroix et al., 2018], experiments showed that it under-performed when applied to deep learning [Wilson et al., 2017]. With RMSProp, Tieleman and Hinton [2012] proposed an exponential moving average instead of a cumulative sum of the past squared gradients. Thus, if we assume that the scale of the stochastic gradients is constant over some period of time, the effective learning rate would stabilize at a constant value, while for Adagrad, it would decay as $1/\sqrt{n}$. Kingma and Ba [2014] built upon RMSProp and developed Adam, currently one of the most popular adaptive algorithms in deep learning, by adding a corrective term to the step sizes at the beginning of training, together with heavy-ball style momentum Polyak [1964]. In their original paper, the authors showed with a decaying overall step size α, their algorithm converges to a minimizer for convex objectives. However their proof is incorrect, as pointed out by Reddi et al. [2019] who gave examples of convex problems where Adam does not converge to an optimal solution. They proposed AMSGrad as a convergent variant of Adam, which consisted in retaining the maximum value of the exponential moving average. They prove the convergence of AMSGrad with a decaying α in the convex settings. Other variants of Adam were developed, in particular AdamW [Loshchilov and Hutter, 2019] that moves the contribution of the L2 regularization outside of the adaptive update or NAdam which uses so called Nesterov style momentum instead of heavy-ball Dozat [2016].

1.4.4. Non convex optimization

Convergence of SGD Notice that the early works on SA did not make any convexity assumption. However, as explained in Section 1.4.2, this prevented the study of non

asymptotic bounds, which gives practical guarantees. When F is convex and lower bounded, one can take F_* its infimum. In the convex setting, any critical point is a local minimum and all local minima share the same value F_*. Therefore, it makes sense to study convergence in the sense of $F(x_n) - F_*$ going to zero. For strictly convex objective, the infimum is reached in a single point x_* and one can also bound $\|x_n - x_*\|$. With the rising popularity of deep learning, results for the non convex setting regained interest. However, it is not true that all local mimima share the same value, or that any critical point is a local minima. Thus, one can only hope to show that SGD converges, in some sense, to a critical point (remember that the original SA algorithm finds a root of a function and that SGD is SA applied to the gradient of F). Besides, we cannot show that the last iterate is a critical point, as the algorithm could be on a plateau before going downhill at any time. Instead, Ghadimi and Lan [2013] show that when taking a random iterate index τ, with τ following the uniform distribution over $\{0, \dots, N-1\}$, $\mathbb{E}\left[\|\nabla F(x_\tau)\|_2^2\right]$ converges to zero as $O(1/\sqrt{N})$ when F is L-smooth and as long as $\gamma \leq \frac{1}{L}$. This rate cannot be improved without further assumptions on F, as shown by Drori and Shamir [2019].

Convergence of adaptive methods Li and Orabona [2019] first tackled the problem of the convergence of Adagrad to a critical point in the non convex settings. However, the bound they derived is only valid for values of $\alpha \leq \epsilon/L$, which limits its practical use. Ward et al. [2019] provided a novel proof framework for adaptive methods and applied it to the scalar version of Adagrad, i.e. when a single learning is used for all coordinates and the total squared norm of the gradients is aggregated. They show a rate of convergence of $O(\ln(N)/\sqrt{N})$, matching that of SGD for any value of α, but unlike SGD, the proof requires for F to have bounded gradients. Zou et al. [2019b] extended the proof to the diagonal learning rate and added support for either heavy-ball or Nesterov style momentum in their proof. Chen et al. [2019] also present bounds for Adagrad and Adam, without convergence guarantees for Adam.

We mentioned in Section 1.4.3 that Adam does not converge even with a decaying overall step size α [Reddi et al., 2019]. However, Zou et al. [2019a] showed that $\mathbb{E}\left[\|\nabla F(x_\tau)\|_2^2\right]$ will be asymptotically bounded, with the bound depending on the characteristic of the optimization problem and the hyper-parameters. In particular, for a known total number of iterations N, one can chose the hyper-parameters of Adam to match the convergence rate of Adagrad.

1.4.5. Contributions

Contributions to sparse optimization One instance where Adagrad is particularly efficient is sparse optimization, i.e. when the stochastic gradient only has a few non zero entries. It can be further combined with the Hogwild! parallel scheme Recht et al. [2011], to provide a parallel optimizer that automatically adapts to the sparsity of the data [Duchi et al., 2013]. In Chapter 4, we explain this behavior as we show that for linear models, e.g. least-mean-square or logistic regression, and under mild assumptions, one should use larger step sizes for rare features. With SGD, this is not possible if not all features appear with the same frequency. Indeed, taking a large step size for rare features would make the frequent features diverge. If a feature appear with probability p, Adagrad will automatically use a per coordinate step size proportional to $1/\sqrt{p}$. We show that it is in fact possible to match the performance of the Adagrad optimizer with a memory-less optimizer called Adabatch based on a simple idea: within one batch and for each coordinate, instead of dividing the sum of gradients by the size of the batch, divide it by the number of time the feature was active in the batch. In fact, one can see this as an approximate second order methods, as we model the part of the Hessian that depends on the frequency of each feature. What is interesting with Adabatch is that unlike Newton methods, the pre-conditionner that is derived from the number of time each feature is active is always stable and convergent, despite its stochasticity.

Contributions to non convex adaptive optimization In Chapter 5 we provide a unified proof covering both Adam and Adagrad in the non convex setting, assuming the boundness of the gradients and the smoothness of the objective function. We discuss the practicality of those assumptions. Indeed, boundness of the gradients is not verified by simple model such as linear least-mean-square nor a deep learning network. In fact, a deep neural network is not uniformly smooth, as the local smoothness constant with respect to one layer depends on the norm of the other layers. We show that for a simple multi layer perceptron with sigmoid activations and L2 regularization, one can show that the iterates of Adam/Adagrad will remain bounded, which allows to apply our convergence results.

Our unified analysis further strengthen the connection between Adam and Adagrad, with the simple idea that Adam is to Adagrad like constant step size SGD is to decaying step size SGD. While not strictly speaking convergent for any hyper-parameter, Adam can be made to converge with the same rate as Adagrad knowing only the time horizon N. Besides, our bounds show that Adam moves away faster from its starting point x_0 than Adagrad, explaining its practical success.

Finally, our proof technique improves the tightness of the bound with respect to the heavy ball parameter. Remember that heavy-ball momentum consists in keeping an exponential moving average of the past gradients

$$m_{n+1} = (1 - \beta_1)m_n + \beta_1 f_{j_{n+1}}(x_n), \tag{1.6}$$

and using it to update x instead of the last stochastic gradient. The case without momentum corresponds to $\beta_1 = 0$. There is currently no theoretical proof that momentum helps, be it for adaptive methods or regular SGD [Yang et al., 2016]. The bounds always get worse as β_1 increases: for SGD and Adagrad, the best dependency is $O((1-\beta_1)^{-3})$ [Zou et al., 2019b, Yang et al., 2016] and for Adam it is $O(1-\beta_1)^{-5})$ [Zou et al., 2019a]. We provide a tighter proof for Adam and Adagrad with a dependency in $O((1 - \beta_1)^{-1})$. Our proof technique can also be used with SGD to achieve the same dependency.

1.4.6. Conclusion

Stochastic optimization is the steam engine that makes deep learning models progress from their random initialization to a high level of accuracy. The use of stochastic estimates of the gradients allows to scale training to large training sets, which is more important than achieving a faster rate of convergence on a smaller dataset. The number of parameters of deep learning models rules out the use of exact second order methods. Even approximate second order methods cannot be used because they are not stable when using stochastic gradients. This explains the popularity of SGD for training deep learning models. Another important aspect is the robustness to the choice of the hyper-parameters, the step size in particular. Adaptive methods like Adagrad or Adam achieves the same rate of convergence as SGD without having to know the smoothness constant of the objective function. They are also less sensitive to a diagonal re-parametrization of the problem, thus obtaining part of the benefit from second order methods at a fraction of their computational cost and still working well in a stochastic settings.

We presented two novel contributions. First, a simple gradient aggregation rule for sparse optimization called Adabatch, which achieves faster convergence when using large batch sizes by performing a reconditioning based on the sparsity of each feature within a batch. Unlike generic second order method, this reconditioning is stable in the stochastic settings and allows to match the performance of adaptive methods such as Adagrad. Second, we provide a unified proof for Adam and Adagrad for non convex optimization, with a tightened dependency in the momentum parameter.

1.5. Summary of contributions

- In Chapter 2, we introduce SING, a universal instrument synthesizer capable of producing notes from 1,000 instruments using an architecture based on a recurrent layer followed by transposed convolutions. The model outputs directly a waveform and is optimized with a loss over the log-amplitude spectrograms. SING is able to generate 16 kHz audio at 8.2 MHz on GPU, and 188 kHz on CPU, i.e. 10 times faster than real time, while improving the state-of-the-art as measured by subjective scores.

- In Chapter 3, we turn to the task of music source separation with Demucs, the first waveform domain model to match the performance of spectrogram based models, while surpassing it for drums and bass lines. Demucs consist in a U-Net auto-encoder, using convolutions followed by a recurrent layer, and transposed convolutions to output the final audio. It is capable of separating a stereophonic song sampled at 44.1 kHz in about the same time as the duration of the track on a laptop CPU.

- We then turn to AdaBatch in Chapter 4, a simple aggregation rule for gradients in a mini-batch that improves the condition number of a sparse convex optimization problem. It allows to increase the batch size without slowing down convergence. Larger batch sizes are important for efficient parallel computing and our method allows to match state-of-the-art performance for parallel sparse optimization without requiring extra memory storage.

- Finally, we provide in Chapter 5 convergence bounds for the adaptive Adagrad and Adam optimizers when applied to non convex problems, with a unified proof technique. We show that for a given number of iterations, the hyper-parameters of Adam can be chosen to match the convergence of Adagrad. Our proof technique improves the tightness of the bound when using heavy-ball momentum, although the improvement is not sufficient to show a theoretical gain with momentum.

2. SING: deep neural generator for music notes

Abstract

Recent progress in deep learning for audio synthesis opens the way to models that directly produce the waveform, shifting away from the traditional paradigm of relying on vocoders or MIDI synthesizers for speech or music generation. Despite their successes, current state-of-the-art neural audio synthesizers such as WaveNet and SampleRNN Oord et al. [2016], Mehri et al. [2016] suffer from prohibitive training and inference times because they are based on autoregressive models that generate audio samples one at a time at a rate of 16kHz. In this work, we study the more computationally efficient alternative of generating the waveform frame-by-frame with large strides. We present SING, a lightweight neural audio synthesizer for the original task of generating musical notes given desired instrument, pitch and velocity. Our model is trained end-to-end to generate notes from nearly 1000 instruments with a single decoder, thanks to a new loss function that minimizes the distances between the log spectrograms of the generated and target waveforms. On the generalization task of synthesizing notes for pairs of pitch and instrument not seen during training, SING produces audio with significantly improved perceptual quality compared to a state-of-the-art autoencoder based on WaveNet Engel et al. [2017] as measured by a Mean Opinion Score (MOS), and is about 32 times faster for training and $2,500$ times faster for inference.

2.1. Introduction

The recent progress in deep learning for sequence generation has led to the emergence of audio synthesis systems that directly generate the waveform, reaching state-of-the-art perceptual quality in speech synthesis, and promising results for music generation. This represents a shift of paradigm with respect to approaches that generate sequences of parameters to vocoders in text-to-speech systems Sotelo et al. [2017], Taigman et al. [2017], Ping et al. [2018], or MIDI partition in music generation Hadjeres et al. [2016], Ebcioğlu

[1988], Herremans and Chew [2016]. A commonality between the state-of-the-art neural audio synthesis models is the use of discretized sample values, so that an audio sample is predicted as a categorical distribution conditionned over past samples, trained with a cross-entropy loss Oord et al. [2016], Mehri et al. [2016], Oord et al. [2017], Kalchbrenner et al. [2018]. Another significant commonality is the use of autoregressive models that generate samples one-by-one, which leads to prohibitive training and inference times Oord et al. [2016], Mehri et al. [2016], or requires specialized implementations and low-level code optimizations to run in real time Kalchbrenner et al. [2018]. An exception is parallel WaveNet Oord et al. [2017] which generates a sequence with a fully convolutional network for faster inference. However, the parallel approach is trained to reproduce the output of a standard WaveNet, which means that faster inference comes at the cost of increased training time.

In this chapter, we study an alternative to both the modeling of audio samples as a categorical distribution and the autoregressive approach. We propose to generate the waveform for entire audio frames of 1024 samples at a time with a large stride, and model audio samples as continuous values. We develop and evaluate this method on the challenging task of generating musical notes based on the desired instrument, pitch, and velocity, using the large-scale NSynth dataset Engel et al. [2017]. We obtain a lightweight synthesizer of musical notes composed of a 3-layer RNN with LSTM cells Hochreiter and Schmidhuber [1997] that produces embeddings of audio frames given the desired instrument, pitch, velocity[1] and time index. These embeddings are decoded by a single four-layer convolutional network to generate notes from nearly 1000 instruments, 65 pitches per instrument on average and 5 velocities.

The successful end-to-end training of the synthesizer relies on two ingredients:

- A new loss function which we call the *spectral loss*, which computes the 1-norm between the log power spectrograms of the waveform generated by the model and the target waveform, where the power spectrograms are obtained by the short-time Fourier transform (STFT).

  Log power spectrograms are interesting both because they are related to human perception Goldstein [1967], but more importantly because the entire loss is invariant to the original phase of the signal, which can be arbitrary without audible differences.

- Initialization with a pre-trained autoencoder: a purely convolutional autoencoder

[1]Quoting Engel et al. [2017]: "MIDI velocity is similar to volume control and they have a direct relationship. For physical intuition, higher velocity corresponds to pressing a piano key harder."

architecture on raw waveforms is first trained with the spectral loss. The LSTM is then initialized to reproduce the embeddings given by the encoder, using mean squared error. After initialization, the LSTM and the decoder are fine-tuned together, backpropagating through the spectral loss.

We evaluate our synthesizer on a new task of pitch completion: generating notes for pitches not seen during training. We perform perceptual experiments with human evaluators to aggregate a Mean Opinion Score (MOS) that characterizes the naturalness and appeal of the generated sounds. We also perform ABX tests to measure the relative similarity of the synthesizer's ability to effectively produce a new pitch for a given instrument, see Section 2.5.3. We use a state-of-the-art autoencoder of musical notes based on WaveNet Engel et al. [2017] as a baseline neural audio synthesis system. Our synthesizer achieves higher perceptual quality than Wavenet-based autoencoder in terms of MOS and similarity to the ground-truth while being about 32 times faster during training and $2,500$ times for generation.

2.2. Related Work

A large body of work in machine learning for audio synthesis focuses on generating parameters for vocoders in speech processing Sotelo et al. [2017], Taigman et al. [2017], Ping et al. [2018] or musical instrument synthesizers in automatic music composition Hadjeres et al. [2016], Ebcioğlu [1988], Herremans and Chew [2016]. Our goal is to learn the synthesizers for musical instruments, so we focus here on methods that generate sound without calling such synthesizers.

A first type of approaches model power spectrograms given by the STFT Engel et al. [2017], Haque et al. [2018], Wang et al. [2017], and generate the waveform through a post-processing that is not part of the training using a phase reconstruction algorithm such as the Griffin-Lim algorithm Griffin and Lim [1984]. The advantage is to focus on a distance between high-level representations that is more relevant perceptually than a regression on the waveform. However, using Griffin-Lim means that the training is not end to end. Indeed the predicted spectrograms may not come from a real signal. In that case, Griffin-Lim performs an orthogonal projection onto the set of valid spectrograms that is not accounted for during training. Notice that our approach with the spectral loss is different: our models directly predict waveforms rather than spectrograms and the spectral loss computes log power spectrograms of these predicted waveforms.

The current state-of-the-art in neural audio synthesis is to generate directly the waveform Oord et al. [2016], Mehri et al. [2016], Ping et al. [2018]. Individual audio samples

are modeled with a categorical distribution trained with a multiclass cross-entropy loss. Quantization of the 16 bit audio is performed (either linear Mehri et al. [2016] or with a μ-law companding Oord et al. [2016]) to map to a few hundred bins to improve scalability. The generation is still extremely costly; distillation Hinton et al. [2015] to a faster model has been proposed to reduce inference time at the expense of an even larger training time Oord et al. [2017]. The recent proposal of Kalchbrenner et al. [2018] partly solves the issue with a small loss in accuracy, but it requires heavy low-level code optimization. In contrast, our approach trains and generate waveforms comparably fast with a PyTorch[2] implementation. Our approach is different since we model the waveform as a continuous signal and use the spectral loss between generated and target waveforms and model audio frames of 1024 samples, rather than performing classification on individual samples. The spectral loss we introduce is also different from the power loss regularization of Oord et al. [2017], even though both are based on the STFT of the generated and target waveforms. In Oord et al. [2017], the primary loss is the classification of individual samples, and their power loss is used to equalize the average amplitude of frequencies over time. Thus the power loss cannot be used alone to learn to reconstruct the waveform.

Works on neural audio synthesis conditioned on symbolic inputs were developed mostly for text-to-speech synthesis Oord et al. [2016], Mehri et al. [2016], Wang et al. [2017]. Experiments on generation of musical tracks based on desired properties were described in Oord et al. [2016], but no systematic evaluation has been published. The model of Engel et al. [2017], which we use as baseline in our experiments on perceptual quality, is an autoencoder of musical notes based on WaveNet Oord et al. [2016] that compresses the signal to generate high-level representations that transfer to music classification tasks, but contrarily to our synthesizer, it cannot be used to generate waveforms from desired properties of the instrument, pitch and velocity without some input signal.

The minimization by gradient descent of an objective function based on the power spectrogram has already been applied to the transformation of a white noise waveform into a specific sound texture Caracalla and Roebel [2017]. However, to the best of our knowledge, such objective functions have not been used in the context of neural audio synthesis.

[2] `https://pytorch.org/`

2.3. The spectral loss for waveform synthesis

Previous work in audio synthesis on the waveform focused on classification losses Mehri et al. [2016], Oord et al. [2016], Engel et al. [2017]. However, their computational cost needs to be mitigated by quantization, which inherently limits the resolution of the predictions. Ultimately, increasing the number of classes is likely necessary to achieve optimal accuracy. Our approach directly predicts a single continuous value for each audio sample and computes distances between waveforms in the domain of power spectra to be invariant to the original phase of the signal. As a baseline, we also consider computing distances between waveforms using plain mean square error (MSE).

2.3.1. Mean square regression on the waveform

The simplest way of measuring the distance between a reconstructed signal $\hat{x}$ and the reference x is to compute the MSE on the waveform directly, that is taking the Euclidean norm between x and $\hat{x}$,

$$\mathrm{L}_{\mathrm{wav}}\left(x, \hat{x}\right) := \left\|x - \hat{x}\right\|^2 . \tag{2.1}$$

The MSE is not a good perceptual distance between waveforms; for instance, it is extremely sensitive to small shifts in the signal.

2.3.2. Spectral loss

As an alternative to the MSE on the waveform, we suggest taking the Short Term Fourier Transform (STFT) of both x and $\hat{x}$ and comparing their power spectrum in log scale:

$$l(x) := \log\left(\epsilon + \left|\mathrm{STFT}\left[x\right]\right|^2\right). \tag{2.2}$$

We use a STFT that decomposes the original signal x in successive frames of 1024 time steps with a stride of 256, so that there is 75% overlap between two successive frames. The output for a single frame is 513 complex numbers, each representing a specific frequency range. Taking the point-wise squareds modulus of those numbers represents how much energy is present in a specific frequency range. We observed that our models generated higher quality sounds when trained using a log scale of those coefficients. Previous work has come to the same conclusion Engel et al. [2017]. We observed that many entries of the spectrograms are close to zero and that small errors on those parts can add up to form noisy artifacts. In order to favor sparsity in the spectrogram, we use the $\left\|\cdot\right\|_1$ norm instead of the MSE,

$$\mathrm{L}_{\mathrm{stft},1}\left(x, \hat{x}\right) := \left\|l(x) - l(\hat{x})\right\|_1 . \tag{2.3}$$

The value of ϵ controls the trade-off between accurately representing low energy and high energy coefficients in the spectrogram. We found that $\epsilon = 1$ gave the best subjective reconstruction quality, using an unormalized STFT with a Hann window (maximum value to 1), for the NSynth dataset presented in Section 2.5.1.

The STFT is a (complex) convolution operator on the waveform and the squared absolute value of the Fourier coefficients makes the power spectrum differentiable with respect to the generated waveform. Since the generated waveform is itself a differentiable function of the parameters (up to the non-differentiability points of activation functions such as ReLU), the spectral loss (2.3) can be minimized by standard backpropagation. Even though we only consider this spectral loss in our experiments, alternatives to the STFT such as the Wavelet transform also define a differentiable loss.

Non unicity of the waveform representation

To illustrate the importance of the spectral loss instead of a waveform loss, let us consider a problem that arises when generating notes in the test set. Let us assume that one of the instrument is a pure sinusoid. For a given pitch at a frequency f, the audio signal is $x_i = \sin(2\pi i \frac{f}{16000} + \phi)$. Our perception of the signal is not affected by the choice of $\phi \in [0, 2\pi[$, and the spectrogram of x is mostly unaltered. When recording an acoustic instrument, the value of ϕ depends on any number of variables characterizing the physical system that generated the sound and there is no guarantee that ϕ stays constant when playing the same note again. For a synthetic sound, ϕ also depends on implementation details of the software generating the sound.

For a sound that is not in the training set and as far as the model is concerned, ϕ is a random variable that can take any value in the range $[0, 2\pi[$. As a result, x_0 is unpredictable in the range $[-1, 1]$, and the mean square error between the generated signal and the ground truth is uninformative. Even on the training dataset, the model has to use extra resources to remember the value of ϕ for each pitch. We believe that this phenomenon is the reason why training the synthesizer using the MSE on the waveform leads to worse reconstruction performance, even though this loss is sufficient in the context of auto-encoding (see Section 2.5.2). The spectral loss solves this issue since the model is free to choose a single canonical value for ϕ.

However, one should note that the spectral loss is permissive. For instance for stationary sounds, we observe "restarts" where the audio cancels out before restarting with a different initial phase. Another example is given by the artifacts introduced by the strided transposed convolutions. Both of them have a small loss, as they are either very localized in time or frequency, but have clearly audible effects. In practice, we obtain

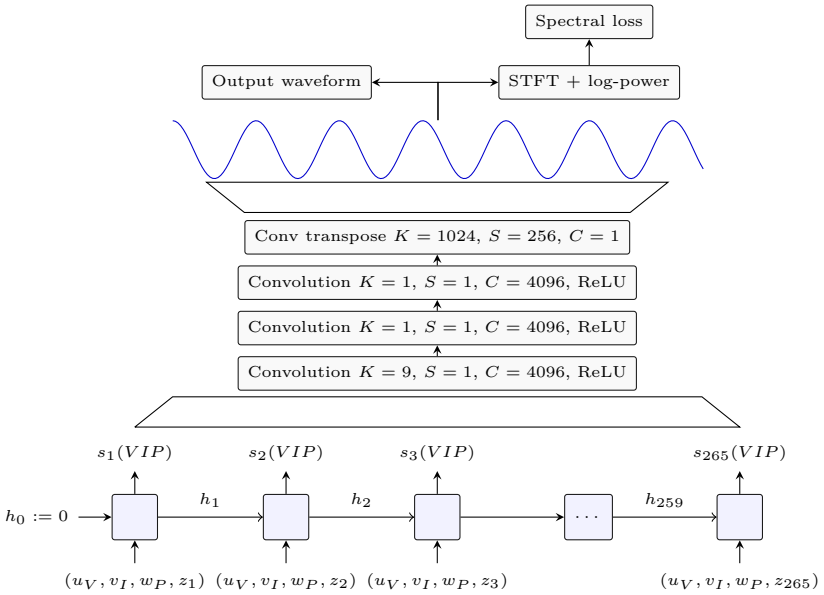


Figure 2.1.: Summary of the entire architecture of SING. u_V, v_I, w_P, z_* represent the look-up tables respectively for the velocity, instrument, pitch and time. h_* represent the hidden state of the LSTM and s_* its output. For convolutional layers, K represents the kernel size, S the stride and C the number of channels.

state-of-the-art results (see Section 2.5), but both issues should be resolved to achieve even higher quality.

2.4. Model

In this section we introduce the SING architecture. It is composed of two parts: a LSTM based sequence generator whose output is plugged to a decoder that transforms it into a waveform. The model is trained to recover a waveform x sampled at 16,000 Hz from the training set based on the one-hot encoded instrument I, pitch P and velocity V. The whole architecture is summarized in Figure 2.1.

2.4.1. LSTM sequence generator

The sequence generator is composed of a 3-layer recurrent neural network with LSTM cells and 1024 hidden units each. Given an example with velocity V, instrument I and pitch P, we obtain 3 embeddings $(u_V, v_I, w_P) \in \mathbb{R}^2 \times \mathbb{R}^{16} \times \mathbb{R}^8$ from look-up tables that are

trained along with the model. Furthermore, the model is provided at each time step with an extra embedding $z_T \in \mathbb{R}^4$ where T is the current time step Sukhbaatar et al. [2015], Gehring et al. [2017], also obtained from a look-up table that is trained jointly. The input of the LSTM is the concatenation of those four vectors (u_V, v_I, w_P, z_T). Although we first experimented with an autoregressive model where the previous output was concatenated with those embeddings, we achieved much better performance and faster training by feeding the LSTM with only on the 4 vectors (u_V, v_I, w_P, z_T) at each time step. Given those inputs, the recurrent network generates a sequence $\forall 1 \leq T \leq N, s(V, I, P)_T \in \mathbb{R}^D$ with a linear layer on top of the last hidden state. In our experiments, we have $D = 128$ and $N = 265$.

2.4.2. Convolutional decoder

The sequence $s(V, I, P)$ is decoded into a waveform by a convolutional network. The first layer is a convolution with a kernel size of 9 and a stride of 1 over the sequence s with 4096 channels followed by a ReLU. The second and third layers are both convolutions with a kernel size of 1 (a.k.a. 1x1 convolution Engel et al. [2017]) also followed by a ReLU. The number of channels is kept at 4096. Finally the last layer is a transposed convolution with a stride of 256 and a kernel size of 1024 that directly outputs the final waveform corresponding to an audio frame of size 1024. In order to reduce artifacts generated by the high stride value, we smooth the deconvolution filters by multiplying them with a squared Hann window. As the stride is one fourth of the kernel size, the squared Hann window has the property that the sum of its values for a given output position is always equal to 1 Griffin and Lim [1984]. Thus the final deconvolution can also be seen as an overlap-add method. We pad the examples so that the final generated audio signal has the right length. Given our parameters, we need $s(V, I, P)$ to be of length $N = 265$ to recover a 4 seconds signal $d(s(V, I, P)) \in \mathbb{R}^{64,000}$.

2.4.3. Training details

All the models are trained on 4 P100 GPUs using Adam Kingma and Ba [2014] with a learning rate of 0.0003 and a batch size of 256.

Initialization with an autoencoder. We introduce an encoder turning a waveform x into a sequence $e(x) \in \mathbb{R}^{N \times D}$. This encoder is almost the mirror of the decoder. It starts with a convolution layer with a kernel size of 1024, a stride of 256 and 4096 channels followed by a ReLU. Similarly to the decoder, we smooth its filters using a squared Hann window. Next are two 1x1 convolutions with 4096 channels and ReLU as an activation

function. A final 1x1 convolution with no non linearity turns those 4096 channels into the desired sequence with D channels. We first train the encoder and decoder together as an auto-encoder on a reconstruction task. We train the auto-encoder for 50 epochs which takes about 12 hours on 4 GPUs.

LSTM training. Once the auto-encoder has converged, we use the encoder to generate a target sequence for the LSTM. We use the MSE between the output $s(V, I, P)$ of the LSTM and the output $e(x)$ of the encoder, only optimizing the LSTM while keeping the encoder constant. The LSTM is trained for 50 epochs using truncated backpropagation through time Williams and Zipser [1995] using a sequence length of 32. This takes about 10 hours on 4 GPUs.

End-to-end fine tuning. We then plug the decoder on top of the LSTM and fine tune them together in an end-to-end fashion, directly optimizing for the loss on the waveform, either using the MSE on the waveform or computing the MSE on the log-amplitude spectrograms and back propagating through the STFT. At that point we stop using truncated back propagation through time and directly compute the gradient on the entire sequence. We do so for 20 epochs which takes about 8 hours on 4 GPUs. From start to finish, SING takes about 30 hours on 4 GPUs to train.

Although we could have initialized our LSTM and decoder randomly and trained end-to-end, we did not achieve convergence until we implemented our initialization strategy.

2.5. Experiments

The source code for SING and a pretrained model are available on our github[3]. Audio samples are available on the article webpage[4].

2.5.1. NSynth dataset

The train set from the NSynth dataset Engel et al. [2017] is composed of 289,205 audio recordings of instruments, some synthetic and some acoustic. Each recording is 4 second long at 16,000 Hz and is represented by a vector $x_{V,I,P} \in [-1, 1]^{64,000}$ indexed by $V \in \{0, 4\}$ representing the velocity of the note, $I \in \{0, \dots, 1005\}$ representing the instrument, $P \in \{0, \dots, 120\}$ representing the pitch. The range of pitches available can

[3] `https://github.com/facebookresearch/SING`

[4] `https://research.fb.com/publications/sing-symbol-to-instrument-neural-generator`

vary depending on the instrument but for any combination of V, I, P, there is at most a single recording.

We did not make use of the validation or test set from the original NSynth dataset because the instruments had no overlap with the training set. Because we use a look-up table for the instrument embedding, we cannot generate audio for unseen instruments. Instead, we selected for each instrument 10% of the pitches randomly that we moved to a separate test set. Because the pitches are different for each instrument, our model trains on all pitches but not on all combinations of a pitch and an instrument. We can then evaluate the ability of our model to generalize to unseen combinations of instrument and pitch. In the rest of the chapter, we refer to this new split of the original train set as the train and test set.

2.5.2. Generalization through pitch completion

We report our results in Table 2.1. We provided both the performance of the complete model as well as that of the autoencoder used for the initial training of SING. This autoencoder serves as a reference for the maximum quality the model can achieve if the LSTM were to reconstruct perfectly the sequence $e(x)$.

Although using the MSE on the waveform works well as far as the autoencoder is concerned, this loss is hard to optimize for the LSTM. Indeed, the autoencoder has access to the signal it must reconstruct, so that it can easily choose which *representation* of the signal to output as explained in Section 2.3.2. SING must be able to recover that information solely from the embeddings given to it as input. It manages to learn some of it but there is an important drop in quality. Besides, when switching to the test set one can see that the MSE on the waveform increases significantly. As the model has never seen those examples, it has no way of picking the right representation. When using a spectral loss, SING is free to choose a *canonical* representation for the signal it has to reconstruct and it does not have to remember the one that was in the training set. We observe that although we have a drop in quality between the train and test set, our model is still able to generalize to unseen combinations of pitch and instrument.

Finally, we tried training a model without the time embedding z_T. Theoretically, the LSTM could do without it by learning to count the number of time steps since the beginning of the sequence. However we do observe a significant drop in performance when removing this embedding, thus motivating our choice.

On Figure 2.2, we represented the rainbowgrams for a particular example from the test set as well as its reconstruction by the Wavenet-autoencoder, SING trained with the spectral and waveform loss and SING without time embedding. Rainbowgrams are

		Spectral loss		Wav MSE	
Model	**Training loss**	*train*	*test*	*train*	*test*
Autoencoder	waveform	0.026	0.028	0.0002	0.0003
SING	waveform	0.075	0.084	0.006	0.039
Autoencoder	spectral	0.028	0.032	N/A	N/A
SING	spectral	0.039	0.051	N/A	N/A
SING no time emb.	spectral	0.050	0.063	N/A	N/A

Table 2.1.: Results on the train and test set of the pitch completion task for different models. The first column specifies the model, either the autoencoder used for the initial training of the LSTM or the complete SING model with the LSTM and the convolutional decoder. We compare models either trained with a loss on the waveform (see (2.1)) or on the spectrograms (see (2.3)). Finally we also trained a model with no temporal embedding.

defined in Engel et al. [2017] as "a CQT spectrogram with intensity of lines proportional to the log magnitude of the power spectrum and color given by the derivative of the phase". A different derivative of the phase will lead to audible deformations of the target signal. Such modification are not penalized by our spectral loss as explained in Section 2.3.2. Nevertheless, we observe a mostly correct reconstruction of the derivative of the phase using SING. More examples from the test set, including the rainbowgrams and audio files are available on the article webpage[5].

2.5.3. Human evaluations

During training, we use several automatic criteria to evaluate and select our models. These criteria include the MSE on spectrograms, magnitude spectra, or waveform, and other perceptually-motivated metrics such as the Itakura-Saito divergence Itakura [1968]. However, the correlation of these metrics with human perception remains imperfect, this is why we use human judgments as a metric of comparison between SING and the Wavenet baseline from Engel et al. [2017].

[5] `https://research.fb.com/publications/sing-symbol-to-instrument-neural-generator`

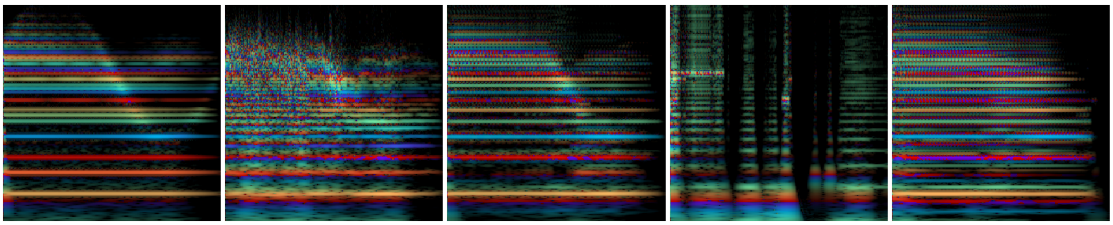


Figure 2.2.: Example of rainbowgrams from the NSynth dataset and the reconstructions by different models. Rainbowgrams are defined in Engel et al. [2017] as "a CQT spectrogram with intensity of lines proportional to the log magnitude of the power spectrum and color given by the derivative of the phase". Time is represented on the horizontal axis while frequencies are on the vertical one. From left to right: ground truth, Wavenet-based autoencoder, SING with spectral loss, SING with waveform loss and SING without the time embedding.

Evaluation of perceptual quality: Mean Opinion Score

The first characteristic that we want to measure from our generated samples is their naturalness: how good they sound to the human ear. To do so, we perform experiments on Amazon Mechanical Turk Buhrmester et al. [2011] to get a Mean Opinion Score for the ground truth samples, and for the waveforms generated by SING and the Wavenet baseline. We did not include a Griffin-Lim based baseline as the authors in Engel et al. [2017] concluded to the superiority of their Wavenet autoencoder.

We randomly select 100 examples from our test set. For the Wavenet-autoencoder, we pass these 100 examples through the network and retrieve the output. The latter is a pre-trained model provided by the authors of Engel et al. [2017][6]. Notice that all of the 100 samples were used for training of the Wavenet-autoencoder, while they were not seen during the training of our models. For SING, we feed it the instrument, pitch and velocity information of each of the 100 samples. Workers are asked to rate the quality of the samples on a scale from 1 ("Very annoying and objectionable distortion. Totally silent audio") to 5 ("Imperceptible distortion"). Each of the 300 samples (100 samples per model) is evaluated by 60 Workers. The quality of the hardware used by Workers being variable, this could impede the interpretability of the results. Thus, we use the crowdMOS toolkit Ribeiro et al. [2011] which detects and discards inaccurate scores. This toolkit also allows to only keep the evaluations that are made with headphones (rather than laptop speakers for example), and we choose to do so as good listening

[6] `https://github.com/tensorflow/magenta/tree/master/magenta/models/nsynth`

conditions are necessary to ensure the validity of our measures. We report the Mean Opinion Score for the ground-truth audio and each of the 2 models in Table 2.2, along with the 95% confidence interval.

We observe that SING shows a significantly better MOS than the Wavenet-autoencoder baseline despite a compression factor which is 66 times higher. Moreover, to spotlight the benefits of our approach compared to the Wavenet baseline, we also report three metrics to quantify the computational load of the different models. The first metric is the training time, expressed in hours multiplied by the number of GPUs. The authors of Engel et al. [2017], mention that their model trains for 10 days on 32 GPUs, which amounts to 7680 hours*GPUs. However, the GPUs used are capable of about half the FLOPs compared to our P100. Therefore, we corrected this value to 3840 hours*GPUs. On the other hand, SING is trained in 30 hours on four P100, which is 32 times faster than Wavenet. A major drawback of autoregressive models such as Wavenet is that the generation process is inherently sequential: generating the sample at time $t + 1$ takes as input the sample at time t. We timed the generation using the implementation of the Wavenet-autoencoder provided by the authors, in its *fastgen* version[7] which is significantly faster than the original model. This yields a 22 minutes time to generate a 4-second sample. On a single P100 GPU, Wavenet can generate up to 64 sequences at the same time before reaching the memory limit, which amounts to 0.2 seconds of audio generated per second. On the other hand, SING can generate 512 seconds of audio per second of processing time, and is thus 2500 times faster than Wavenet. Finally, SING is also efficient in memory compared to Wavenet, as the model size in MB is more than 4 times smaller than the baseline.

ABX similarity measure

Besides absolute audio quality of the samples, we also want to ensure that when we condition SING on a chosen combination of instrument, pitch and velocity, we generate a relevant audio sample. To do so, we measure how close samples generated by SING are to the ground-truth relatively to the Wavenet baseline. This measure is made by performing ABX Macmillan and Creelman [2004] experiments: the Worker is given a ground-truth sample as a reference. Then, they are presented with the corresponding samples of SING and Wavenet, in a random order to avoid bias and with the possibility of listening as many times to the samples as necessary. They are asked to pick the sample which is the closest to the reference according to their judgment. We perform this experiment on 100 ABX triplets made from the same data as for the MOS, each

[7] `https://magenta.tensorflow.org/nsynth-fastgen`

Model	MOS	Training time	Generation speed	Compression factor	Model size
Ground Truth	3.86 ± 0.24	-	-	-	-
Wavenet	2.85 ± 0.24	3840* GPU hours	0.2 sec/sec	32	948 MB
SING	3.55 ± 0.23	120 GPU hours	512 sec/sec	2133	243 MB

Table 2.2.: Mean Opinion Score (MOS) and computational load of the different models. The training time is expressed in hours * GPU units, the generation time is expressed as the number of seconds of audio that can be generated per second of processing time. The compression factor represents the ratio between the dimensionality of the audio sequences ($64,000$ values) and either the latent state of Wavenet or the input vectors to SING. We also report the size of the models, in MB.
(*) Time corrected to account for the difference in FLOPs of the GPUs used.

triplet being evaluated by 10 Workers. On average over 1000 ABX tests, 69.7% are in favor of SING over Wavenet, which shows a higher similarity between our generated samples and the target musical notes than Wavenet.

2.6. Conclusion

We introduced a simple model architecture, SING, based on LSTM and convolutional layers to generate waveforms. We achieve state-of-the-art results as measured by human evaluation on the NSynth dataset for a fraction of the training and generation cost of existing methods. We introduced a spectral loss on the generated waveform as a way of using time-frequency based metrics without requiring a post-processing step to recover the phase of a power spectrogram. We experimentally validated that SING was able to embed music notes into a small dimension vector space where the pitch, instrument and velocity were disentangled when trained with this spectral loss, as well as synthesizing pairs of instruments and pitches that were not present in the training set. We believe SING opens up new opportunities for lightweight quality audio synthesis with potential applications for speech synthesis and music generation.

3. Demucs: music source separation in the waveform domain

Abstract

Source separation for music is the task of isolating contributions, or *stems*, from different instruments recorded individually and arranged together to form a song. Such components include voice, bass, drums and any other accompaniments. Contrarily to many audio synthesis tasks where the best performances are achieved by models that directly generate the waveform, the state-of-the-art in source separation for music is to compute masks on the magnitude spectrum. In this chapter, we first show that an adaptation of Conv-Tasnet [Luo and Mesgarani, 2019], a waveform-to-waveform model for source separation for speech, significantly beats the state-of-the-art on the MusDB dataset, the standard benchmark of multi-instrument source separation. Second, we observe that Conv-Tasnet follows a masking approach on the input signal, which has the potential drawback of removing parts of the relevant source without the capacity to reconstruct it. We propose Demucs, a new waveform-to-waveform model, which has an architecture closer to models for audio generation with more capacity on the decoder. Experiments on the MusDB dataset show that Demucs beats previously reported results in terms of signal to distortion ratio (SDR), but lower than Conv-Tasnet. Human evaluations show that Demucs has significantly higher quality (as assessed by mean opinion score) than Conv-Tasnet, but slightly more contamination from other sources, which explains the difference in SDR. Additional experiments with a larger dataset suggest that the gap in SDR between Demucs and Conv-Tasnet shrinks, showing that our approach is promising.

3.1. Introduction

Cherry first noticed the "cocktail party effect" [Cherry, 1953]: how the human brain is able to separate a single conversation out of a surrounding noise from a room full of people chatting. Bregman later tried to understand how the brain was able to analyse

a complex auditory signal and segment it into higher level streams. His framework for auditory scene analysis [Bregman, 1990] spawned its computational counterpart, trying to reproduce or model accomplishments of the brains with algorithmic means [Wang and Brown, 2006], in particular regarding source separation capabilities.

When producing music, recordings of individual instruments called *stems* are arranged together and mastered into the final song. The goal of source separation is to recover those individual stems from the mixed signal. Unlike the cocktail party problem, there is not a single source of interest to differentiate from an unrelated background noise, but instead a wide variety of tones and timbres playing in a coordinated way. In the SiSec Mus evaluation campaign for music separation [Stöter et al., 2018], those individual stems were grouped into 4 broad categories: (1) `drums`, (2) `bass`, (3) `other`, (4) `vocals`. Given a music track which is a mixture of these four sources, also called the mix, the goal is to generate four waveforms that correspond to each of the original sources. We consider here the case of supervised source separation, where the training data contain music tracks (i.e., mixtures), together with the ground truth waveform for each of the sources.

State-of-the-art approaches in music source separation still operate on the spectrograms generated by the short-time Fourier transform (STFT). They produce a mask on the magnitude spectrums for each frame and each source, and the output audio is generated by running an inverse STFT on the masked spectrograms reusing the input mixture phase [Takahashi and Mitsufuji, 2017, Takahashi et al., 2018]. Several architectures trained end-to-end to directly synthesize the waveforms have been proposed [Lluís et al., 2018, Stoller et al., 2018], but their performances are far below the state-of-the-art: in the last SiSec Mus evaluation campaign [Stöter et al., 2018], the best model that directly predicts waveforms achieves an average signal-to-noise ratio (SDR) over all four sources of 3.2, against 5.3 for the best approach that predicts spectrograms masks (also see Table 3.1 in Section 3.6). An upper bound on the performance of all methods relying on masking spectrograms is given by the SDR obtained when using a mask computed using the ground truth sources spectrograms, for instance the Ideal Ratio Mask (IRM) or the Ideal Binary Mask (IBM) oracles. For speech source separation, Luo and Mesgarani [2019] proposed Conv-Tasnet, a model that reuses the masking approach of spectrogram methods but learns the masks jointly with a convolutional front-end, operating directly in the waveform domain for both the inputs and outputs. Conv-Tasnet surpasses both the IRM and IBM oracles.

Our first contribution is to adapt the Conv-Tasnet architecture, originally designed for monophonic speech separation and audio sampled at 8 kHz, to the task of stereophonic

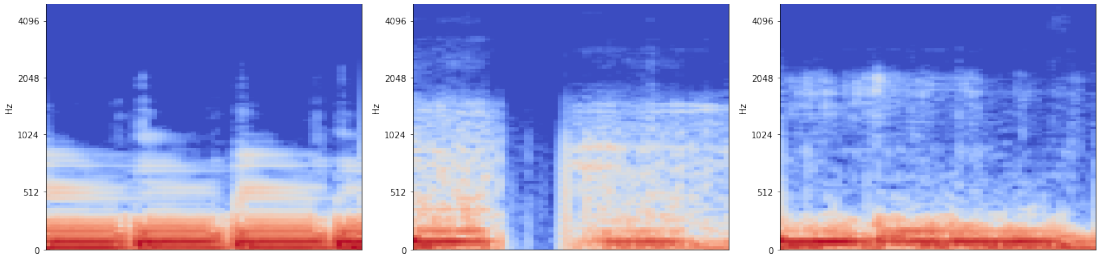


Figure 3.1.: Mel-spectrogram for a 0.8 seconds extract of the `bass` source from the track "Stich Up" of the MusDB test. From left to right: ground truth, Conv-Tasnet estimate and Demucs estimate. We observe that Conv-Tasnet missed one note entirely.

music source separation for audio sampled at 44.1 kHz. Our experiments show that Conv-Tasnet outperforms all previous methods by a large margin, with an SDR (averaged over all sources) of 5.7, but still under the SDR of the IRM oracle at 8.2 [Stöter et al., 2018]. While Conv-Tasnet separates with a high accuracy the different sources, we observed artifacts when listening to the generated audio: a constant broadband noise, hollow instruments attacks or even missing parts. They are especially noticeable on the `drums` and `bass` sources and we give one example on Figure 3.1. Conv-Tasnet uses an over-complete linear representation on which it applies a mask obtained from a deep convolutional network. Because both the encoder and decoder are linear, the masking operation cannot synthesize new sounds. We conjecture that the overlap of multiples instruments sometimes lead to a loss of information that is not reversible by a masking operation.

To overcome the limitations of Conv-Tasnet, our second contribution is to propose Demucs, a new architecture for music source separation. Similarly to Conv-Tasnet, Demucs is a deep learning model that directly operates on the raw input waveform and generates a waveform for each source. Demucs is inspired by models for music synthesis rather than masking approaches. It is a U-net architecture with a convolutional encoder and a decoder based on wide transposed convolutions with large strides inspired by the SING architecture presented in Chapter 2. The other critical features of the approach are a bidirectional LSTM between the encoder and the decoder, increasing the number of channels exponentially with depth, gated linear units as activation function [Dauphin et al., 2017] which also allow for masking, and a new initialization scheme.

We present experiments on the MusDB benchmark, which first show that both Conv-Tasnet and Demucs achieve performances significantly better than the best methods that operate on the spectrogram, with Conv-Tasnet being better than Demucs in terms of

SDR. We also perform human evaluations that compare Conv-Tasnet and our Demucs, which show that Demucs has significantly better perceived quality. The smaller SDR of Demucs is explained by more contamination from other sources. We also conduct an in-depth ablation study of the Demucs architecture to demonstrate the impact of the various design decisions. Finally, we carry out additional experiments by adding 150 songs to the training set. In this experiment, Demucs and TasNet both achieve an SDR of 6.3, suggesting that the gap in terms of SDR between the two models diminishes with more data, making the Demucs approach promising. The 6.3 points of SDR also set a new state-of-the-art, since it improves on the best previous result of 6.0 on the MusDB test set obtained by training with 800 additional songs.

We discuss in more detail the related work in the next Section. We then describe the original Conv-Tasnet model of Luo and Mesgarani [2018] and its adaptation to music source separation. Our Demucs architecture is detailed in Section 3.4. We present the experimental protocol in Section 3.5, and the experimental results compared to the state-of-the-art in Section 3.6. Finally, we describe the results of the human evaluation and the ablation study.

3.2. Related Work

A first category of methods for supervised music source separation work on time-frequency representations. They predict a power spectrogram for each source and reuse the phase from the input mixture to synthesise individual waveforms. Traditional methods have mostly focused on blind (unsupervised) source separation. Non-negative matrix factorization techniques [Smaragdis et al., 2014] model the power spectrum as a weighted sum of a learnt spectral dictionary, whose elements are grouped into individual sources. Independent component analysis [Hyvärinen et al., 2004] relies on independence assumptions and multiple microphones to separate the sources. Learning a soft/binary mask over power spectrograms has been done using either HMM-based prediction [Roweis, 2001] or segmentation techniques [Bach and Jordan, 2005].

With the development of deep learning, fully supervised methods have gained momentum. Initial work was performed on speech source separation [Grais et al., 2014], followed by works on music using simple fully connected networks over few spectrogram frames [Uhlich et al., 2015], LSTMs [Uhlich et al., 2017], or multi scale convolutional/recurrent networks [Liu and Yang, 2018, Takahashi and Mitsufuji, 2017]. Nugraha et al. [2016] showed that Wiener filtering is an efficient post-processing step for spectrogram-based models and it is now used by all top performing models in this

category. Those methods have performed the best during the last SiSec 2018 evaluation [Stöter et al., 2018] for source separation on the MusDB [Rafii et al., 2017] dataset. After the evaluation, a reproducible baseline called Open Unmix has been released by Stöter et al. [2019] and matches the top submissions trained only on MusDB. MMDenseLSTM, a model proposed by Takahashi et al. [2018] and trained on 807 unreleased songs currently holds the absolute record of SDR in the SiSec campaign. Both Demucs and Conv-Tasnet obtain significantly higher SDR.

More recently, models operating in the waveform domain have been developed, so far with worse performance than those operating in the spectrogram domain. A convolutional network with a U-Net structure called Wave-U-Net was used first on spectrograms [Jansson et al., 2017] and then adapted to the waveform domain [Stoller et al., 2018]. Wave-U-Net was submitted to the SiSec 2018 evaluation campaign with a performance inferior to that of most spectrogram domain models by a large margin. A Wavenet-inspired, although using a regression loss and not auto-regressive, was first used for speech denoising [Rethage et al., 2018] and then adapted to source separation [Lluís et al., 2018]. Our model significantly outperforms Wave-U-Net.Given that the Wavenet inspired model performed worse than Wave-U-Net, we did not consider it for comparison.

In the field of monophonic speech source separation, spectrogram masking methods have enjoyed good performance [Kolbæk et al., 2017, Isik et al., 2016]. Luo and Mesgarani [2018] developed a waveform domain methods using masking over a learnable front-end obtained from a LSTM that reached the same accuracy. Improvements were obtained by Wang et al. [2018] for spectrogram methods using the unfolding of a few iterations of a phase reconstruction algorithm in the training loss. In the mean time, Luo and Mesgarani [2019] refined their approach, replacing the LSTM with a superposition of dilated convolutions, which improved the SDR and definitely surpassed spectrogram based approaches, including oracles that use the ground truth sources such as the ideal ratio mask (IRM) or the ideal binary mask (IBM). We show in this chapter that Conv-Tasnet also outperforms all known methods for music source separation. However it suffers from significantly more artifacts than the Demucs architecture we introduce in this chapter, as measured by mean opinion score.

3.3. Adapting Conv-Tasnet for music source separation

We describe in this section the Conv-Tasnet architecture of Luo and Mesgarani [2019] and give the details of how we adapted the architecture to fit the setting of the MusDB

dataset.

Overall framework Each source s is represented by a waveform $x_s \in \mathbb{R}^{C,T}$ where C is the number of channels (1 for mono, 2 for stereo) and T the number of samples of the waveform. The mixture (i.e., music track) is the sum of all sources $x := \sum_{s=1}^{S} x_s$. We aim at training a model g parameterized by θ, such that $g(x) = (g_s(x;\theta))_{s=1}^{S}$, where $g_s(x;\theta)$ is the predicted waveform for source s given x, that minimizes

$$\min_{\theta} \sum_{x \in \mathcal{D}} \sum_{s=1}^{S} L(g_s(x;\theta), x_s) \tag{3.1}$$

for some dataset $\mathcal{D}$ and reconstruction error L. The original Conv-Tasnet was trained using a loss called scale-invariant source-to-noise ratio (SI-SNR), similar to the SDR loss described in Section 3.5. We instead use a simple L1 loss between the estimated and ground truth sources. We discuss in more details regression losses in the context of our Demucs architecture in Section 3.4.2.

The original Conv-Tasnet architecture Conv-Tasnet [Luo and Mesgarani, 2018] is composed of a learnt front-end that transforms back and forth between the input monophonic mixture waveform sampled at 8 kHz and a 128 channels over-complete representation sampled at 1 kHz using a convolution as the encoder and a transposed convolution as the decoder, both with a kernel size of 16 and stride of 8. The high dimensional representation is masked through a separation network composed of stacked residual blocks. Each block is composed of a a 1x1 convolution, a PReLU [He et al., 2015] non linearity, a layer wise normalization over all channels jointly [Ba et al., 2016], a depth-wise separable convolution [Chollet, 2017, Howard et al., 2017] with a kernel size of 3, a stride of 1 and a dilation of $2^{n \bmod N}$, with n the 0-based index of the block and N an hyper-parameter, and another PReLU and normalization. The output of each block participates to the final mask estimation through a skip connection, preceded by a 1x1 convolution. The original Conv-Tasnet counted $3 \times N$ blocks with $N = 8$. The mask is obtained summing the output of all blocks and then applying ReLU. The output of the encoder is multiplied by the mask and before going through the decoder.

Conv-Tasnet for music source separation We adapted their architecture to the task of stereophonic music source separation: the original Conv-Tasnet has a receptive field of 1.5 seconds for audio sampled at 8 kHz, we take $N = 10$ and increased the kernel size (resp. stride) of the encoder/decoder from 16 (resp. 8) to 20 (resp. 10), leading to the same receptive field at 44.1 kHz. We observed better results using $4 \times N$ blocks instead

of $3 \times N$ and 256 channels for the encoder/decoder instead of 128. With those changes, Conv-Tasnet obtained state-of-the-art performance on the MusDB dataset, surpassing all known spectrogram based methods by a large margin as shown in Section 3.6.

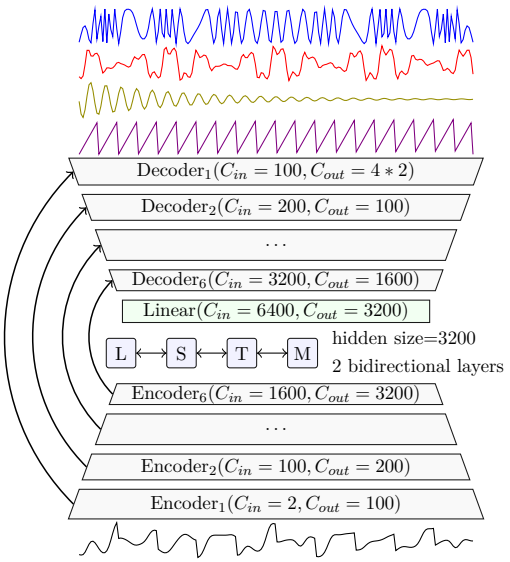
Separating entire songs The original Conv-Tasnet model was designed for short sentences of a few seconds at most. When evaluating it on an entire track, we obtained the best performance by first splitting the input track into chunks of 8 seconds each. We believe this is because of the global layer normalization. During training, only small audio extracts are given, so that a quiet part or a loud part would be scaled back to an average volume. However, when using entire songs as input, it will most likely contain both quiet and loud parts. The normalization will not map both to the same volume, leading to a difference between training and evaluation. We did not observe any side effects when going from one chunk to the next, so we did not look into fancier overlap-add methods.

3.4. The Demucs Architecture

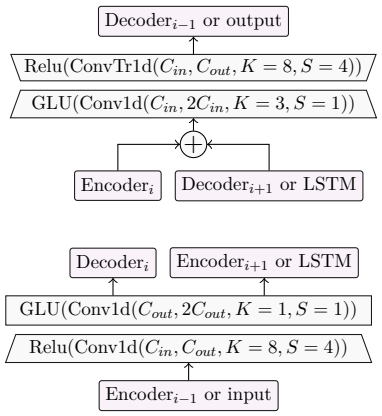
The architecture we propose, which we call Demucs, is described in the next few subsections, and the reconstruction loss is discussed in Section 3.4.2. Demucs takes a stereo mixture as input and outputs a stereo estimate for each source ($C = 2$). It is an encoder/decoder architecture composed of a convolutional encoder, a bidirectional LSTM, and a convolutional decoder, with the encoder and decoder linked with skip U-Net connections. Similarly to other work in generation in both image [Karras et al., 2018, 2017] and sound (see Chapter 2), we do not use batch normalization [Ioffe and Szegedy, 2015] as our early experiments showed that it was detrimental to the model performance. The overall architecture is depicted in Figure 3.2a.

3.4.1. Convolutional auto-encoder

Encoder The encoder is composed of $L := 6$ stacked convolutional blocks numbered from 1 to L. Each block i is composed of a convolution with kernel size $K := 8$, stride $S := 4$, C_{i-1} input channels, C_i output channels and ReLU activation, followed by a convolution with kernel size 1, $2C_i$ output channels and gated linear units (GLU) as activation function [Dauphin et al., 2017]. Since GLUs halve the number of channels, the final output of block i has C_i output channels. A block is described in Figure 3.2b. Convolutions with kernel width 1 increase the depth and expressivity of the model at low computational cost. As we show in our ablation study 3.6.2, the usage of GLU



(a) Demucs architecture with the mixture waveform as input and the four sources estimates as output. Arrows represents U-Net connections.



(b) Detailed view of the layers Decoder$_i$ on the top and Encoder$_i$ on the bottom. Arrows represent connections to other parts of the model. For convolutions, C_{in} (resp C_{out}) is the number of input channels (resp output), K the kernel size and S the stride.

Figure 3.2.: Demucs complete architecture on the left, with detailed representation of the encoder and decoder layers on the right.

activations after these convolutions significantly boost performance. The number of channels in the input mixture is $C_0 = C = 2$, while we use $C_1 := 100$ as the number of output channels for the first encoder block. The number of channels is then doubled at each subsequent block, i.e., $C_i := 2C_{i-1}$ for $i = 2..L$, so the final number of channels is $C_L = 3200$. We then use a bidirectional LSTM with 2 layers and a hidden size C_L. The LSTM outputs $2C_L$ channels per time position. We use a linear layer to take that number down to C_L.

Decoder The decoder is mostly the inverse of the encoder. It is composed of L blocks numbered in reverse order from L to 1. The i-th blocks starts with a convolution with stride 1 and kernel width 3 to provide context about adjacent time steps, input/output channels C_i and a ReLU activation. Finally, we use a transposed convolution with kernel width 8 and stride 4, C_{i-1} outputs and ReLU activation. The S sources are synthesized at the final layer only, after all decoder blocks. The final layer is linear with $S \cdot C_0$ output channels, one for each source (4 stereo channels in our case), without any additional activation function. Each of these channels directly generate the corresponding waveform.

U-network structure Similarly to Wave-U-Net [Jansson et al., 2017], there are skip connections between the encoder and decoder blocks with the same index, as originally proposed in U-networks [Ronneberger et al., 2015]. While the main motivation comes from empirical performances, an advantage of the skip connections is to give a direct access to the original signal, and in particular allows to directly transfers the phase of the input signal to the output, as discussed in Section 3.4.2. Unlike Wave-U-Net, we use transposed convolutions rather than linear interpolation followed by a convolution with a stride of 1. For the same increase in the receptive field, transposed convolutions require 4 times less operations and memory. This limits the overall number of channels that can be used before running out of memory. As we observed that a large number of channels was key to obtaining good results, we favored the use of transposed convolutions, as explained in Section 3.6.

Motivation: synthesis vs masking The approach we follow uses the U-Network architecture [Ronneberger et al., 2015, Stoller et al., 2018, Jansson et al., 2017], and builds on transposed convolutions with large number of channels and large strides (4) inspired by the approach to the synthesis of music notes described in Chaper 2. The U-Net skip connections and the gating performed by GLUs imply that this architecture is expressive enough to represent masks on a learnt representation of the input signal, in

a similar fashion to Conv-Tasnet. The Demucs approach is then more expressive than Conv-Tasnet, and its main advantages are the multi-scale representations of the input and the non-linear transformations to and from the waveform domain.

3.4.2. Loss function

For the reconstruction loss $L(g_s(x;\theta), x_s)$ in (3.1), we either use the average mean square error or average absolute error between waveforms: for a waveform x_s containing T samples and corresponding to source s, a predicted waveform $\hat{x}_s$ and denoting with a subscript t the t-th sample of a waveform, we use one of L_1 or L_2:

$$L_1(\hat{x}_s, x_s) = \frac{1}{T}\sum_{t=1}^{T} |\hat{x}_{s,t} - x_{s,t}| \qquad L_2(\hat{x}_s, x_s) = \frac{1}{T}\sum_{t=1}^{T} (\hat{x}_{s,t} - x_{s,t})^2 \,. \tag{3.2}$$

In generative models for audio, direct reconstruction losses on waveforms can pose difficulties because they are sensitive to the initial phases of the signals: two signals whose only difference is a shift in the initial phase are perceptually the same, but can have arbitrarily high L_1 or L_2 losses. It can be a problem in pure generation tasks because the initial phase of the signal is unknown, and losses on power/magnitude spectrograms are alternative that do not suffer from this lack of specification of the output. Approaches that follow this line either generate spectrograms [e.g., Wang et al., 2017], or use a loss that compares power spectrograms of target/generated waveforms (see Chapter 2).

The problem of invariance to a shift of phase is not as severe in source separation as it is in unconditional generation, because the model has access to the original phase of the signal. The phase can easily be recovered from the skip connections in U-net-style architectures for separation, and is directly used as input of the inverse STFT for methods that generate masks on power spectrograms. As such, losses such as $L1/L2$ are totally valid for source separation. Early experiments with an additional term including a loss on the log-power spectrogram did not suggest that it boosts performance, so we did not pursue this direction any further. Most our experiments use $L1$ loss, and the ablation study presented in Section 3.6.2 suggests that there is no significant difference between $L1$ and $L2$.

3.4.3. Weight rescaling at initialization

The initialization of deep neural networks is known to have a critical impact on the overall performances [Glorot and Bengio, 2010, He et al., 2015], up to the point that Zhang et al. [2019] showed that with a different initialization called fixup, very deep residual networks and transformers can be trained without batch normalization. While

Fixup is not designed for U-Net-style skip connections, we observed that the following different initialisation scheme had great positive impact on performances compared to the standard initialization of He et al. [2015] used in U-Networks.

Considering the so-called Kaiming initialization [He et al., 2015] as a baseline, let us look at a single convolution layer for which we denote w the weights after the first initialization. We take $\alpha := \text{std}(w)/a$, where a is a reference scale, and replace w by $w' = w/\sqrt{\alpha}$. Since the original weights have element-wise order of magnitude $(KC_{\text{in}})^{-1/2}$ where K is the kernel width and C_{in} the number of output channels, it means that our initialization scheme produces weights of order of magnitude $(KC_{\text{in}})^{-1/4}$, together with a non-trivial scale. Based a search over the values [0.01, 0.05, 0.1], we select $a = 0.1$ for all the regular and transposed convolutions, see Section 3.6 for more details. We experimentally observed that on a randomly initialized model applied to an audio extract, it kept the standard deviation of the features along the layers of the same order of magnitude. Without initial rescaling, the output the last layer has a magnitude 20 times smaller than the first.

3.4.4. Randomized equivariant stabilization

A perfect source separation model is time equivariant, i.e. shifting the input mixture by X samples will shift the output Y by the exact same amount. Thanks to its dilated convolutions with a stride of 1, the mask predictor of Conv-Tasnet is naturally time equivariant and even if the encoder/decoder is not strictly equivariant, Conv-Tasnet still verifies this property experimentally [Luo and Mesgarani, 2019]. Spectrogram based method will also verify approximately this property. Shifting the input by a small amount will only reflect in the phase of the spectrogram. As the mask is computed only from the magnitude, and the input mixture phase is reused, the output will naturally be shifted by the same amount. On the other hand, we noticed that our architecture did not naturally satisfy this property. We propose a simple workaround called randomized equivariant stabilization, where we sample S random shifts of an input mixture x and average the predictions of our model for each, after having applied the opposite shift. This technique does not require changing the training procedure or network architecture. Using $S = 10$, we obtained a 0.3 SDR gain, see Section 3.6.2 for more details. It does make evaluation of the model S times slower, however, on a V100 GPU, separating 1 minute of audio at 44.1 kHz with Demucs takes only 0.8 second. With this technique, separation of 1 minute takes 8 seconds which is still more than 7 times faster than real time.

3.5. Experimental setup

3.5.1. Evaluation framework

MusDB and additional data We use the MusDB dataset [Rafii et al., 2017] , which is composed of 150 songs with full supervision in stereo and sampled at 44100Hz. For each song, we have the exact waveform of the `drums`, `bass`, `other` and `vocals` parts, i.e. each of the sources. The actual song, the mixture, is the sum of those four parts. The first 84 songs form the *train set*, the next 16 songs form the *valid set* (the exact split is defined in the `musdb` python package) while the remaining 50 are kept for the *test set*. We collected raw stems for 150 tracks, i.e., individual instrument recordings used in music production software to make a song. We manually assigned each instrument to one of the sources in MusDB. We call this extra supervised data the *stem set*. We also report the performances of Tasnet and Demucs trained using these 150 songs in addition to the 84 from MusDB, to anaylze the effect of adding more training data.

Source separation metrics Measurements of the performance of source separation models was developed by Vincent et al. for blind source separation [Vincent et al., 2006] and reused for supervised source separation in the SiSec Mus evaluation campaign [Stöter et al., 2018]. Similarly to previous work [Stoller et al., 2018, Takahashi and Mitsufuji, 2017, Takahashi et al., 2018], we focus on the SDR (Signal to Distortion Ratio) which measures the log ratio between the volume of the estimated source projection onto the ground truth, and the volume of what is left out of this projection, typically contamination by other sources or artifacts. Other metrics can be defined (SIR and SAR) and we present them in the supplementary material. We used the python package `museval` which provide a reference implementation for the SiSec Mus 2018 evaluation campaign. As done in the SiSec Mus competition, we report the median over all tracks of the median of the metric over each track computed using the `museval` package.

3.5.2. Baselines

As baselines, we selected Open Unmix [Stöter et al., 2019], a 3-layer BiLSTM model with encoding and decoding fully connected layers on spectrogram frames. It was released by the organizers of the SiSec 2018 to act as a strong reproducible baseline and matches the performances of the best candidates trained only on MusDB. We also selected MM-DenseLSTM [Takahashi et al., 2018], a multi-band dense net with LSTMs at different scales of the encoder and decoder. This model was submitted as TAK2 and trained

with 804 extra labeled songs[1]. Both MMDenseLSTM and Open Unmix use Wiener filtering [Nugraha et al., 2016] as a last post processing step. The only waveform based method submitted to the evaluation campaign is Wave-U-Net [Stoller et al., 2018] with the identifier STL2. Metrics were downloaded from the SiSec submission repository. for Wave-U-Net and MMDenseLSTM. For Open Unmix they were provided by their authors[2]. We also provide the metrics for the Ideal Ratio Mask oracle (IRM), which computes the best possible mask using the ground truth sources and is the topline of spectrogram based method [Stöter et al., 2018].

3.5.3. Training procedure

Epoch definition and augmentation We define one epoch over the dataset as a pass over all 11-second extracts with a stride of 1 second. We use a random audio shift between 0 and 1 second and keep 10 seconds of audio from there as a training example. We perform the following data augmentation [Uhlich et al., 2017], also used by Open Unmix and MMDenseLSTM: shuffling sources within one batch to generate one new mix, randomly swapping channels. We additionally multiply each source by ± 1 [Nachmani and Wolf, 2019]. All Demucs models were trained over 240 epochs. Conv-Tasnet was trained for 360 epochs when trained only on MusDB and 240 when trained with extra data and using only 2-seconds audio extracts.

Training setup and hyper-parameters All models are trained with 16 V100 GPUs with 32GB of RAM. We use a batch size of 64, the Adam [Kingma and Ba, 2014] optimizer with a learning rate was chosen among [3e-4, 5e-4] and the initial number of channels was chosen in [64, 80, 100] based on the L1 loss on the validation set. We obtained best performance with a learning rate of $3e-4$ and 100 channels. We then tried 3 different values for the initial weight rescaling reference level described in Section 3.4.3, [0.01, 0.05, 0.1] and selected 0.1. We computed confidence intervals using 5 random seeds in Table 3.1. For the ablation study on Table 3.4, we provide metrics for a single run.

3.6. Experimental results

In this section, we first provide experimental results on the MusDB dataset for Conv-Tasnet and Demucs compared with state-of-the-art baselines. We then dive into the ablation study of Demucs.

[1]Source: `https://sisec18.unmix.app/#/methods/TAK2`
[2]`https://zenodo.org/record/3370486`

Table 3.1.: Comparison of Conv-Tasnet and Demucs to state-of-the-art models that operate on the waveform (Wave-U-Net) and on spectrograms (Open-Unmix without extra data, MMDenseLSTM with extra data) as well as the IRM oracle on the MusDB test set. The *Extra?* indicates the number of extra training songs used. We report the median over all tracks of the median SDR over each track, as done in the SiSec Mus evaluation campaign [Stöter et al., 2018]. The `All` column reports the average over all sources. Demucs metrics are averaged over 5 runs, the confidence interval is the standard deviation over $\sqrt{5}$. In bold are the values that are statistically state-of-the-art either with or without extra training data.

			Test SDR in dB				
Architecture	**Wav?**	**Extra?**	`All`	`Drums`	`Bass`	`Other`	`Vocals`
IRM oracle	✗	N/A	8.22	8.45	7.12	7.85	9.43
Open-Unmix	✗	✗	5.33	5.73	5.23	4.02	6.32
Wave-U-Net	✓	✗	3.23	4.22	3.21	2.25	3.25
Demucs	✓	✗	5.58 ±.03	**6.08** ±.06	**5.83** ±.07	4.12 ±.04	6.29 ±.07
Conv-Tasnet	✓	✗	**5.73** ±.03	**6.08** ±.06	5.66 ±.16	**4.37** ±.02	**6.81** ±.04
Demucs	✓	150	**6.33** ±.02	**7.08** ±.07	6.70 ±.06	4.47±.03	7.05 ±.04
Conv-Tasnet	✓	150	**6.32** ±.04	**7.11** ±.13	**7.00** ±.05	4.44±.03	6.74 ±.06
MMDenseLSTM	✗	804	6.04	6.81	5.40	**4.80**	**7.16**

Table 3.2.: Mean Opinion Scores (MOS) evaluating the quality and absence of artifacts of the separated audio. 38 people rated 20 samples each, randomly sample from one of the 3 models or the ground truth. There is one sample per track in the MusDB test set and each is 8 seconds long. Ratings of 5 means that the quality is perfect (no artifacts).

	Quality Mean Opinion Score				
Architecture	`All`	`Drums`	`Bass`	`Other`	`Vocals`
Ground truth	4.46 ±.07	4.56 ±.13	4.25 ±.15	4.45 ±.13	4.64 ±.13
Open-Unmix	3.03 ±.09	3.10 ±.17	2.93 ±.20	3.09 ±0.16	**3.00** ±.17
Demucs	**3.22** ±.09	**3.77** ±.15	**3.26** ±.18	3.32 ±.15	2.55 ±.20
Conv-Tasnet	2.85 ±.08	3.39 ±.14	2.29 ±.15	3.18 ±.14	2.45 ±.16

Table 3.3.: Mean Opinion Scores (MOS) evaluating contamination by other sources. 38 people rated 20 samples each, randomly sampled from one of the 3 models or the ground truth. There is one sample per track in the MusDB test set and each is 8 seconds long. Ratings of 5 means no contamination by other sources.

	Contamination Mean Opinion Score				
Architecture	`All`	`Drums`	`Bass`	`Other`	`Vocals`
Ground truth	4.59 ±.07	4.44 ±.18	4.69 ±.09	4.46 ±.13	4.81 ±.11
Open-Unmix	3.27 ±.11	3.02 ±.19	**4.00** ±.20	3.11 ±.21	2.91 ±.20
Demucs	3.30 ±.10	3.08 ±.21	**3.93** ±.18	3.15 ±.19	3.02 ±.20
Conv-Tasnet	3.42 ±.09	**3.37** ±.17	3.73 ±.18	**3.46** ±.17	3.10 ±.17

3.6.1. Comparison with baselines

We provide a comparison the state-of-the-art baselines on Table 3.1. The models on the top half were trained without any extra data while the lower half used unreleased training songs. As no previous work included confidence intervals, we considered the single metric provided by for the baselines as the exact estimate of their mean performance.

Quality of the separation We first observe that Demucs and Conv-Tasnet outperforms all previous methods for music source separation. Conv-Tasnet has significantly higher SDR with 5.73, improving by 0.4 over Open-Unmix. Our proposed Demucs architecture has worse overall performance but matches Conv-Tasnet for the `drums` source and surpasses it for the `bass`. When training on 150 extra songs, the two methods have the same overall performance of 6.3 SDR, beating MMDenseLSTM by nearly 0.3 SDR, despite MMDenseLSTM being tained on 804 extra songs. Unlike for speech separation [Luo and Mesgarani, 2019], all methods are still far below the IRM oracle, leaving room for future improvements. We provide results for the other metrics (SIR and SAR) as well as box plots with quantiles over the test set tracks in Appendix, Section A.2. Audio samples for Demucs, Conv-Tasnet and all baselines are provided at the address `https://ai.honu.io/papers/demucs/`.

Human evaluations We noticed strong artifacts on the audio separated by Conv-Tasnet, especially for the `drums` and `bass` sources: static noise between 1 and 2 kHz, hollow instrument attacks or missing notes as illustrated on Figure 3.1. In order to confirm this observation, we organized a mean opinion score survey. We separated 8 seconds extracts from all of the 50 test set tracks for Conv-Tasnet, Demucs and Open-Unmix. We asked 38 participants to rate 20 samples each, randomly taken from one of the 3 models or the ground truth. For each one, they were required to provide 2 ratings on a scale of 1 to 5. The first one evaluated the quality and absence of artifacts (1: many artifacts and distortion, content is hardly recognizable, 5: perfect quality, no artifacts) and the second one evaluated contamination by other sources (1: contamination if frequent and loud, 5: no contamination). We show the results on Tables 3.2 and 3.3. We confirmed that the presence of artifacts in the output of Conv-Tasnet degrades the user experience, with a rating of $2.85 \pm .08$ against $3.22 \pm .09$ for Demucs. On the other hand, Conv-Tasnet samples had less contamination by other sources than Open-Unmix or Demucs, although by a small margin, with a rating of $3.42 \pm .09$ against $3.30 \pm .10$ for Demucs and $3.27 \pm .11$ for Open-Unmix.

Table 3.4.: Ablation study for the novel elements in our architecture described in Section 3.4. We use only the train set from MusDB and report best L1 loss over the valid set throughout training as well the SDR on the test set for the epoch that achieved this loss.

	Valid set	**Test set**
Difference	L1 loss	SDR
no initial weight rescaling	0.172	4.94
no BiLSTM	0.175	5.12
ReLU instead of GLU	0.177	5.19
no 1x1 convolutions in encoder	0.176	5.30
no randomized equivariant stabilization	N/A	5.34
kernel size of 1 in decoder convolutions	0.166	5.51
MSE loss	N/A	5.55
Reference	0.164	5.58

Training speed We measured the time taken to process a single batch of size 16 with 10 seconds of audio at 44.1kHz (the original Wave-U-Net being only trained on 22 kHz audio, we double the time for fairness), ignoring data loading and using `torch.cuda.synchronize` to wait on all kernels to be completed. MMDenseLSTM does not provide a reference implementation. Wave-U-Net takes 1.2 seconds per batch, Open Unmix 0.2 seconds per batch and Demucs 1.6 seconds per batch. Conv-Tasnet cannot be trained with such a large sample size, however a single iteration over 2 seconds of audio with a batch size of 4 takes 0.7 seconds.

Model size The model size for the proposed version of Demucs is 2.4GB, and only 42MB for Conv-Tasnet. Open-Unmix in comparison is 136MB.

3.6.2. Ablation study for Demucs

We provide an ablation study of the main design decisions for Demucs in Table 3.4. Given the cost of training a single model, we did not compute confidence intervals for each variation. Yet, any difference inferior to .06, which is the standard deviation observed over 5 repetitions of the Reference model, could be attributed to noise.

We observe a small but not significant improvement when using the L1 loss instead of the MSE loss. Adding a BiLSTM and using the initial weight rescaling described in Section 3.4.3 provides significant gain, with an extra 0.48 SDR for the first and 0.64 for the second. We observe that using randomized equivariant stabilization as described in Section 3.4 gives a gain of almost 0.3 SDR. We did not report the validation loss as we only use the stabilization when computing the SDR over the test set. We applied the randomized stabilization to Open-Unmix and Conv-Tasnet with no gain, since, as explained in Section 3.4.4, both are naturally equivariant with respect to initial time shifts.

We introduced extra convolutions in the encoder and decoder, as described in Sections 3.4.1. The two proved useful, improving the expressivity of the model, especially when combined with GLU activation. Using a kernel size of 3 instead of 1 in the decoder further improves performance. We conjecture that the context from adjacent time steps helps the output of the transposed convolutions to be consistent through time and reduces potential artifacts arising from using a stride of 4.

3.7. Conclusion

We showed that Conv-Tasnet, a state-of-the-art architecture for speech source separation that predicts masks on a learnt front-end over the waveform domain, achieves state-of-the-art performance for music source separation, improving over all previous spectrogram or waveform domain methods by 0.4 SDR. While Conv-Tasnet has excellent performance to separate sources, it suffers from noticeable artifacts as confirmed by human evaluations. We developed an alternative approach, Demucs, that combines the ability to mask over a learnt representation with stronger decoder capacity that allows for audio synthesis. We conjecture that this can be useful when information is lost in the mix of instruments and cannot simply be recovered by masking. We show that our approach produces audio of significantly higher quality as measures by mean opinion scores and matches the SDR of Conv-Tasnet when trained with 150 extra tracks. We believe those results make it a promising alternative to methods based on masking only.

4. Adabatch: an efficient gradient aggregation rule for sparse optimization

Abstract

We study a new aggregation operator for gradients coming from a mini-batch for *stochastic gradient* (SG) methods that allows a significant speed-up in the case of sparse optimization problems. We call this method AdaBatch and it only requires a few lines of code change compared to regular mini-batch SGD algorithms. We provide a theoretical insight to understand how this new class of algorithms is performing and show that it is equivalent to an implicit per-coordinate rescaling of the gradients, similarly to what Adagrad methods can do. In theory and in practice, this new aggregation allows to keep the same sample efficiency of SG methods while increasing the batch size. Experimentally, we also show that in the case of smooth convex optimization, our procedure can even obtain a better loss when increasing the batch size for a fixed number of samples. We then apply this new algorithm to obtain a parallelizable stochastic gradient method that is synchronous but allows speed-up on par with Hogwild! methods as convergence does not deteriorate with the increase of the batch size. The same approach can be used to make mini-batch provably efficient for variance-reduced SG methods such as SVRG.

4.1. Introduction

We consider large-scale supervised learning with sparse features, such as logistic regression, least-mean-square or support vector machines, with a very large dimension as well as a very large number of training samples with many zero elements, or even an infinite stream. A typical example of such use of machine learning is given by Ads click prediction where many sparse features can be used to improve prediction on a problem with a massive online usage. For such problems, *stochastic gradient* (SG) methods have been used successfully [Bottou and Bousquet, 2008, Bach and Moulines, 2011, McMahan et al., 2013].

Sparse optimization requires the usage of CPUs and unlike other domains in machine

learning, it did not benefit much from the ever increasing parallelism accessible in GPUs or dedicated hardware. The frequency of CPUs has been stagnating and we can no longer rely on the increase of CPU sequential computational power for SG methods to scale with the increase of data [Venu, 2011]. New CPUs now rely on multi-core and sometimes multi-socket design to offer more power to its users. As a consequence, many attemps have been made at parallelizing and distributing SG methods [Zinkevich et al., 2010, 2009, Niu et al., 2011, Hsieh et al., 2015]. Those approaches can be classified in two types: (a) synchronous methods, that seek a speed-up while staying logically equivalent to a sequential algorithm, (b) asynchronous methods, which allow some differences and approximations from the sequential algorithms, such as allowing delays in gradient updates, dropping overlapping updates from different workers or allowing inconsistent reads from the model parameters. The latter such as Hogwild! [Niu et al., 2011] have been more successful as the synchronization overhead between workers from synchronous methods made them impractical.

Such methods however do not lead to a complete provability of convergence for step-sizes used in practice, as most proof methods require some approximation [Niu et al., 2011, Mania et al., 2015]. Proving convergence for such methods is not as straightforward as there is not anymore a clear sequence of iterates that actually exist in memory and conflicting writes to memory or inconsistent reads can occur. When increasing the number of workers it is also likely to increase how stale a gradient update is when being processed.

Synchronous approaches rely mostly on the usage of mini-batches in order to parallelize the workload [Dekel et al., 2012]. Increasing the size of the mini-batches will lead to a reduction of the variance of the gradients and the overall estimator. However for the same number of samples we will be doing B times less iterations where B is the size of the mini-batch. In practice one has to increase the learning rate (i.e., the step size) in order to compensate and achieve the same final accuracy as without mini-batches; however increasing the step size can lead to divergence and is sometimes impossible [Jain et al., 2016]. The decrease in sample efficiency (i.e., a worse performance for a given number of processed training samples) is especially visible early during optimization and will lower over time as the algorithm reaches an asymptotic regime where using mini-batches of size B will have the same sample efficiency as without mini-batches.

In this chapter, we make the following contributions:

- We propose in Section 4.2 a new merging operator for gradients computed over a mini-batch, to replace taking the average. Instead, for each mini-batch we count for each coordinate how many samples had a non zero gradient in that direction.

Rather than taking the sum of all gradients and dividing by B we instead divide each coordinate independently by the number of times it was non zero in the batch. This happens to be equivalent to reconditioning the initial problem in order to exploit its sparsity. Because each coordinate is still an average (albeit a stochastic one), the norm of the gradient will stay under control. In order to notice this effect, one has to look at the problem in a different geometry that accounts for the sparsity of the data. We also draw a parallel with Adagrad-type methods [Duchi et al., 2011, Roux et al., 2008] as our operator is equivalent to an implicit rescaling of the gradients per coordinate.

- We show in Section 4.3 that this new merging rule outperforms regular mini-batch on sparse data and that it can have the same if not an improved sample efficiency compared to regular SGD without mini-batch.

- We explain in Section 4.4 how this can be used to make synchronous parallel or distributed methods able to compete with asynchronous ones while being easier to study as they are logically equivalent to the sequential version.

- We extend our results to variance-reduced SG methods such as SVRG in Section 4.5 and show similar gains are obtained when using AdaBatch.

- We present in Section 4.6 experimental results to support our theoretical claims as well as a proof of concept that our new merging operator can make synchronous parallel SG methods competitive with asynchronous ones. We also extend our experiments to variance reduction methods like SVRG and show that AdaBatch yields similar improvement as in the case of SG methods.

Notations. Throughout this chapter, $\|\cdot\|$ denotes the Euclidean norm on $\mathbb{R}^d$ and for any symmetric positive definite matrix D, $\|\cdot\|_D$ is the norm defined by D so that $\forall x \in \mathbb{R}^d, \|x\|_D^2 = x^T D x$; for a set A, $|A|$ denotes the cardinality of A. If $x \in \mathbb{R}^d$ then $x^{(k)}$ denotes the k-th coordinate and $\mathrm{Diag}\,(x)$ is the $d \times d$ diagonal matrix with the coefficient of x on its diagonal. We define for any integer n, $[n] := \{1, 2, \ldots, n\}$. Finally, for any function $h : \mathbb{R}^d \to \mathbb{R}$, we will define the support of h as

$$\mathcal{S}\,(h) = \{k \in [d] : \exists (x, y) \in \mathbb{R}^d \times \mathbb{R}^d, x^{(k)} \neq y^{(k)} \text{ and } h(x) \neq h(y)\}. \tag{4.1}$$

4.1.1. Problem setup

We consider f a random variable with values in the space of convex functions $\mathcal{F}$ from $\mathbb{R}^d$ to $\mathbb{R}$. We define $F(w) := \mathbb{E}[f(w)]$ and we wish to solve the optimization problem

$$F_* = \min_{w \in \mathbb{R}^d} F(w). \tag{4.2}$$

It should be noted that the gradient f' will only have non zero coordinate along the directions of the support $\mathcal{S}(f)$ so that if the support of f is sparse, so will the update for SG methods. We define $p \in \mathbb{R}^d$ by $\forall k \in [d], p^{(k)} := \mathbb{P}[k \in \mathcal{S}(f)]$. We take $p_{\min} := \min(p)$ and $p_{\max} = \max(p)$.

This setup covers many practical cases, such as finite sum optimization where f would have the uniform distribution over the sum elements or stochastic online learning where f would be an infinite stream of training samples.

One example of possible values for f is given by linear predictions with sparse features. Let us assume X is a random variable with values in $\mathbb{R}^d$ and $\phi : \mathbb{R} \to \mathbb{R}$ a random convex function. Then one can take $f(w) := \phi(X^T w)$; $\mathcal{S}(f)$ would be the same as $\mathcal{S}(X)$ defined as the non zero coordinates of the vector X. The problem given by (4.2) becomes

$$F_* = \min_{w \in \mathbb{R}^d} \mathbb{E}\left[\phi(X^T w)\right]. \tag{4.3}$$

In the case of logistic regression, one would have for instance $\phi(X^T w) = \log(1 + \exp -YX^T w)$ for $Y \in \{-1, 1\}$ the random label associated with the feature vector X.

The convergence properties of SG methods depend on the properties of the Hessian F'' of our objective function F, as we will show in Section 4.3. The closer it is to identity, the faster SG methods will converge and this convergence will be as fast for all the coordinates of w. For example, in the case of sparse linear prediction such as given by (4.3), with binary features $X^{(k)} \in \{0, 1\}$ that are uncorrelated, the Hessian is such that $F''(w) = \mathbb{E}\left[\phi''(X^T w) X X^T\right]$. If there exist M and m so that we have $\forall z \in \mathbb{R}, m \leq \phi''(z) \leq M$, then we immediately have

$$\begin{aligned} \forall w \in \mathbb{R}^d,\ & m(1 - p_{\max})\mathrm{Diag}(p) \preceq F''(w), \\ & F''(w) \preceq M(1 + \textstyle\sum_{k \in [d]} p^{(k)})\mathrm{Diag}(p). \end{aligned} \tag{4.4}$$

We notice here that we have a specific structure to the geometry of F which depends on p and which need to be taken into account. The proof of (4.4) is given in the supplementary material (Section B.3.4).

Finally, we want not only to solve problems (4.2) or (4.3), but to be able to do so while using W workers. Those workers can either be running on the same machine with shared memory or in a distributed fashion.

4.1.2. Related work

There have been several approaches for parallelizing or distributing SG methods.

Parallelized stochastic gradient descent. This approach described by [Zinkevich et al., 2010] consists in splitting a dataset in W different parts and averaging the model obtained by W independent workers. Model averaging always reduces the variance of the final estimator but the impact on the bias is not as clear. For sparse optimization this approach does not in practice outperform a purely sequential algorithm [Niu et al., 2011].

Delayed stochastic gradient descent. The effect of delay for constant step-size stochastic gradient descent has been studied by [Zinkevich et al., 2009]. Allowing for delay will remove the need for synchronization and thus limit the overhead when parallelizing. The main result of Zinkevich et al. [2009] concludes that there is two different regimes. During the first phase, delay will not help convergence, although once the asymptotic terms are dominating, a theoretical linear speedup with the number of worker is recovered.

Using mini-batches is a popular alternative for parallelizing or distributing SGD. In [Dekel et al., 2012], the reduction of the variance of the gradient estimate is used to prove improvement in convergence. Our theoretical and practical results show that in the case of sparse learning, mini-batch do not offer an improvement during the first stage of optimization. We believe our merging rule is a simple modification of mini-batch SGD that can considerably improve convergence speed compared to regular mini-batch.

The case of averaged stochastic gradient descent with constant step size for least-squares regression has been studied in much detail and in that case it is possible to get an explicit expression for the convergence of the algorithm [Jain et al., 2016]. During the first phase of optimization, in order to achieve the same accuracy after a given number of samples, the step size must be increased proportionally to the batch size which is possible up to a point after which the algorithm will diverge. We draw the same conclusions in a more generic case in Section 4.3.

In [Li et al., 2014], a specific subproblem is solved instead of just averaging the gradients in a mini-batch. However, solving a subproblem is much more complex to put in place and requires the tuning of extra parameters. Our method is very simple as it only requires a per-coordinate rescaling of the gradients and does not require any parameter tuning.

Hogwild! is a very simple parallel SG method. Each worker processes training examples completely in parallel, with no synchronization and accessing the same model in memory [Niu et al., 2011]. The overhead is minimal, however the theoretical analysis

is complex. New proof techniques have been introduced to tackle those issues [Mania et al., 2015]. Our contribution here is to make synchronous methods almost as fast as Hogwild!. This has an interest for cases where Hogwild! cannot perform optimally, for instance with a mixture of dense and sparse features, or in the distributed setting where memory cannot be shared. Hogwild! has inspired parallel versions for SDCA, SVRG and SAGA [Hsieh et al., 2015, Mania et al., 2015, Leblond et al., 2017]. AdaBatch can similarly be extended to those algorithms and we provide proof for SVRG. **Cyclades** [Pan et al., 2016] builds on Hogwild!, assigning training samples to specific workers using graph theory results to remove conflicts.

Adagrad. Adagrad [Duchi et al., 2011] performs a per coordinate rescaling dependent on the size of past gradients that has proven to be highly efficient for sparse problem, besides it can be combined with Hogwild! for parallel optimization [Duchi et al., 2013]. Adagrad rescaling is similar in nature to the one performed by AdaBatch. Adagrad has a step size going to 0 with the number of iterations which gives good convergence properties for various problems. On the other hand, AdaBatch works with a wider range of methods such as SVRG. Constant step size has proven useful for least-mean-square problems [Bach and Moulines, 2013] or in the field of deep learning [Sutskever et al., 2013].

4.2. AdaBatch for SGD

In this section, we will focus on constant step size stochastic gradient descent to give an intuition on how AdaBatch works. AdaBatch can be extended in the same way to SVRG (see Section 4.5). We assume we are given a starting point $w_0 \in \mathbb{R}^d$ and we define recursively

$$\forall n > 0, w_n = w_{n-1} - g_n, \tag{4.5}$$

for a sequence g_n of stochastic gradient estimates based on independent gradients $f'_{n,1}, \dots, f'_{n,B}$. We define $g_{\text{mb},n}$ as

$$\forall k \in [d],\ g^{(k)}_{\text{mb},n} = \frac{1}{B} \sum_{b:k \in \mathcal{S}(f)} f'_{n,b}(w_{n-1})^{(k)}. \tag{4.6}$$

Plugging (4.6) into (4.5) yields the regular SGD mini-batch algorithm with constant step size γ and batch size B.

For each iteration $n > 0$ and dimension $k \in [d]$, we denote $D_{n,k} := \{b \in [B] : k \in$

$\mathcal{S}(f_{n,b})\}$. We introduce $g_{\text{ab},n}$, the gradient estimate of AdaBatch, as, $\forall k \in [d]$,

$$g_{\text{ab},n}^{(k)} = \begin{cases} \frac{\sum_{b \in D_{n,k}} f'_{n,b}(w_{n-1})^{(k)}}{|D_{n,k}|} & \text{if } D_{n,k} \neq \varnothing, \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

Instead of taking the average of all gradients, for each coordinate we make an average but taking only the non zero gradients into account. Let us take a coordinate $k \in [d]$; if $p^{(k)}$ is close to 1, then the AdaBatch update for this coordinate will be the same as with regular mini-batch with high probability. On the other hand, if $p^{(k)}$ is close to zero, the update of AdaBatch will be close to summing the gradients instead of averaging them. Adding updates instead of averaging has been shown to be beneficial in previous work such as in CoCoa+ [Ma et al., 2015], a distributed SDCA-inspired optimization algorithm. We observed experimentally that in order to achieve the same performance with mini-batch compared to AdaBatch, one has to take a step size that is proportional to B. This will boost convergence for less frequent features but can lead to divergence when B increases because the gradient for frequent features will get too large. Our method allows to automatically and smoothly move from summing to averaging depending on how frequent a feature is.

Let us consider the expectation for those two updates rules, we have to use the expectation of $g_{\text{mb},n}$ and $g_{\text{ab},n}$. We have immediately $\mathbb{E}[g_{\text{mb},n}] = F'(w_{n-1})$. Thus when using the regular mini-batch update rule, one obtains an unbiased estimate of the gradient. The main advantage of mini-batch is a reduction by a factor B of the stochastic noise near the optimal value, as explained in Section 4.3. With our new rule, using Lemma 1 from the supplementary material (with divisions of probabilites taken element-wise), we have:

$$\mathbb{E}[g_{\text{ab},n}] = \text{Diag}\left(\frac{1-(1-p)^B}{p}\right) F'(w_{n-1}). \tag{4.8}$$

Interestingly, we now have a gradient that is equivalent to a reconditioning of F'. We can draw here a parallel with Adagrad [Duchi et al., 2011] which similarly uses per-coordinate step sizes. When using Adagrad, the update rule becomes

$$w_n = w_{n-1} - \gamma (C_n^{\text{adag}})^{-1} f'_n(w_{n-1}) \text{ with}$$
$$C_n^{\text{adag}} = \text{Diag}\left(\alpha^{-1}\sqrt{\epsilon + \textstyle\sum_{i \in [n-1]} (f'_i(w_{i-1})^{(k)})^2}\right)_{k \in [d]}.$$

The goal is to have an adaptative step size that will have a larger step size for coordinate for which the gradients have a smaller magnitude. One should note a few differences though:

- Adagrad relies on past informations and updates the reconditioning at every iteration. It works without any particular requirement on the problem. Adagrad also forces a decaying step size. Although this give Adagrad good convergence properties, it is not always suitable, for instance when using variance reduction methods such as SVRG where the step size is constant.
- On the other side, the AdaBatch scaling stays the same (in expectation) through time and only tries to exploit the structure coming from the sparsity of the problem. It does not require storing extra information and can be adapted to other algorithms such as SVRG or SAGA.

When used together with mini-batch, Adagrad will act similarly to AdaBatch and maintain the same sample efficiency automatically even when increasing the batch size.

Deterministic preconditioning. One can see that when $B \to \infty$, the reconditioning in (4.8) goes to $\operatorname{Diag}(p)^{-1}$. One could think of directly scaling the gradient by $\operatorname{Diag}(p)^{-1}$ to achieve a tighter bound in (4.4). However this would make the variance of the gradient explodes and thus is not usable in practice as shown in Section 4.3. A key feature of our update rule is stability; because every coordinate of $g_{\mathrm{ab},n}$ is an average, there is no chance it can diverge.

Using directly $\operatorname{Diag}(p)^{-1}$ is not possible, however reconditioning by

$$C_{B,p} := \operatorname{Diag}\left(\frac{1-(1-p)^B}{p}\right) \tag{4.9}$$

when using a mini-batch of size B might just work as it will lead to the same expectation as (4.7). If we define $g_{\mathrm{C},n} := C_{B,p} f'(w_{n-1})$, we immediately have $\mathbb{E}[g_{\mathrm{C},n}] = C_{B,p} F'(w_{n-1})$. Intuitively, $C_{B,p}$ is getting us as close as possible to the ideal reconditioning $\operatorname{Diag}(p)^{-1}$ leveraging the mini-batch size in order to keep the size of the gradients under control. Both $g_{\mathrm{C},n}$ and $g_{\mathrm{ab},n}$ allow to obtain a very similar performance both in theory and in practice, so that which version to choose will depend on the specific task to solve. If it is possible to precompute the probabilities p then one can use $g_{\mathrm{C},n}$ which has the advantages of not requiring the extra step of counting the features present in a batch. On the other hand, using $g_{\mathrm{ab},n}$ allows to automatically perform the same reconditioning with no prior knowledge of p.

4.3. Convergence results

We make the following assumptions which generalize our observation from Section 4.1.1 for sparse linear prediction.

Assumption 1. *We assume there exists a convex compact set $\mathcal{D} \subset \mathbb{R}^d$, μ, L and R strictly positive so that the following assumptions are satisfied.*

1. *The Hessians F'' (resp. f'') of F (resp. f) are such that:*

$$\begin{aligned} &\forall w \in \mathcal{D}, \quad \mu \mathrm{Diag}\,(p) \preceq F''(w) \preceq L \mathrm{Diag}\,(p)\,, \quad \textit{and} \\ &\forall w \in \mathcal{D}, f''(w) \preceq R^2 \,\mathrm{Id}\,. \end{aligned} \tag{4.10}$$

2. *Let $w_* := \arg\min_{w \in \mathcal{D}} F(w)$,*

$$F'(w_*) = 0. \tag{4.11}$$

In particular, w_* is a global minimizer of F over $\mathbb{R}^d$.

Those assumptions are easily met in the case of sparse linear predictions. If $f(w) := \phi(X^T w)$, with $\forall k \in [d]$, $X^{(k)} \in \{0, 1\}$ uncorrelated, $\|X\|^2 \leq G^2$ almost surely and $m \leq \phi'' \leq M$, then the assumptions above are verified for $L = M(1 + \sum_{k \in [d]} p^{(k)})$, $\mu = m(1 - p_{\max})$, and $R^2 = G^2 M$. In the case of the logistic loss, $M := 1/4$ and μ typically exist on any compact but is not explicitly available. Note though that it is not required to know μ in order to train any of the algorithms studied here. More details are given in the supplementary material (Section B.3.4).

Detailed proofs of the following results are given in the supplementary material (Section B.3). Our proof technique is based on a variation from the one introduced by [Needell and Ward, 2016]. It requires an extra projection step on $\mathcal{D}$. In practice however, we did not require it for any reasonable step size that does not make the algorithm diverge and our bounds do not depend on $\mathcal{D}$ because of (4.11). The results are summarized in Table 4.1. We use a constant step size as a convenience for comparing the different algorithms. It is different but equivalent to using a decreasing step size (see [Nesterov and Vial, 2008] just before Corollary 1). For instance, if we know the total number of iterations is $n \gg 1$, taking $\gamma = \frac{6 \log(n)}{n}$, the bias term is approximately $\frac{\mu}{4 \log(n) n^2}$. The variance term which is proportional to γ is a $O(\log(n)/n)$ which is the usual rate for strongly convex SGD.

Bias/variance decomposition. We notice that the final error is made of two terms, one that decreases exponentially fast and measures how quickly we move away from the starting point and one that is constant, proportional to γ and that depends on the stochastic noise around the optimal value. We will call the former the *bias* term and the latter the *variance* term, following the terminology introduced by [Bach and Moulines,

Method	$F_{N/B} - F_*$		$\gamma_{\max}$
Mini-batch	$(1 - \gamma p_{\min}\mu/2)^{N/B}\frac{\delta_0}{\gamma}$	$+\gamma\frac{2\sigma^2}{B}$	$\gamma\left[Lp_{\max} + \frac{2R^2}{B}\right] \leq 1$
AdaBatch	$(1 - \gamma p_{\min}^{+B}\mu/2)^{N/B}\frac{\delta_0}{\gamma}$	$+2\gamma\sigma^2$	$\gamma\left[L + 2R^2\right] \leq 1$
Diag $(p)^{-1}$	$(1 - \gamma\mu/2)^{N/B}\frac{\delta_0}{\gamma}$	$+\gamma\frac{2\sigma^2}{p_{\min}B}$	$\gamma\left[L + \frac{2R^2}{p_{\min}B}\right] \leq 1$

Table 4.1.: Convergence rates for the different methods introduced in Section 4.2. N represents the total number of samples, so that the number of iterations is N/B; $\gamma_{\max}$ is the maximum step size that guarantees this convergence, $\sigma^2 := \mathbb{E}\left[\|f'(w_*)\|^2\right]$ the gradient variance at the optimum, and $\forall p \in [0,1], p^{+B} := 1 - (1-p)^B$. The Diag $(p)^{-1}$ method consists in reconditioning by Diag $(p)^{-1}$. Moreover, $\delta_0 := \|w_0 - w_*\|_A^2$ where $A = \mathrm{Id}$ for mini-batch SGD, $A = \mathrm{Diag}\left(p^{+B}/p\right)$ for AdaBatch and $A = \mathrm{Diag}\,(p)^{-1}$ for the last method.

2013]. The bias term decreases exponentially fast and will be especially important during the early stage of optimization and when μ is very small. The variance term is the asymptotically dominant term and will prevail when close to the optimum. In practical applications, the bias term can be the most important one to optimize for [Défossez and Bach, 2015]. This has also been observed for deep learning, where most of the optimization is spent far from the optimum [Sutskever et al., 2013].

We immediately notice that rescaling gradients by Diag $(p)^{-1}$ is infeasible in practice unless B is taken of the order of $p_{\min}^{-1}$, because of the exploding variance term and the tiny step size.

Mini-batch. Let us now study the results for mini-batch SGD. The variance term is always improved by a factor of B if we keep the same step size. Most previous works on mini-batch only studied this asymptotic term and concluded that because of this linear scaling, mini-batch was efficient for parallel optimization [Dekel et al., 2012]. However, when increasing the batch size, the number of iterations is divided by B but the exponential rate is still the same. The bias term will thus not converge as fast unless we increase the step size. We can see two regimes depending on B. If $B \ll \frac{2R^2}{Lp_{\max}}$, then the constraint is $\gamma \leq \frac{B}{2R^2}$. Thus, we can scale γ linearly and achieve the same convergence for both the variance and bias term as when $B = 1$. However, if $B \gg \frac{2R^2}{Lp_{\max}}$, then the constraint is $\gamma \leq \frac{1}{Lp_{\max}}$. In this regime, it is not possible to scale up infinitely γ and thus it is not possible to achieve the same convergence for the bias term as when $B = 1$.

We have observed this in practice on some datasets.

AdaBatch. With AdaBatch though, if $p_{\min} \ll 1$, then $1-(1-p_{\min})^B \approx Bp_{\min}$. In such case, the bias term is $(1-\gamma B p_{\min}\mu/2)^{N/B}\frac{\delta_0}{\gamma} \approx (1-\gamma p_{\min}\mu/2)^N\frac{\delta_0}{\gamma}$, thus showing that we can achieve the same convergence speed for a given number of samples as when $B=1$ as far as the bias term is concerned. The variance term and the maximum step size are exactly the same as when $B=1$. Thus, as long as $1-(1-p_{\min})^B \approx Bp_{\min}$, AdaBatch is able to achieve at least the same sample efficiency as when $B=1$. This is a worst case scenario, in practice we observe on some datasets an improved efficiency when increasing B, see Section 4.6. Indeed for rare features the variance of the gradient estimate will not be decreased with larger batch-size, however for features that are likely to appear more than once in a batch, AdaBatch will still obtain partial variance reduction through averaging more than one gradient. Although we do not provide the full proof of this fact, this is a consequence of Lemma 2 from the supplementary material.

4.4. Wild AdaBatch

We now have a SG method trick that allows us to increase the size of mini-batches while retaining the same sample efficiency. Intuitively one can think of sample efficiency as how much information we extract from each training example we process i.e. how much the loss will decrease after a given number of samples have been processed. Although it is easy to increase the number of samples processed per seconds when doing parallel optimization, this will only lead to a true speedup if we can retain the same sample efficiency as sequential SGD. If the sample efficiency get worse, for instance when using regular mini-batch, then we will have to perform more iterations to reach the same accuracy, potentially canceling out the gain obtained from parallelization.

When using synchronous parallel SG methods such as [Dekel et al., 2012], using large mini-batches allows to reduce the overhead and thus increase the number of samples processed per second. SGD with mini-batches typically suffers from a lower sample efficiency when B increases. It has been shown to be asymptotically optimal [Li et al., 2014, Dekel et al., 2012], however our results summarized in Section 4.3 show that in cases where the step size cannot be taken too large, it will not be able to achieve the same sample efficiency as SGD without mini-batch.

We have shown in Section 4.3 that for sparse linear prediction, AdaBatch can achieve the same sample efficiency as for $B=1$. Therefore, we believe it is a better candidate than regular mini-batch for parallel SGD. In Section 4.6, we will present our experimental results for Wild AdaBatch, a Hogwild! inspired, synchronous SGD algorithm. Given a

batch size B and W workers, they will first compute in parallel B gradients, wait for everyone to be done and then apply the updates in parallel to update w_n. The key advantage here is that thanks to the synchronization, analysis of this algorithm is easier. We have a clear sequence of iterates w_n in memory and there is no delay or inconsistent read. It is still possible that during the update phase, some updates will be dropped because of overlapping writes to memory, but that can be seen simply as a slight decrease of the step size for those coordinate. In practice, we did not notice any difference with the sequential version of AdaBatch.

4.5. AdaBatch for SVRG

SVRG [Johnson and Zhang, 2013] is a variance-reduced SG method that has a linear rate of convergence on the training error when F is given by a finite mean of functions $F := \frac{1}{N}\sum_{i\in[N]} f_i$. This is equivalent to f following the uniform law over the set $\{f_1, \ldots, f_N\}$. SVRG is able to converge with a constant step size. To do so, it replaces the gradient $f'(w)$ by $f'(w) - f'(y) + F'(y)$ where y is updated every epoch (an epoch being m iterations where m is a parameter to the algorithm, typically of the order of the number of training samples). Next, we show the difference between regular mini-batch SVRG and AdaBatch SVRG and give theoretical results showing improved convergence for the latter.

We now only assume that F verifies the following inequalities for $\mu > 0$, almost surely,

$$\forall w \in \mathbb{R}^d, \quad \mu \mathrm{Diag}\,(p) \preceq F''(w) \text{ and } f(w) \preceq L\mathrm{Diag}\,(p)\,. \tag{4.12}$$

Let us take a starting point $y_0 \in \mathbb{R}^d$ and $m \in \mathbb{N}^*$. For all $s = 0, 1, \ldots$, we have $w_{s,0} := y_s$ and for all $n \in [m]$ let us define

$$w_{s,n} := w_{s,n-1} - \gamma g_{s,n}, \quad y_{s+1} := \frac{1}{m}\sum_{n\in[m]} w_{s,n},$$

with $g_{s,n}$ the SVRG update based on $(f_{s,n,b})_{b\in[B]}$ i.i.d. samples of f. Let us introduce

$$\forall k \in [d], D_{s,n}^{(k)} := \left\{ b \in [B] : k \in \mathcal{S}\left(f'_{s,n,b}\right)\right\}.$$

For any dimension k such that $D_{s,n}^{(k)} \neq \varnothing$ we have

$$g_{s,n}^{(k)} := \frac{1}{C_{s,n}^{(k)}}\Big(\sum_{b\in D_{s,n}^{(k)}} f'_{s,n,b}(w_{s,n-1})^{(k)} - f'_{s,n,b}(y_s)^{(k)} + F'(y_s)^{(k)}/p^{(k)}\Big),$$

and for $D_{s,n}^{(k)} = \varnothing$ we take $g_{s,n}^{(k)} := 0$. Regular mini-batch SVRG is recovered for $C_{s,n}^{(k)} := B$. On the other hand, AdaBatch SVRG is obtained for $C_{s,n}^{(k)} := \left|D_{s,n}^{(k)}\right|$. One can note that we used a similar trick to the one in [Mania et al., 2015] in order to preserve the sparsity of the updates.

For both updates, there exists a choice of γ and m such that

$$\mathbb{E}\left[F(y_s) - F_*\right] \leq 0.9^s (F(y_0) - F_*).$$

For regular mini-batch, it is provably sufficient to take $\gamma_{\text{mb}} = \frac{1}{L}$ and $m_{\text{mb}} \approx \frac{2.2L}{p_{\min}\mu}$. For AdaBatch, we have $\gamma_{\text{ab}} = \frac{1}{10L}$ and $m_{\text{ab}} \approx \frac{20L}{Bp_{\min}\mu}$. We notice that as we increase the batch size, we require the same number of inner iterations when using regular mini-batch update. However, each update requires B times more samples as when $B = 1$. On the other hand when using AdaBatch, m_{ab} is inversely proportional to B so that the total number of samples required to reach the same accuracy is the same as when $B = 1$. We provide detailed results and proofs in the supplementary material (Section B.4). Note that similar results should hold for epoch-free variance-reduced SG methods such as SAGA [Defazio et al., 2014].

4.6. Experimental results

We have implemented both Hogwild! and Wild AdaBatch and compared them on three datasets, *spam*[1], *news20*[2], and *url* [Ma et al., 2009]. *Spam* has 92,189 samples of dimension 823,470 and an average of 155 active features per sample. *News20* has 19,996 samples of dimension 1,355,191 and an average of 455 active features per example. Finally, *url* has 2,396,130 samples of dimension 3,231,961 and an average of 115 active features per example.

We implemented both in C++ and tried our best to optimize both methods. We ran them on an Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz with 24 CPUs divided in two sockets. Each socket contains 12 physical CPUs for 24 virtual ones. In our experiments though, it is better to keep the number of threads under the number of actual physical CPUs on a single socket. We restricted each experiment to run on a single socket in order to prevent NUMA (non uniform memory access) issues. For all the experiments we ran (and all methods), we perform a grid search to optimize the step size (or the main parameter of Adagrad). We then report the test error on a separate test set.

Wild AdaBatch. We trained each algorithm for logistic regression with 5 passes over the dataset, 1 pass only in the case of *url*. We normalized the features so that each

[1] `http://plg.uwaterloo.ca/~gvcormac/trecspamtrack05/trecspam05paper.pdf`
[2] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`

sample has norm 1. For each dataset, we evaluate 3 methods: Wild AdaBatch (AB) and Wild mini-batch SGD (MB, the same as AB but with the regular average of the gradients) as well as Hogwild! (HW) for various numbers of workers W. For AdaBatch and mini-batch SGD, the batch size is set to $B = 10W$ for *news20* and $B = 50$ for *url* and *spam* which was giving a better speedup for those datasets. We take W going from 1 to 12, which is the maximum number of physical CPUs on a single socket on our machine. We also evaluate purely sequential SGD without mini-batch (SEQ).

We only present here the result for *news20*. The figures for the other datasets can be found in the supplementary material Section B.5. In Figure 4.1 we show the convergence as a function of the wall-clock time. In Figure 4.3, we give the wall-clock time to reach a given test error (where our method is achieving close to a linear speed-up) and the number of processed samples per seconds. On *news20*, the gain in sample efficiency actually allows Wild AdaBatch to reach the goal the fastest, even though Wild AdaBatch can process less samples per seconds than Hogwild!, thanks to its improved sampled efficiency.

Comparison with Adagrad. We compare AdaBatch with Adagrad for various batch-size on Figure 4.4 when trained with a fixed number of samples, so that when B increases, we perform less iterations. On the *url* dataset, Adagrad performs significantly better than AdaBatch, however we notice that as the batch-size increases, the gap between AdaBatch and Adagrad reduces. On the *spam* dataset with the least-mean-square loss, constant step size SGD performs better than Adagrad. We believe this is because Adagrad is especially well suited for non strictly convex problems. For strictly convex problem though, constant step size SGD is known to be very efficient [Bach and Moulines, 2013]. We also plotted the performance of constant step size regular mini-batch SGD. In all cases, regular mini-batch scales very badly as the batch size increases. We fine tune the step size for each batch size and observed the regular mini-batch will take a larger step size for small batch sizes, that allows to keep roughly the same final test error. However, when the batch size increases too much, this is no longer possible as it makes optimization particularly unstable, thus leading to a clear decrease in sample efficiency. Finally, AdaBatch can even improve the sample complexity when increasing B. We believe this comes from the variance reduction of the gradient for features that occurs more than once in a mini-batch, which in turn allows for a larger step size.

SVRG. We also compared the effect of AdaBatch on SVRG. On Figure 4.2 we show the training gap $F_N - F^*$ on *url* for the log loss with a small L2 penalty. This penalty is given by $\frac{10^{-4}}{2} \|w\|^2_{\mathrm{diag}(p)}$, chosen to respect our hypothesis and to prevent overfitting without degrading the testing error. All the models are trained with 10 iterations over all

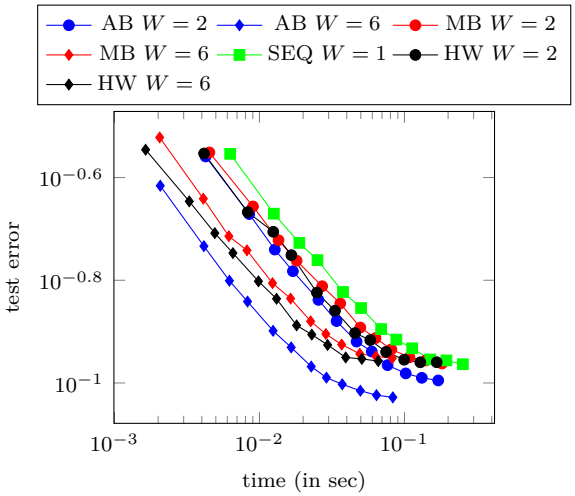


Figure 4.1.: Convergence result for *news20*. The error is given as a function of the wall-clock time.

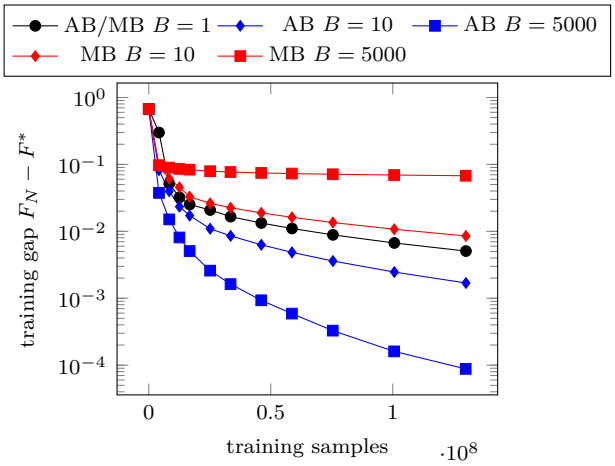


Figure 4.2.: Comparison of the training gap $F_n - F_*$ for regular mini-batch vs. AdaBatch with SVRG on *url* for the log loss with an L2 penalty of $\frac{10^{-4}}{2} \|w\|^2_{\text{diag}(p)}$.

the samples in the dataset, so that if B is larger, the model will perform less iterations. We observe that as we increase the batch size, the sample efficiency of regular mini-batch deteriorates. The variance reduction coming from using a larger mini-batch is not sufficient to compensate the fact that we perform less iterations, even when we increase the step size. On the other hand, AdaBatch actually allows to improve the sample efficiency as it allows to take a larger step size thanks to the gradient variance reduction for the coordinates with $p^{(k)}B$ large enough. To the best of ourknowledge, AdaBatch is the first mini-batch aggregation rule that is both extremely simple and allows for a better sample efficiency than sequential SGD.

4.7. Conclusion

We have introduced a new way of merging gradients when using SG methods with mini-batches. We have shown both theoretically and experimentally that this approach allows to keep the same sample efficiency as when not using any mini-batch and sometimes even improve it. Thanks to this feature, AdaBatch allowed us to make synchronous parallel SG methods competitive with Hogwild!. Our approach can extend to any SG methods

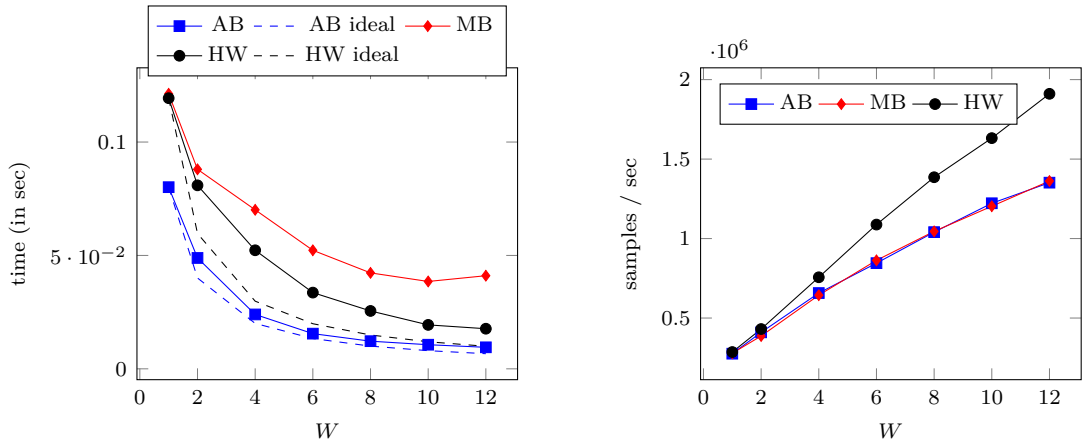


Figure 4.3.: **Left**: time to achieve a given test error when varying the number of workers on *news20*. The dashed line is the ideal speedup. **Right**: number of process sampled per second as a function of W.

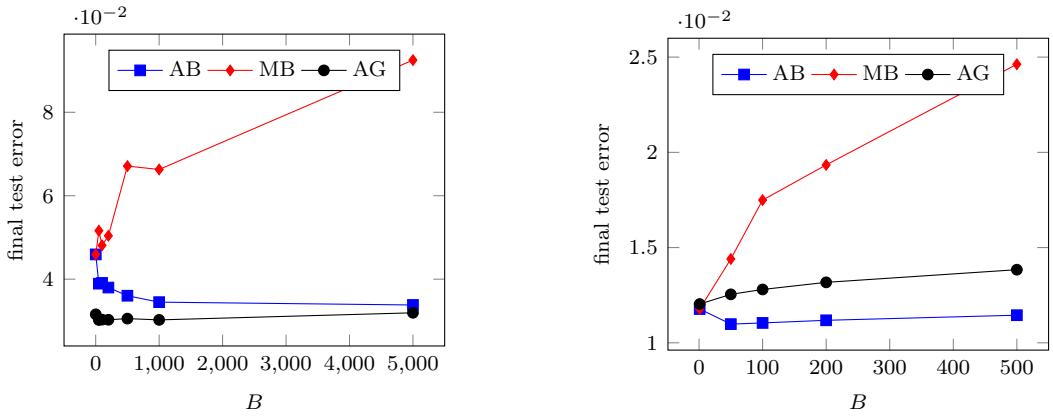


Figure 4.4.: Comparison of Adagrad (AG), AdaBatch (AB) and regular mini-batch (MB). **Left**: on *url* for the log loss; **right**: on *spam* for the least-mean-square loss. We plot the final test error after the same number of samples (larger batches means less iterations).

including variance-reduced methods. Although not explored yet, we also believe that AdaBatch is promising for distributed optimization. In such a case, memory is no longer shared so that Hogwild! cannot be used. Distributed mini-batch or SGD with delay have been used in such case [Dekel et al., 2012, Zinkevich et al., 2009]; AdaBatch is a few line change for distributed mini-batch which could vastly improve the convergence of those methods.

5. On the Convergence of Adam and Adagrad

Abstract

We provide a simple proof of the convergence of the optimization algorithms Adam and Adagrad with the assumptions of smooth gradients and almost sure uniform bound on the ℓ_∞ norm of the gradients. This work builds on the techniques introduced by Ward et al. [2019] and extends them to the Adam optimizer. We show that in expectation, the squared norm of the objective gradient averaged over the trajectory has an upper-bound which is explicit in the constants of the problem, parameters of the optimizer and the total number of iterations N. This bound can be made arbitrarily small. In particular, Adam with a learning rate $\alpha = 1/\sqrt{N}$ and a momentum parameter on squared gradients $\beta_2 = 1 - 1/N$ achieves the same rate of convergence $O(\ln(N)/\sqrt{N})$ as Adagrad. Thus, it is possible to use Adam as a finite horizon version of Adagrad, much like constant step size SGD can be used instead of its asymptotically converging decaying step size version.

5.1. Introduction

First order methods with adaptive step sizes have proved useful in many fields of machine learning, be it for sparse optimization [Duchi et al., 2013], tensor factorization [Lacroix et al., 2018] or deep learning [Goodfellow et al., 2016].

Adagrad [Duchi et al., 2011] rescales each coordinate by a sum of squared past gradient values. While Adagrad proved effective for sparse optimization [Duchi et al., 2013], experiments showed that it under-performed when applied to deep learning [Wilson et al., 2017]. The large impact of past gradients prevents it from adapting to local changes in the smoothness of the function. With RMSProp, Tieleman and Hinton [2012] proposed an exponential moving average instead of a cumulative sum to forget past gradients. Adam [Kingma and Ba, 2014], currently one of the most popular adaptive algorithms in deep learning, built upon RMSProp and added corrective term to the step sizes at the

beginning of training, together with heavy-ball style momentum.

In the online convex optimization setting, Adagrad was shown to achieve minimal regret for online convex optimization [Duchi et al., 2011]. In the original Adam paper, Kingma and Ba [2014] offered a proof that it would converge to the optimum with a step size decaying as $O(1/\sqrt{N})$ where N is the number of iterations, even though this proof was later questioned by Reddi et al. [2019]. In the non-convex setting, Ward et al. [2019] showed convergence with rate $O(\ln(N)/\sqrt{N})$ to a critical point for the scalar, i.e., single step size, version of Adagrad. Zou et al. [2019b] extended this proof to the vector case, while Zou et al. [2019a] proved the convergence of Adam when the decay of the exponential moving average scales as $1 - 1/N$ and the learning rate scales as $1/\sqrt{N}$. Moreover, compared to plain stochastic gradient descent, adaptive algorithms are known to be less sensitive to hyperparameter setting. The theoretical results above confirm this observation by showing the convergence for a step size parameter that does not depend on the regularity parameters of the objective function or the bound on the variance of the stochastic gradients.

In this chapter, we present a new proof of convergence to a critical point for Adagrad and Adam for stochastic non-convex smooth optimization, under the assumptions that the stochastic gradients of the iterates are almost surely bounded. These assumptions are weaker and more realistic than those of prior work on these algorithms. In particular, we show for a fully connected feed forward neural networks with sigmoid activation trained with ℓ_2 regularization, the iterates of Adam or Adagrad almost surely stay bounded, which in turn implies a bound on the stochastic gradient as long as the training input data is also bounded. We recover the standard $O(\ln(N)/\sqrt{N})$ convergence rate for Adagrad for all step sizes, and the same rate with Adam with an appropriate rescaling of the step sizes and decay parameters. Compared to previous work, our bound significantly improves the dependency on the momentum parameter β_1. The best know bounds for Adagrad and Adam are respectively in $O((1-\beta_1)^{-3})$ and $O((1-\beta_1)^{-5})$ (see Section 5.3), while our result is in $O((1-\beta_1)^{-1})$ for both algorithms.

Another important contribution of this work is a significantly simpler proof than previous ones. The reason is that in our approach, the main technical steps are carried out jointly for Adagrad and Adam with constant parameters, while previous attempts at unified proofs required varying parameters through the iterations [Chen et al., 2019, Zou et al., 2019a,b].

The precise setting and assumptions are stated in the next section, and previous work is then described 5.3. Next, we discuss the relevance of our assumptions in the context of deep learning using containment arguments inspired by Bottou [1999]. The main

theorems are presented in Section 5.5, followed by a full proof for the case without momentum in Section 5.6. The full proof of the convergence with momentum is deferred to the supplementary material.

5.2. Setup

5.2.1. Notation

Let $d \in \mathbb{N}$ be the dimension of the problem and take $[d] = \{1, 2, \dots, d\}$. Given a function $h : \mathbb{R}^d \to \mathbb{R}$, we note ∇h its gradient and $\nabla_i h$ the i-th component of the gradient. In this chapter, ϵ represents a small constant, e.g., 10^{-8}, used for numerical stability. Given a sequence $(u_n)_{n\in\mathbb{N}}$ with $\forall n \in \mathbb{N}, u_n \in \mathbb{R}^d$, we note $u_{n,i}$ for $n \in \mathbb{N}$ and $i \in [d]$ the i-th component of the n-th element of the sequence.

We want to optimize a function $F : \mathbb{R}^d \to \mathbb{R}$. We assume there exists a random function $f : \mathbb{R}^d \to \mathbb{R}$ such that $\mathbb{E}[\nabla f(x)] = \nabla F(x)$ and that we have access to an oracle providing i.i.d. samples $(f_n)_{n\in\mathbb{N}^*}$. In machine learning, $x \in \mathbb{R}^d$ typically represents the weights of a linear or deep model, f represents the loss from individual training examples or minibatches, and F is the full training objective function. The goal is to find a critical point of F.

5.2.2. Adaptive methods

We study a family of algorithms that covers both Adagrad [Duchi et al., 2011] and Adam [Kingma and Ba, 2014]. We assume we have an infinite stream $(f_n)_{n\in\mathbb{N}^*}$ of i.i.d. copies of f, $0 \le \beta_2 \le 1$ and $0 \le \beta_1 < \beta_2$, and a non negative sequence $(\alpha_n)_{n\in\mathbb{N}^*}$.

Given $x_0 \in \mathbb{R}^d$ our starting point and $m_0 = 0$, $v_0 = 0$, we iterate, for every $n \in \mathbb{N}^*$,

$$m_{n,i} = \beta_1 m_{n-1,i} + \nabla_i f_n(x_{n-1}) \tag{5.1}$$

$$v_{n,i} = \beta_2 v_{n-1,i} + (\nabla_i f_n(x_{n-1}))^2 \tag{5.2}$$

$$x_{n,i} = x_{n-1,i} - \alpha_n \frac{m_{n,i}}{\sqrt{\epsilon + v_{n,i}}}. \tag{5.3}$$

The real number β_1 is a heavy-ball style momentum parameter [Polyak, 1964], while β_2 controls the rate at which the scale of past gradients is forgotten.

Taking $\beta_1 = 0$, $\beta_2 = 1$ and $\alpha_n = \alpha$ gives Adagrad. While the original Adagrad algorithm [Duchi et al., 2011] did not include a heavy-ball-like momentum, our analysis also applies to the case $\beta_1 > 0$. On the other hand, when $0 < \beta_2 < 1$, $0 \le \beta_1 < \beta_2$,

taking

$$\alpha_n = \alpha(1-\beta_1)\sqrt{\sum_{j=0}^{n-1}\beta_2^j}, \tag{5.4}$$

leads to an algorithm close to Adam. Indeed, the step size in (5.4) is rescaled based on the number of past gradients that were accumulated. This is equivalent to the correction performed by Adam, which compensates for the possible smaller scale of v_n when only few gradients have been accumulated.[1] When there is no momentum ($\beta_1 = 0$) the only difference with Adam is that ϵ in (5.3) is outside the square root in the original algorithm. When $\beta_1 > 0$, an additional difference is that we do not compensate for m_n being smaller during the first few iterations.

The slight difference in step size when $\beta_1 > 0$ simplifies the proof at a minimum practical cost: the first few iterations of Adam are usually noisy, in particular due to v_n having seen few samples, and (5.4) is equivalent to taking a smaller step size during the first $\frac{1}{1-\beta_1}$ iterations. Since Kingma and Ba [2014] suggested a default value of $\beta_1 = 0.9$, our update rule differs significantly from the original Adam only during the first few tens of iterations.

5.2.3. Assumptions

We make four assumptions. We first assume F is bounded below by F_*, that is,

$$\forall x \in \mathbb{R}^d,\ F(x) \geq F_*. \tag{5.5}$$

We assume *the iterates are contained within an ℓ_∞ ball almost surely*,

$$\forall n \in \mathbb{N},\ \|x_n\|_\infty \leq B \quad \text{a.s.}. \tag{5.6}$$

We then assume *the ℓ_∞ norm of the stochastic gradients is almost surely bounded over this ball*: for all $x \in \mathbb{R}^d$ such that $\|x\|_\infty \leq B$,

$$\|\nabla f(x)\|_\infty \leq R - \sqrt{\epsilon} \quad \text{a.s.}, \tag{5.7}$$

and finally, the *smoothness of the objective function over this ball*, e.g., its gradient is L-Liptchitz-continuous with respect to the ℓ_2-norm: for all $x, y \in \mathbb{R}^d$ such that $\|x\|_\infty \leq B$ and $\|y\|_\infty \leq B$,

$$\|\nabla F(x) - \nabla F(y)\|_2 \leq L \|x - y\|_2 . \tag{5.8}$$

[1] Adam updates are usually written $\alpha_n = \alpha(1-\beta_1)\sqrt{1-\beta_2^n}$ and $v_{n,i} = \beta_2 v_{n-1,i} + (1-\beta_1)\left(\nabla_i f_n(x_{n-1})\right)^2$. These are equivalent to ours because the factor $(1-\beta_1)$ is transfered to a multiplication of α_n by $1/\sqrt{1-\beta_2}$. The same apply to m_n.

Note that, if F is L-smooth over $\mathbb{R}^d$ and the stochastic gradients are uniformly almost surely bounded over $\mathbb{R}^d$, then one can take $B = \infty$, and (5.6) is then verified. This case matches more usual assumptions, but it is rarely met in practice, as explained in Section 5.3. However, note that (5.6) is verified with $B < \infty$ for some cases of deep neural network training, as proven in Section 5.4.

5.3. Related work

Work on adaptive optimization methods started with the seminal papers of McMahan and Streeter [2010] and Duchi et al. [2011]. They showed that adaptive methods like Adagrad achieve an optimal rate of convergence of $O(1/\sqrt{N})$ for convex optimization [Agarwal et al., 2009]. Practical experiences with training deep neural networks led to the development of adaptive methods using an exponential moving average of past squared gradients like RMSProp [Tieleman and Hinton, 2012] or Adam [Kingma and Ba, 2014].

Kingma and Ba [2014] claimed that Adam with decreasing step sizes converges to an optimal solutions for convex objectives. However, the proof contained a mistake spotted by Reddi et al. [2019], who also gave examples of convex problems where Adam does not converge to an optimal solution. They proposed AMSGrad as a convergent variant of Adam, which consisted in retaining the maximum value of the exponential moving average. The examples given by Reddi et al. [2019] illustrate a behavior of Adam that is coherent with our results and previous work [Zou et al., 2019a], because they use a small exponential decay parameter $\beta_2 < 1/5$. Under our assumptions, Adam with constant β_2 is guaranteed to not diverge, but it is not guaranteed to converge to a stationary point.

Regarding the non-convex setting, Li and Orabona [2019] showed the convergence of Adagrad for the non-convex case but under unpractical conditions, in particular the step size α should verify $\alpha \leq \sqrt{\epsilon}/L$. Ward et al. [2019] showed the convergence of a variant of Adagrad (in the sense of the expected squared norm at a random iterate) for any value of α, but only for the "scalar" version of Adagrad, with a rate of $O(\ln(N)/\sqrt{N})$. While our approach builds on this work, we significantly extend it to apply to both Adagrad and Adam, in their coordinate-wise version used in practice, while also supporting heavy-ball momentum.

Zou et al. [2019b] showed the convergence of Adagrad with either heavy-ball or Nesterov style momentum. We recover a similar result for Adagrad with heavy-ball momentum, under different but interchangeable hypotheses, as explained in Section 5.5.2. Their proof technique work with a variety of averaging scheme for the past squared gradients,

including Adagrad. In that case, we obtain the same rate as them as a function of N (i.e., $O(\ln(N)/\sqrt{N})$), but we improve the dependence on the momentum parameter β_1 from $O((1-\beta_1)^{-3})$ to $O((1-\beta_1)^{-1})$. Chen et al. [2019] also present bounds for Adagrad and Adam, without convergence guarantees for Adam. The dependence of their bounds in β_1 is worse than that of Zou et al. [2019b].

Zou et al. [2019a] propose unified convergence bounds for Adagrad and Adam. We recover the same scaling of the bound with respect to α and β_2. However their bound has a dependency in $O((1-\beta_1)^{-5})$ with respect to β_1, while we prove $O((1-\beta_1)^{-1})$, a significant reduction.

In previous work [Zou et al., 2019a,b], the assumption given by (5.7) is replaced by

$$\forall x \in \mathbb{R}^d,\ \mathbb{E}\left[\|\nabla f(x)\|_2^2\right] \leq R^2. \tag{5.9}$$

First, notice that we assume an almost sure bound instead of a bound on the expectation of the squared stochastic gradients. The almost sure bound is a stronger requirement but it gives a stronger convergence result, namely a bound on the expected norm of the full gradient at the iterates taken to the power 2 instead of 4/3, as explained in Section 5.5.2. The proof remains mostly identical whether we assume an almost sure bound or bound in expectation of the squared stochastic gradients. Given that for a fixed $x \in \mathbb{R}^d$, the variance of the stochastic gradients for machine learning models comes from the variance of the training data, going from a bound in expectation of the squared gradients to an almost sure bound is easily accomplished by the removal of outliers in the training set.

Second, assumption (5.9) rarely hold in practice as it assume boundness of the gradient over $\mathbb{R}^d$. It is not verified by any deep learning network with more than one layer, linear regression, nor logistic regression with ℓ_2 regularization. In fact, a deep learning network with two layers is not even L-smooth over $\mathbb{R}^d$, as the norm of the gradient for the first layer is multiplied by the norm of the gradient for the second layer. We show in the next section that for deep neural networks with sigmoid activations and ℓ_2 regularization, (5.6) is verified, as long as the data in the training set is bounded, which implies both (5.7) and (5.8).

5.4. Containment of the iterates

Following Bottou [1999] we show in this section that (5.6) is verified for a fully connected feed forward neural network with sigmoid activations and ℓ_2 regularization. The goal of this section is to show that there is an upper-bound on the weights of this neural

network when trained with Adam or Adagrad even though the bound we obtain grows super exponentially with the depth.

We assume that $\beta_1 = 0$ for simplicity, so that for any iteration $n \in \mathbb{N}^*$ and coordinate $i \in [d]$, $m_{n,i} = \nabla_i f_n(x_{n-1})$. We assume $x \in \mathbb{R}^d$ is the concatenation of $[w_1\, w_2\, \dots\, w_l]$, where l is the number of layers and for all $s \in [l]$, $w_s \in \mathbb{R}^{c_s \times c_{s-1}}$ is the weight of the s-th layer, c_0 being the dimension of the input data. For clarity, we assume $c_l = 1$, i.e. the neural network has a single output. The fully connected network is represented by the function,

$$\forall z \in \mathbb{R}^{c_0}, h(x, z) = \sigma(w_l \sigma(w_{l-1} \dots \sigma(w_1 z))).$$

Then, the stochastic objective function is given by,

$$f(x) = D(h(x, Z), Y) + \frac{\lambda}{2} \|x\|_2^2,$$

where Z is a random variable over $\mathbb{R}^{c_0}$ representing the input training data, Y is the label over a set $\mathcal{Y}$, D is the loss function, and λ the ℓ_2 regularization parameter. We assume that the ℓ_∞ norm of Z is almost surely bounded by 1 and that for any label $y \in \mathcal{Y}$, $|D'(\cdot, y)| \leq M'$. This is verified for the Huber loss, or the cross entropy loss. When writing D', we always mean its derivative with respect to its first argument. Finally, we note $o_s(x, z)$ the output of the s-th layer, i.e.

$$\forall s \in [l], o_s(x, z) = \sigma(w_s \sigma(w_{s-1} \dots \sigma(w_1 z))),$$

and $o_0(x, z) = z$. In particular, $\|o_s(x, z)\|_\infty \leq 1$.

We will prove the bound on the iterates through induction, starting the output layer and going backward up to the input layer. We assume all the weights are initialized with a size much smaller than the bound we will derive.

5.4.1. Containment of the last layer

In the following, ∇_w is the Jacobian operator with respect to the weights of a specific layer w. Taking the derivative of $f(x)$ with respect to w_l, we get,

$$\begin{aligned} \nabla_{w_l} D(h(x, Z), Y) &= D'(h(x, Z), Y) \nabla_{w_l} h(x, Z) \\ &= D'(h(x, Z), Y) \sigma'(w_s o_{l-1}) o_{l-1}. \end{aligned}$$

Given that $\sigma' \leq 1/4$, we have,

$$\|\nabla_{w_l} D(h(x, Z), Y)\|_\infty \leq \frac{M'}{4}. \tag{5.10}$$

Updates of Adam or Adagrad are bounded For any iteration $n \in \mathbb{N}^*$, we have for Adam $\alpha_n \leq \frac{\alpha}{\sqrt{1-\beta_2}}$ and for Adagrad $\alpha_n \leq \alpha$. We note $A = \max_{n\in\mathbb{N}^*} \alpha_n$. Besides, for any coordinate $i \in [d]$, we have $|m_{n,i}| \leq \sqrt{v_{n,i}}$, so that

$$\|x_n - x_{n-1}\|_\infty \leq A \tag{5.11}$$

Bound on w_l Let us assume that there exist $n_0 \in \mathbb{N}^*$ and a coordinate i corresponding to a weight of the last layer, such that $x_{n_0,i} \geq \frac{M'}{4\lambda} + A$. Given (5.10), we have,

$$\nabla_i f(x_{n_0}) \geq -\frac{M'}{4} + \lambda\frac{M'}{4\lambda} \geq 0.$$

Thus $m_{n_0,i} \geq 0$ and using (5.11),

$$0 \leq w_{n_0-1,i} - A \leq w_{n_0,i} \leq w_{n_0-1,i},$$

so that $|x_{n_0,i}| \leq x_{n_0-1,i}$. So if at any point $x_{n-1,i}$ goes over $\frac{M'}{4\lambda} + A$, the next iterates decrease until they go back below $\frac{M'}{4\lambda} + A$. Given that the maximum increase between two update is A, it means we have for any iteration $n \in \mathbb{N}^*$, and for any coordinate i corresponding to a weight of the last layer,

$$x_{n,i} \leq \frac{M'}{4\lambda} + 2A.$$

Applying the same technique we can show that $x_{n,i} \geq -\frac{M'}{4\lambda} - 2A$ and finally,

$$|x_{l,i}| \leq \frac{M'}{4\lambda} + 2A.$$

In particular, this implies that the Frobenius norm of the weight of the last layer w_l stays bounded for all the iterates.

5.4.2. Containment of the previous layers

Now taking a layer $s \in [l-1]$, we have,

$$\nabla_{w_s} D(h(x,Z),Y) = \\ D'(h(x,Z,Y)) \left(\prod_{k=l}^{s+1} \sigma'(o_{k-1}) w_k\right) o_{s-1}.$$

Let us assume we have shown that for layers $k > s$, $\|w_k\|_F \leq M_k$, then we can immediately derive that the above gradient is bounded in ℓ_∞ norm. Applying the same method as in 5.4.1, we can then show that the weights w_s stay bounded as well, with respect to the ℓ_∞ norm, by $\frac{M' \prod_{k>s} M_k}{4^{l-s+1}} + 2A$. Thus, by induction, we can show that the weights of all layers stay bounded for all iterations, albeit with a bound growing more than exponentially with depth.

5.5. Main results

For any total number of iterations $N \in \mathbb{N}^*$, we define τ_N a random index with value in $\{0, \dots, N-1\}$, verifying

$$\forall j \in \mathbb{N}, j < N, \mathbb{P}\left[\tau = j\right] \propto 1 - \beta_1^{N-j}. \tag{5.12}$$

If $\beta_1 = 0$, this is equivalent to sampling τ uniformly in $\{0, \dots, N-1\}$. If $\beta_1 > 0$, the last few $\frac{1}{1-\beta_1}$ iterations are sampled rarely, and all iterations older than a few times that number are sampled almost uniformly. All our results bound the expected squared norm of the total gradient at iteration τ, which is standard for non convex stochastic optimization [Ghadimi and Lan, 2013].

5.5.1. Convergence bounds

For simplicity, we first give convergence results for $\beta_1 = 0$, along with a complete proof in Section 5.6.1 and Section 5.6.2. We show convergence for any $\beta_1 < \beta_2$, however the theoretical bound is always worse than for $\beta_1 = 0$, while the proof becomes significantly more complex. Therefore, we delay the complete proof with momentum to the Appendix, Section C.5. We still provide the results with momentum in the second part of this section. Note that the disadvantageous dependency of the bound on β_1 is not specific to our proof but can be observed in previous adaptive methods bounds [Chen et al., 2019, Zou et al., 2019b].

Theorem 1 (Convergence of Adam without momentum)**.** *Given the assumptions introduced in Section 5.2.3, the iterates x_n defined in Section 5.2.2 with hyper-parameters verifying $0 < \beta_2 < 1$, $\alpha_n = \sqrt{\sum_{j=0}^{n-1} \beta_2^j}\alpha$ with $\alpha > 0$ and $\beta_1 = 0$, we have for any $N \in \mathbb{N}^*$, taking τ defined by* (5.12)*,*

$$\begin{aligned}\mathbb{E}\left[\|\nabla F(x_\tau)\|^2\right] \leq 2R\frac{F(x_0) - F_*}{\alpha N} + \\ C\left(\frac{1}{N}\ln\left(1 + \frac{R^2}{(1-\beta_2)\epsilon}\right) - \ln(\beta_2)\right),\end{aligned} \tag{5.13}$$

with

$$C = \frac{4dR^2}{\sqrt{1-\beta_2}} + \frac{\alpha dRL}{1-\beta_2}.$$

Theorem 2 (Convergence of Adagrad without momentum)**.** *Given the assumptions introduced in Section 5.2.3, the iterates x_n defined in Section 5.2.2 with hyper-parameters*

verifying $\beta_2 = 1$, $\alpha_n = \alpha$ *with* $\alpha > 0$ *and* $\beta_1 = 0$, *we have for any* $N \in \mathbb{N}^*$, *taking* τ *as defined by* (5.12),

$$\mathbb{E}\left[\|\nabla F(x_\tau)\|^2\right] \leq 2R\frac{F(x_0) - F_*}{\alpha\sqrt{N}} + \frac{1}{\sqrt{N}}\left(4dR^2 + \alpha dRL\right)\ln\left(1 + \frac{NR^2}{\epsilon}\right). \tag{5.14}$$

Theorem 3 (Convergence of Adam with momentum)**.** *Given the assumptions introduced in Section 5.2.3, the iterates* x_n *defined in Section 5.2.2 with hyper-parameters verifying* $0 < \beta_2 < 1$, $\alpha_n = (1-\beta_1)\sqrt{\sum_{j=0}^{n-1}\beta_2^j}\alpha$ *with* $\alpha > 0$ *and* $0 \leq \beta_1 < \beta_2$, *we have for any* $N \in \mathbb{N}^*$ *such that* $N > \frac{\beta_1}{1-\beta_1}$, *taking* τ *defined by* (5.12),

$$\mathbb{E}\left[\|\nabla F(x_\tau)\|^2\right] \leq 2R\frac{F(x_0) - F_*}{\alpha\tilde{N}} + C\left(\frac{1}{\tilde{N}}\ln\left(1 + \frac{R^2}{(1-\beta_2)\epsilon}\right) - \frac{N}{\tilde{N}}\ln(\beta_2)\right), \tag{5.15}$$

with

$$\tilde{N} = N - \frac{\beta_1}{1-\beta_1},$$

and,

$$C = \frac{\alpha dRL(1-\beta_1)}{(1-\beta_1/\beta_2)(1-\beta_2)} + \frac{12dR^2\sqrt{1-\beta_1}}{(1-\beta_1/\beta_2)^{3/2}\sqrt{1-\beta_2}} + \frac{2\alpha^2 dL^2\beta_1}{(1-\beta_1/\beta_2)(1-\beta_2)^{3/2}}. \tag{5.16}$$

Theorem 4 (Convergence of Adagrad with momentum)**.** *Given the assumptions introduced in Section 5.2.3, the iterates* x_n *defined in Section 5.2.2 with hyper-parameters verifying* $\beta_2 = 1$, $\alpha_n = \alpha$ *with* $\alpha > 0$ *and* $0 \leq \beta_1 < 1$, *we have for any* $N \in \mathbb{N}^*$ *such that* $N > \frac{\beta_1}{1-\beta_1}$, *taking* τ *as defined by* (5.12),

$$\mathbb{E}\left[\|\nabla F(x_\tau)\|^2\right] \leq 2R\sqrt{N}\frac{F(x_0) - F_*}{\alpha\tilde{N}} + \frac{\sqrt{N}}{\tilde{N}}C\ln\left(1 + \frac{NR^2}{\epsilon}\right), \tag{5.17}$$

with

$$\tilde{N} = N - \frac{\beta_1}{1-\beta_1}, \tag{5.18}$$

and,

$$C = \left(\alpha dRL + \frac{12dR^2}{1-\beta_1} + \frac{2\alpha^2 dL^2\beta_1}{1-\beta_1}\right). \tag{5.19}$$

5.5.2. Analysis of the bounds

Depencency in d. Looking at bounds introduced in the previous section, one can notice the presence of two terms: the forgetting of the initial condition, proportional to $F(x_0) - F_*$, and a second term that scales as d. The scaling as d is inevitable given our hypothesis, in particular the use of a bound on the ℓ_∞-norm of the gradients. Indeed, for any bound valid for a function F_1 with $d = 1$, then we can build a new function $F_d = \sum_{i \in [d]} F_1(x_i)$, i.e., we replicate d times the same optimization problem. The Hessian of F_d is diagonal with each diagonal element being the same as the Hessian of F_1, thus the smoothness constant is unchanged, nor is the ℓ_∞ bound on the stochastic gradients. Each dimension is independent from the other and equivalent to the single dimension problem given by F_1, thus $\mathbb{E}\left[\|\nabla F_d(x_\tau)\|_2^2\right]$ scales as d.

Almost sure bound on the gradient. We chose to assume the existence of an almost sure ℓ_∞-bound on the gradients given by (5.7). We use it only in (5.39) and (5.41). It is possible instead to use the Hölder inequality, which is the choice made by Ward et al. [2019] and Zou et al. [2019b]. This however deteriorate the bound, instead of a bound on $\mathbb{E}\left[\|\nabla F(x_\tau)\|_2^2\right]$, this would give a bound on $\mathbb{E}\left[\|\nabla F(x_\tau)\|_2^{4/3}\right]^{2/3}$. We also used the bound on the gradient in Lemma 5.1, to obtain (5.32) and (5.35), however in that case, a bound on the expected squared norm of the gradients is sufficient.

Impact of heavy-ball momentum. Looking at Theorems 3 and 4, we see that increasing β_1 always deteriorate the bound. Taking $\beta_1 = 0$ in those theorems gives us almost exactly the bound without heavy-ball momentum from Theorems 1 and 2, up to a factor 3 in the terms of the form dR^2. As discussed in the related work, Section 5.3, we significantly improve the dependency in $(1-\beta_1)^{-1}$, compared with previous work [Zou et al., 2019b,a]. We provide a more detailed analysis in the Appendix, Sections C.

5.5.3. Optimal finite horizon Adam is Adagrad

Let us take a closer look at the result from Theorem 1. It might seem like some quantities might explode but actually not for any reasonable values of α, β_2 and N. Let us assume $\epsilon \ll R$, $\alpha = N^{-a}$ and $\beta_2 = 1 - N^{-b}$. Then we immediately have

$$\frac{1}{RN}\sum_{n=1}^{N}\mathbb{E}\left[\|\nabla F(x)\|^2\right] \leq \frac{F(x_0) - F_*}{N^{1-a}} + C\left(\frac{1}{N}\ln\left(\frac{R^2 N^b}{\epsilon}\right) + N^{-b}\right), \tag{5.20}$$

with

$$C = dRN^{b/2} + \frac{LN^{b-a}}{2}. \tag{5.21}$$

Putting those together and ignoring the log terms for now,

$$\frac{1}{RN}\sum_{n=1}^{N}\mathbb{E}\left[\|\nabla F(x)\|^2\right] \lessapprox \frac{F(x_0)-F_*}{N^{1-a}} + dRN^{b/2-1} + dRN^{-b/2} + \frac{L}{2}N^{b-a-1} + \frac{L}{2}N^{-a}. \tag{5.22}$$

The best overall rate we can obtain is $O(1/\sqrt{N})$, and it is only achieved for $a = 1/2$ and $b = 1$, i.e., $\alpha = \alpha_1/\sqrt{N}$ and $\beta_2 = 1 - 1/N$. We can see the resemblance between Adagrad and Adam with a finite horizon and such parameters, as the exponential moving average for the denominator has a typical averaging window length of N. In particular, the bound for Adam now becomes

$$\frac{1}{RN}\sum_{n=1}^{N}\mathbb{E}\left[\|\nabla F(x)\|^2\right] \leq \frac{F(x_0)-F_*}{\alpha_1\sqrt{N}} + \frac{1}{\sqrt{N}}\left(dR + \frac{\alpha dL}{2}\right)\left(\ln\left(1+\frac{RN}{\epsilon}\right)+1\right), \tag{5.23}$$

which differ from (5.14) only by a +1 next to the log term.

Adam and Adagrad are twins. We discovered an important fact from the bounds we introduced in Section 5.5.1: Adam is to Adagrad like constant step size SGD is to decaying step size SGD. While Adagrad is asymptotically optimal, it has a slower forgetting of the initial condition $F(x_0) - F_*$, as $1/\sqrt{N}$ instead of $1/N$ for Adam. Furthermore, Adam adapts to local change of the smoothness faster than Adagrad as it eventually forgets about past gradients. This fast forgetting of the initial condition and improved adaptivity comes at a cost as Adam does not converge. It is however possible to chose parameters α and β_2 as to achieve an ϵ critical point for ϵ arbitrarily small and in particular, for a known time horizon, they can be chosen to obtain the exact same bound as Adagrad.

5.6. Proofs for $\beta_1 = 0$ (no momentum)

We assume here for simplicity that $\beta_1 = 0$, i.e., there is no heavy-ball style momentum. The recursions introduced in Section 5.2.2 can be simplified into

$$v_{n,i} = \beta_2 v_{n-1,i} + \left(\nabla_i f_n(x_{n-1})\right)^2 \tag{5.24}$$

$$x_{n,i} = x_{n-1,i} - \alpha_n \frac{\nabla_i f_n(x_{n-1})}{\sqrt{\epsilon + v_{n,i}}}. \tag{5.25}$$

Throughout the proof we note by $\mathbb{E}_{n-1}[\cdot]$ the conditional expectation with respect to $f_1, \dots, f_{n-1}$. In particular, x_{n-1}, v_{n-1} is deterministic knowing $f_1, \dots, f_{n-1}$. For all $n \in \mathbb{N}^*$, we also define $\tilde{v}_n \in \mathbb{R}^d$ so that for all $i \in [d]$,

$$\tilde{v}_{n,i} = \beta_2 v_{n-1,i} + \mathbb{E}_{n-1}\left[(\nabla_i f_n(x_n))^2\right], \tag{5.26}$$

i.e., $\tilde{v}_n$ is obtained from v_n by replacing the last gradient contribution by its expected value knowing $f_1, \dots, f_{n-1}$.

5.6.1. Technical lemmas

A problem posed by the update in (5.25) is the correlation between the numerator and denominator. This prevents us from easily computing the conditional expectation and as noted by Reddi et al. [2019], the expected direction of update can have a positive dot product with the objective gradient. It is however possible to control the deviation from the descent direction, following Ward et al. [2019] with this first lemma.

Lemma 5.1 (adaptive update approximately follow a descent direction)**.** *For all $n \in \mathbb{N}^*$ and $i \in [d]$, we have:*

$$\mathbb{E}_{n-1}\left[\nabla_i F(x_{n-1}) \frac{\nabla_i f_n(x_{n-1})}{\sqrt{\epsilon + v_{n,i}}}\right] \geq \frac{(\nabla_i F(x_{n-1}))^2}{2\sqrt{\epsilon + \tilde{v}_{n,i}}} - 2R\mathbb{E}_{n-1}\left[\frac{(\nabla_i f_n(x_{n-1}))^2}{\epsilon + v_{n,i}}\right]. \tag{5.27}$$

Proof. We take $i \in [d]$ and note $G = \nabla_i F(x_{n-1})$, $g = \nabla_i f_n(x_{n-1})$, $v = v_{n,i}$ and $\tilde{v} = \tilde{v}_{n,i}$.

$$\mathbb{E}_{n-1}\left[\frac{Gg}{\sqrt{\epsilon + v}}\right] = \mathbb{E}_{n-1}\left[\frac{Gg}{\sqrt{\epsilon + \tilde{v}}}\right] + \mathbb{E}_{n-1}\Bigg[\underbrace{Gg\left(\frac{1}{\sqrt{\epsilon + v}} - \frac{1}{\sqrt{\epsilon + \tilde{v}}}\right)}_{A}\Bigg]. \tag{5.28}$$

Given that g and $\tilde{v}$ are independent given $f_1, \dots, f_{n-1}$, we immediately have

$$\mathbb{E}_{n-1}\left[\frac{Gg}{\sqrt{\epsilon + \tilde{v}}}\right] = \frac{G^2}{\sqrt{\epsilon + \tilde{v}}}. \tag{5.29}$$

Now we need to control the size of A,

$$\begin{aligned} A &= Gg\frac{\tilde{v}-v}{\sqrt{\epsilon+v}\sqrt{\epsilon+\tilde{v}}(\sqrt{\epsilon+v}+\sqrt{\epsilon+\tilde{v}})} \\ &= Gg\frac{\mathbb{E}_{n-1}\left[g^2\right]-g^2}{\sqrt{\epsilon+v}\sqrt{\epsilon+\tilde{v}}(\sqrt{\epsilon+v}+\sqrt{\epsilon+\tilde{v}})} \\ |A| &\leq \underbrace{|Gg|\frac{\mathbb{E}_{n-1}\left[g^2\right]}{\sqrt{\epsilon+v}(\epsilon+\tilde{v})}}_{\kappa} + \underbrace{|Gg|\frac{g^2}{(\epsilon+v)\sqrt{\epsilon+\tilde{v}}}}_{\rho}, \end{aligned}$$

the last inequality coming from the fact that $\sqrt{\epsilon+v}+\sqrt{\epsilon+\tilde{v}} \geq \max(\sqrt{\epsilon+v}, \sqrt{\epsilon+\tilde{v}})$ and $\left|\mathbb{E}_{n-1}\left[g^2\right]-g^2\right| \leq \mathbb{E}_{n-1}\left[g^2\right]+g^2$.

Following Ward et al. [2019], we can use the following inequality to bound κ and ρ,

$$\forall \lambda > 0,\, x, y \in \mathbb{R}, xy \leq \frac{\lambda}{2}x^2 + \frac{y^2}{2\lambda}. \tag{5.30}$$

First applying (5.30) to κ with

$$\lambda = \frac{\sqrt{\epsilon+\tilde{v}}}{2},\; x = \frac{|G|}{\sqrt{\epsilon+\tilde{v}}},\; y = \frac{|g|\,\mathbb{E}_{n-1}\left[g^2\right]}{\sqrt{\epsilon+\tilde{v}}\sqrt{\epsilon+v}},$$

we obtain

$$\kappa \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{v}}} + \frac{g^2\mathbb{E}_{n-1}\left[g^2\right]^2}{(\epsilon+\tilde{v})^{3/2}(\epsilon+v)}.$$

Given that $\epsilon+\tilde{v} \geq \mathbb{E}_{n-1}\left[g^2\right]$ and taking the conditional expectation, we can simplify as

$$\mathbb{E}_{n-1}\left[\kappa\right] \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{v}}} + \frac{\mathbb{E}_{n-1}\left[g^2\right]}{\sqrt{\epsilon+\tilde{v}}}\mathbb{E}_{n-1}\left[\frac{g^2}{\epsilon+v}\right]. \tag{5.31}$$

Given that $\sqrt{\mathbb{E}_{n-1}\left[g^2\right]} \leq \sqrt{\epsilon+\tilde{v}}$ and $\sqrt{\mathbb{E}_{n-1}\left[g^2\right]} \leq R$, we can simplify (5.31) as

$$\mathbb{E}_{n-1}\left[\kappa\right] \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{v}}} + R\mathbb{E}_{n-1}\left[\frac{g^2}{\epsilon+v}\right]. \tag{5.32}$$

Now turning to ρ, we use (5.30) with

$$\lambda = \frac{\sqrt{\epsilon+\tilde{v}}}{2\mathbb{E}_{n-1}\left[g^2\right]},\; x = \frac{|Gg|}{\sqrt{\epsilon+\tilde{v}}},\; y = \frac{g^2}{\epsilon+v}, \tag{5.33}$$

we obtain

$$\rho \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{v}}}\frac{g^2}{\mathbb{E}_{n-1}\left[g^2\right]} + \frac{\mathbb{E}_{n-1}\left[g^2\right]}{\sqrt{\epsilon+\tilde{v}}}\frac{g^4}{(\epsilon+v)^2},$$

Given that $\epsilon + v \geq g^2$ and taking the conditional expectation we obtain

$$\mathbb{E}_{n-1}\left[\rho\right] \leq \frac{G^2}{4\sqrt{\epsilon + \tilde{v}}} + \frac{\mathbb{E}_{n-1}\left[g^2\right]}{\sqrt{\epsilon + \tilde{v}}} \mathbb{E}_{n-1}\left[\frac{g^2}{\epsilon + v}\right], \tag{5.34}$$

which we simplify using the same argument as for (5.32) into

$$\mathbb{E}_{n-1}\left[\rho\right] \leq \frac{G^2}{4\sqrt{\epsilon + \tilde{v}}} + R\mathbb{E}_{n-1}\left[\frac{g^2}{\epsilon + v}\right]. \tag{5.35}$$

Notice that in 5.33, we possibly divide by zero. It suffice to notice that if $\mathbb{E}_{n-1}\left[g^2\right] = 0$ then $g^2 = 0$ a.s. so that $\rho = 0$ and (5.35) is still verified.

Summing (5.32) and (5.35) we can bound

$$\mathbb{E}_{n-1}\left[|A|\right] \leq \frac{G^2}{2\sqrt{\epsilon + \tilde{v}}} + 2R\mathbb{E}_{n-1}\left[\frac{g^2}{\epsilon + v}\right]. \tag{5.36}$$

Injecting (5.36) and (5.29) into (5.28) finishes the proof. □

Anticipating on Section 5.6.2, we can see that for a coordinate $i \in [d]$ and iteration $n \in \mathbb{N}^*$, the deviation from a descent direction is at most

$$2R\mathbb{E}_{n-1}\left[\frac{\left(\nabla_i f_n(x_{n-1})\right)^2}{\epsilon + v_{n,i}}\right].$$

While for any specific iteration, this deviation can take us away from a descent direction, the next lemma tells us that when we sum those deviations over all iterations, it cannot grow larger than a logarithmic term. This key insight introduced by Ward et al. [2019] is what makes the proof work.

Lemma 5.2 (sum of ratios with the denominator increasing as the numerator)**.** *We assume we have $0 < \beta_2 \leq 1$ and a non-negative sequence $(a_n)_{n\in\mathbb{N}^*}$. We define $b_n = \sum_{j=1}^{n} \beta_2^{n-j} a_j$ with the convention $b_0 = 0$. Then we have,*

$$\sum_{j=1}^{N} \frac{a_j}{\epsilon + b_j} \leq \ln\left(1 + \frac{b_N}{\epsilon}\right) - N\ln(\beta_2). \tag{5.37}$$

Proof. Given that concavity of ln, and the fact that $b_j > a_j$, we have for all $j \in \mathbb{N}^*$,

$$\begin{aligned}
\frac{a_j}{\epsilon + b_j} &\leq \ln(\epsilon + b_j) - \ln(\epsilon + b_j - a_j) \\
&= \ln(\epsilon + b_j) - \ln(\epsilon + \beta_2 b_{j-1}) \\
&= \ln\left(\frac{\epsilon + b_j}{\epsilon + b_{j-1}}\right) + \ln\left(\frac{\epsilon + b_{j-1}}{\epsilon + \beta_2 b_{j-1}}\right).
\end{aligned}$$

The first term on the right hand side forms a telescoping series, while the last term is bounded by $-\ln(\beta)$ as $\epsilon \geq \beta_2\epsilon$. increasing with b_{j-1} and thus is bounded by $-\ln(\beta_2)$. Summing over all $j \in [N]$ gives the desired result. □

5.6.2. Proof of Adam and Adagrad without momentum

For all iterations $n \in \mathbb{N}^*$, we define the update $u_n \in \mathbb{R}^d$,

$$\forall i \in [d], u_{n,i} = \frac{\nabla_i f_n(x_{n-1})}{\sqrt{\epsilon + v_{n,i}}}. \tag{5.38}$$

Adam Let us a take an iteration $n \in \mathbb{N}^*$. We note $\alpha_n = \alpha\sqrt{\sum_{j=0}^{n} \beta_2^j}$ (see (5.4) in Section 5.2.2. Using the smoothness of F defined in (5.8), we have

$$F(x_n) \leq F(x_{n-1}) - \alpha_n \nabla F(x_{n-1})^T u_n + \frac{\alpha_n^2 L}{2} \|u_n\|_2^2 .$$

Notice that due to the a.s. ℓ_∞ bound on the gradients (5.7), we have for any $i \in [d]$, $\sqrt{\epsilon + \tilde{v}_{n,i}} \leq R\sqrt{\sum_{j=0}^{n-1} \beta_2^j}$, so that,

$$\alpha_n \frac{(\nabla_i F(x_{n-1}))^2}{2\sqrt{\epsilon + \tilde{v}_{n,i}}} \geq \frac{\alpha \left(\nabla_i F(x_{n-1})\right)^2}{R}. \tag{5.39}$$

Taking the conditional expectation with respect to $f_0, \ldots, f_{n-1}$ we can apply the descent Lemma 5.1 and use (5.39) to obtain,

$$\begin{aligned}\mathbb{E}_{n-1}\left[F(x_n)\right] \leq F(x_{n-1}) &- \frac{\alpha}{2R} \|\nabla F(x_{n-1})\|_2^2 \\ &+ \left(2\alpha_n R + \frac{\alpha_n^2 L}{2}\right) \mathbb{E}_{n-1}\left[\|u_n\|_2^2\right].\end{aligned}$$

Given that $\beta_2 < 1$, we have $\alpha_n \leq \frac{\alpha}{\sqrt{1-\beta_2}}$. Summing the previous inequality for all $n \in [N]$ and taking the complete expectation yields

$$\begin{aligned}\mathbb{E}\left[F(x_N)\right] \leq F(x_0) &- \frac{\alpha}{2R} \sum_{n=0}^{N-1} \mathbb{E}\left[\|\nabla F(x_n)\|_2^2\right] \\ &+ \left(\frac{2\alpha R}{\sqrt{1-\beta_2}} + \frac{\alpha^2 L}{2(1-\beta_2)}\right) \sum_{n=0}^{N-1} \mathbb{E}\left[\|u_n\|_2^2\right].\end{aligned} \tag{5.40}$$

The application of Lemma 5.2 immediately gives for all $i \in [d]$,

$$\mathbb{E}\left[\sum_{n=0}^{N-1} u_{n,i}^2\right] \leq \ln\left(1 + \frac{R^2}{(1-\beta)\epsilon}\right) - N\ln(\beta).$$

Injecting into (5.40) and rearranging the terms, the result of Theorem 1 follows immediately.

Adagrad. Let us now take $\alpha_n = \alpha$ and β_2 to recover Adagrad. Using again the smoothness of F defined in (5.8), we have

$$F(x_{n+1}) \leq F(x_n) - \alpha \nabla F(x_n)^T u_n. + \frac{\alpha^2 L}{2} \|u_n\|_2^2 .$$

Notice that due to the a.s. ℓ_∞ bound on the gradients (5.7), we have for any $i \in [d]$, $\sqrt{\epsilon + \tilde{v}_{n,i}} \leq R\sqrt{n}$, so that,

$$\alpha \frac{(\nabla_i F(x_{n-1}))^2}{2\sqrt{\epsilon + \tilde{v}_{n,i}}} \geq \frac{\alpha \left(\nabla_i F(x_{n-1})\right)^2}{2R\sqrt{n}}. \tag{5.41}$$

Taking the conditional expectation with respect to $f_0, \ldots, f_{n-1}$ we can apply the descent Lemma 5.1 and use (5.41) to obtain,

$$\begin{aligned}\mathbb{E}_{n-1}\left[F(x_n)\right] \leq F(x_{n-1}) - \frac{\alpha}{2R} \|\nabla F(x_{n-1})\|_2^2 \\ + \left(2\alpha R + \frac{\alpha^2 L}{2}\right) \mathbb{E}_{n-1}\left[\|u_n\|_2^2\right].\end{aligned}$$

Summing the previous inequality for all $n \in [N]$, taking the complete expectation, and using that $\sqrt{n} \leq \sqrt{N}$ gives us,

$$\begin{aligned}\mathbb{E}\left[F(x_N)\right] \leq F(x_0) - \frac{\alpha}{2R\sqrt{N}} \sum_{n=0}^{N-1} \mathbb{E}\left[\|\nabla F(x_n)\|_2^2\right] \\ + \left(2\alpha R + \frac{\alpha^2 L}{2}\right) \sum_{n=0}^{N-1} \mathbb{E}\left[\|u_n\|_2^2\right].\end{aligned} \tag{5.42}$$

The application of Lemma 5.2 immediately gives for all $i \in [d]$,

$$\mathbb{E}\left[\sum_{n=0}^{N-1} u_{n,i}^2\right] \leq \ln\left(1 + \frac{R^2 N}{\epsilon}\right).$$

Injecting into (5.42) and rearranging the terms, the result of Theorem 2 follows immediately.

5.7. Conclusion

We provided a simple proof on the convergence of Adam and Adagrad without heavy-ball style momentum. The extension to include heavy-ball momentum is slightly more complex, but our approach leads to simpler proofs than previous ones, while significantly improving the dependence on the momentum parameter. The bounds clarify the important parameters for the convergence of Adam. A main practical takeaway, increasing the

exponential decay factor is as critical as decreasing the learning rate for converging to a critical point. Our analysis also highlights a link between Adam and a finite-horizon version of Adam: for fixed N, taking $\alpha = 1/\sqrt{N}$ and $\beta_2 = 1 - 1/N$ for Adam gives the same convergence bound as Adagrad.

6. Future directions

Audio Since the redaction of the present manuscript, we adapted the Demucs architecture to the task of real time noise removal in speech recordings. It achieves state-of-the-art performance on real life recordings, with a model of 15MB able to enhance 16ms of audio in 13ms of processing time on an average laptop CPU. Those results are additional evidence that our approach can be used for real life applications thanks to its computational efficiency. However, we are limited by the "noise + noise" problem: when two sources contains noise-like stochastic textures at the same time, it is not possible to separate them (except on the train set where the model can overfit) using a regression loss over the waveform.

This problem could be solved with more complex losses on spectrograms, taking into account local statistics rather than doing only pointwise regression, as already done by Caracalla and Roebel [2017] for audio texture synthesis. We also noticed that the strided transposed convolutions tend to favor some frequencies within the noise (those with a wavelength that is proportional to the stride), so that the synthesized noise clearly sounds "artificial". In order to restore the balance of frequencies, a potential solution would be to use parallel layers with different strides. Intuitively, the current architecture works like a single guitar string with few frets. We would like instead to have multiple strings with different base lengths and more frets so that the model can recreate any frequency easily. This new architecture could perform better separation when many stochastic textures overlap, which is the case for the original "cocktail party problem" with a large number of guests. Such a model could also be used for fast speech synthesis on CPU. While it is already possible to do so[1] it is currently only possible for devices with a power supply, and using all available cores. A lighter approach could generalize high quality speech synthesis to many more devices and free up resources for other complementary tasks.

Optimization There are two directions to pursue regarding convex and non convex optimization. First, the bounds we present in Chapter 5 worsen when using heavy-ball

[1] https://ai.facebook.com/blog/a-highly-efficient-real-time-text-to-speech-[...]-on-cpus/

momentum, which contradicts practical evidences. It is possible however that real life problems have enough regularity which makes momentum work well. An interesting question to study is whether there exist a worst case problem on which momentum degrades performance. A second question is why it helps on practical problems, and whether it also impacts the generalization of the model.

A second interesting direction would be to extend the Adabatch approach presented in Chapter 4 to more complex "reliable" estimates of the Hessian. Indeed, we have seen that Adabatch performs a reconditioning of the problem by estimating the contribution to the Hessian that is coming from the sparsity of the problem. This estimate is always stable, unlike with generic stochastic second order methods. In Chapter 5, we have seen that we need the objective function to be uniformly L smooth. This is not verified by deep neural networks: for a simple feed-forward network, the smoothness with respect to one layer will roughly grow as the product of the norms of the other layers. An interesting question is whether this knowledge on the structure of the model can be used to derive a stable pre-conditioner that speeds up performance. This pre-conditioner could also help provide convergence bounds without requiring uniform smoothness when a proxy for the local smoothness is available.

Bibliography

Alekh Agarwal, Martin J Wainwright, Peter L Bartlett, and Pradeep K Ravikumar. Information-theoretic lower bounds on the oracle complexity of convex optimization. In *Advances in Neural Information Processing Systems*, 2009.

Manu Airaksinen. Analysis/synthesis comparison of vocoders utilized in statistical parametric speech synthesis. Master's thesis, Aalto University, 2012.

Jonathan Allen. Short term spectral analysis, synthesis, and modification by discrete fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25 (3):235–238, 1977.

Shunichi Amari. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, EC-16(3):299–307, 1967.

Joakim Andén and Stéphane Mallat. Deep scattering spectrum. *IEEE Transactions on Signal Processing*, 62(16):4114–4128, 2014.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. Preprint on arXiv:1607.06450, 2016.

Francis Bach and Eric Moulines. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, 2011.

Francis Bach and Eric Moulines. Non-strongly-convex smooth stochastic approximation with convergence rate o(1/n). In *Advances in Neural Information Processing Systems*, 2013.

Francis R Bach and Michael I Jordan. Blind one-microphone speech separation: A spectral learning approach. In *Advances in Neural Information Processing Systems*, 2005.

Anthony J Bell and Terrence J Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural computation*, 7(6):1129–1159, 1995.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems 28*, 2015.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Julius R Blum. Multidimensional stochastic approximation methods. *The Annals of Mathematical Statistics*, pages 737–744, 1954.

Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6(Sep):1579–1619, 2005.

Léon Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8):12, 1991.

Léon Bottou. *On-Line Learning and Stochastic Approximations*, chapter 2. Cambridge University Press, 1999.

Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, 2008.

Léon Bottou and Yann Le Cun. On-line learning for very large data sets. *Applied stochastic models in business and industry*, 21(2):137–151, 2005.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

A. S. Bregman. *Auditory Scene Analysis*. MIT Press, Cambridge, MA, 1990.

Sébastien Bubeck et al. *Convex optimization: Algorithms and complexity*. Now Publishers, Inc., 2015.

Michael Buhrmester, Tracy Kwang, and Samuel D Gosling. Amazon's mechanical turk: A new source of inexpensive, yet high-quality, data? *Perspectives on psychological science*, 6(1):3–5, 2011.

Hugo Caracalla and Axel Roebel. Gradient conversion between time and frequency domains using wirtinger calculus. In *International conference on digital audio effects (DAFx)*, 2017.

Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

Xiangyi Chen, Sijia Liu, Ruoyu Sun, and Mingyi Hong. On the convergence of a class of Adam-type algorithms for non-convex optimization. In *International Conference on Learning Representations*, 2019.

E. Colin Cherry. Some experiments on the recognition of speech, with one and with two ears. *The Journal of the Acoustic Society of America*, 1953.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *IEEE conference on computer vision and pattern recognition*, 2017.

Alice Cohen-Hadria, Axel Roebel, and Geoffroy Peeters. Improving singing voice separation using deep u-net and wave-u-net with data augmentation. In *27th European Signal Processing Conference (EUSIPCO)*. IEEE, 2019.

Pierre Comon. Independent component analysis, a new concept? *Signal processing*, 36 (3):287–314, 1994.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International Conference on Machine Learning*, 2017.

Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4):357–366, 1980.

Laurent De Wilde. *Les fous du son.* Grasset, 2016.

Aaron Defazio and Léon Bottou. On the ineffectiveness of variance reduced optimization for deep learning. In *Advances in Neural Information Processing Systems*, 2019.

Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, 2014.

Alexandre Défossez and Francis Bach. Averaged least-mean-squares: Bias-variance trade-offs and optimal sampling distributions. In *International Conference on Artificial Intelligence and Statistics (AI Stats)*, 2015.

Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 2012.

Timothy Dozat. Incorporating nesterov momentum into adam. Technical report, Stanford, 2016.

Yoel Drori and Ohad Shamir. The complexity of finding stationary points with stochastic gradient descent. Preprint on arXiv:1910.01845, 2019.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul), 2011.

John Duchi, Michael I Jordan, and Brendan McMahan. Estimation, optimization, and parallelism when data is sparse. In *Advances in Neural Information Processing Systems*, 2013.

Kemal Ebcioğlu. An expert system for harmonizing four-part chorales. *Computer Music Journal*, 12(3):43–51, 1988.

Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. Neural audio synthesis of musical notes with wavenet autoencoders. In *34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017.

Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. Preprint on arXiv:1902.08710, 2019.

Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. Ddsp: Differentiable digital signal processing. In *International Conference on Learning Representations*, 2020.

Gunnar Fant. *Acoustic theory of speech production*. Walter de Gruyter, 1970.

James L Flanagan and RM Golden. Phase vocoder. *Bell System Technical Journal*, 45 (9):1493–1509, 1966.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. Preprint on arXiv:1705.03122, 2017.

Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4), 2013.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AI Stats)*, 2010.

JL Goldstein. Auditory nonlinearity. *The Journal of the Acoustical Society of America*, 41(3):676–699, 1967.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Emad M. Grais, Mehmet Umut Sen, and Hakan Erdogan. Deep neural networks for single channel source separation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014.

Daniel Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243, 1984.

Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation. Preprint on arXiv:1612.01010, 2016.

Albert Haque, Michelle Guo, and Prateek Verma. Conditional end-to-end audio transforms. Preprint on arXiv:1804.00047, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE international conference on computer vision*, 2015.

Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. Technical Report 2002.05651, arXiv, 2020.

D. Herremans and E. Chew. Morpheus: automatic music generation with recurrent pattern constraints and tension profiles. In *of the IEEE Region 10 Conference (TENCON)*, 2016.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. Preprint on arXiv:1503.02531, 2015.

Yu Chi Ho. On the stochastic approximation method and optimal filtering theory. *Journal of mathematical analysis and applications*, 6(1):152–154, 1963.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. Preprint on arXiv:1704.04861, 2017.

Cho-Jui Hsieh, Hsiang-Fu Yu, and Inderjit Dhillon. Passcode: Parallel asynchronous stochastic dual co-ordinate descent. In *International Conference on Machine Learning*, 2015.

Qiong Hu, Korin Richmond, Junichi Yamagishi, and Javier Latorre. An experimental comparison of multiple vocoder types. In *ISCA Workshop on Speech Synthesis*, 2013.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *IEEE conference on computer vision and pattern recognition*, 2017.

Po-Sen Huang, Minje Kim, Mark Hasegawa-Johnson, and Paris Smaragdis. Joint optimization of masks and deep recurrent neural networks for monaural source separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(12):2136–2147, 2015.

Andrew J Hunt and Alan W Black. Unit selection in a concatenative speech synthesis system using a large speech database. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, volume 1. IEEE, 1996.

Aapo Hyvarinen. Survey on independent component analysis. *Neural computing surveys*, 2(4):94–128, 1999.

Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*. John Wiley & Sons, 2004.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. Preprint on arXiv:1502.03167, 2015.

Yusuf Isik, Jonathan Le Roux, Zhuo Chen, Shinji Watanabe, and John R Hershey. Single-channel multi-speaker separation using deep clustering. Preprint on arXiv:1607.02173, 2016.

Fumitada Itakura. Analysis synthesis telephony based on the maximum likelihood method. In *6th international congress on acoustics, 1968*, 1968.

P. Jain, S. M. Kakade, R. Kidambi, P. Netrapalli, and A. Sidford. Parallelizing stochastic approximation through mini-batching and tail-averaging. Preprint on arXiv:1610.03774, 2016.

Andreas Jansson, Eric Humphrey, Nicola Montecchio, Rachel Bittner, Aparna Kumar, and Tillman Weyde. Singing voice separation with deep u-net convolutional networks. In *International Society for Music Information Retrieval (ISMIR) conference*, 2017.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, 2013.

Christian Jutten and Jeanny Herault. Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture. *Signal processing*, 24(1):1–10, 1991.

Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. Preprint on arXiv:1802.08435, 2018.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. Preprint on arXiv:1710.10196, 2017.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. Preprint on arXiv:1812.04948, 2018.

RL Kashyap and CC Blaydon. Recovery of functions from noisy measurements taken at randomly selected points and its application to pattern classification. *Proceedings of the IEEE*, 54(8), 1966.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. Preprint on arXiv:1412.6980, 2014.

Morten Kolbæk, Dong Yu, Zheng-Hua Tan, Jesper Jensen, Morten Kolbaek, Dong Yu, Zheng-Hua Tan, and Jesper Jensen. Multitalker speech separation with utterance-level permutation invariant training of deep recurrent neural networks. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 2017.

Simon Lacoste-Julien, Mark Schmidt, and Francis Bach. A simpler approach to obtaining an o (1/t) convergence rate for the projected stochastic subgradient method. Technical Report 1212.2002, arXiv, 2012.

Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. Preprint on arXiv:1806.07297, 2018.

Jean Laroche and Mark Dolson. Phase-vocoder: About this phasiness business. In *Workshop on Applications of Signal Processing to Audio and Acoustics*. IEEE, 1997.

R. Leblond, F. Pedregosa, and S. Lacoste-Julien. Asaga: Asynchronous parallel saga. In *International Conference on Artificial Intelligence and Statistics (AI Stats)*, 2017.

Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, 2001.

Jason D Lee, Max Simchowitz, Michael I Jordan, and Benjamin Recht. Gradient descent converges to minimizers. Preprint on arXiv:1602.04915, 2016.

Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. Efficient mini-batch training for stochastic optimization. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

Xiaoyu Li and Francesco Orabona. On the convergence of stochastic gradient descent with adaptive stepsizes. In *International Conference on Artificial Intelligence and Statistics (AI Stats)*, 2019.

Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. Preprint on arXiv:1712.09203, 2017.

Jen-Yu Liu and Yi-Hsuan Yang. Denoising auto-encoder with recurrent skip connections and residual regression for music source separation. In *17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018.

Francesc Lluís, Jordi Pons, and Xavier Serra. End-to-end music source separation: is it possible in the waveform domain? Preprint on arXiv:1810.12187, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Yi Luo and Nima Mesgarani. Tasnet: time-domain audio separation network for real-time, single-channel speech separation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

Yi Luo and Nima Mesgarani. Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2019.

Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I. Jordan, Peter Richtárik, and Martin Takác. Adding vs. averaging in distributed primal-dual optimization. In *International Conference on Machine Learning*, 2015.

Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Identifying suspicious urls: An application of large-scale online learning. In *International Conference on Machine Learning*, 2009.

Neil A Macmillan and C Douglas Creelman. *Detection theory: A user's guide*. Psychology press, 2004.

Stephane Mallat. A wavelet tour of signal processing. *The Sparse Way*, 1999.

H. Mania, X. Pan, D. Papailiopoulos, B. Recht, K. Ramchandran, and M. I. Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. Preprint on arXiv:1507.06970, 2015.

H Brendan McMahan and Matthew Streeter. Adaptive bound optimization for online convex optimization. In *Conference On Learning Theory (COLT)*, 2010.

H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. Ad click prediction: a view from the trenches. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013.

Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model. Preprint on arXiv:1612.07837, 2016.

Paul Mermelstein. Distance measures for speech recognition, psychological and instrumental. *Pattern recognition and artificial intelligence*, 116:374–388, 1976.

Masanori Morise, Fumiya Yokomori, and Kenji Ozawa. World: a vocoder-based high-quality speech synthesis system for real-time applications. *IEICE TRANSACTIONS on Information and Systems*, 99(7):1877–1884, 2016.

Eric Moulines and Jean Laroche. Non-parametric techniques for pitch-scale and time-scale modification of speech. *Speech communication*, 16(2):175–205, 1995.

Eliya Nachmani and Lior Wolf. Unsupervised singing voice conversion. Preprint on arXiv:1904.06590, 2019.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. Preprint on arXiv:1912.02292, 2019.

D. Needell and R. Ward. Batched stochastic gradient descent with weighted sampling. Preprint on arXiv:1608.07641, 2016.

Deanna Needell, Nathan Srebro, and Rachel Ward. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, 2014.

Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

Yu Nesterov and J-Ph Vial. Confidence level solutions for stochastic programming. *Automatica*, 2008.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, 2017.

Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems*, 2011.

Aditya Arie Nugraha, Antoine Liutkus, and Emmanuel Vincent. Multichannel music separation with deep neural networks. In *24th European Signal Processing Conference (EUSIPCO)*. IEEE, 2016.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. Preprint on arXiv:1609.03499, 2016.

Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C Cobo, Florian Stimberg, et al. Parallel wavenet: Fast high-fidelity speech synthesis. Preprint on arXiv:1711.10433, 2017.

Pentti Paatero. Least squares formulation of robust non-negative factor analysis. *Chemometrics and intelligent laboratory systems*, 37(1):23–35, 1997.

X. Pan, M. Lam, S. Tu, D. Papailiopoulos, C. Zhang, M. I. Jordan, K. Ramchandran, C. Re, and B. Recht. Cyclades: Conflict-free asynchronous machine learning. In *Advances in Neural Information Processing Systems*, 2016.

Wei Ping, Kainan Peng, Andrew Gibiansky, S Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John Miller. Deep voice 3: Scaling text-to-speech with convolutional sequence learning. In *6th International Conference on Learning Representations*, 2018.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5), 1964.

Michael Portnoff. Time-scale modification of speech based on short-time fourier analysis. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(3):374–390, 1981.

Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019.

Zafar Rafii, Antoine Liutkus, Fabian-Robert Stöter, Stylianos Ioannis Mimilakis, and Rachel Bittner. The musdb18 corpus for music separation, 2017.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, 2011.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of Adam and beyond. Preprint on arXiv:1904.09237, 2019.

Aarthi M Reddy and Bhiksha Raj. Soft mask methods for single-channel speaker separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(6):1766–1776, 2007.

Dario Rethage, Jordi Pons, and Xavier Serra. A wavenet for speech denoising. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

Flávio Ribeiro, Dinei Florêncio, Cha Zhang, and Michael Seltzer. Crowdmos: An approach for crowdsourcing mean opinion score studies. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2011.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Axel Roebel. Transient detection and preservation in the phase vocoder. In *International Computer Music Conference (ICMC)*, 2003.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, 2015.

Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

Nicolas Le Roux, Pierre-Antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In *Advances in Neural Information Processing Systems*, 2008.

Sam T Roweis. One microphone source separation. In *Advances in Neural Information Processing Systems*, 2001.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

Daniel Schobben, Kari Torkkola, and Paris Smaragdis. Evaluation of blind signal separation methods. In *International Workshop on Independent Component Analysis*, volume 99, 1999.

Stephanie Seneff. System to independently modify excitation and/or spectrum of speech waveform without explicit pitch extraction. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 30(4):566–578, 1982.

Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(Feb):567–599, 2013.

Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

Jonathan Shen, Ruoming Pang, Ron J Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, et al. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018.

Brian K Shepard. *Refining sound: A practical guide to synthesis and synthesizers*. Oxford University Press, 2013.

Paris Smaragdis. Non-negative matrix factor deconvolution; extraction of multiple sound sources from monophonic inputs. In *International Conference on Independent Component Analysis and Signal Separation*. Springer, 2004.

Paris Smaragdis, Cedric Fevotte, Gautham J Mysore, Nasser Mohammadiha, and Matthew Hoffman. Static and dynamic source separation using nonnegative factorizations: A unified view. *IEEE Signal Processing Magazine*, 31(3):66–75, 2014.

Julius Orion Smith. *Introduction to digital filters: with audio applications*, volume 2. Julius Smith, 2007.

Juliu O Smith III. *Spectral audio signal processing*. W3K publishing, 2011.

Jose Sotelo, Soroush Mehri, Kundan Kumar, Joao Felipe Santos, Kyle Kastner, Aaron Courville, and Yoshua Bengio. Char2wav: End-to-end speech synthesis. In *Workshop of the International Conference on Learning Representation*, 2017.

Daniel Stoller, Sebastian Ewert, and Simon Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation. Preprint on arXiv:1806.03185, 2018.

F.-R. Stöter, S. Uhlich, A. Liutkus, and Y. Mitsufuji. Open-unmix - a reference implementation for music source separation. *Journal of Open Source Software*, 2019.

Fabian-Robert Stöter, Antoine Liutkus, and Nobutaka Ito. The 2018 signal separation evaluation campaign. In *International Conference on Latent Variable Analysis and Signal Separation*. Springer, 2018.

Nicolas Sturmel, Laurent Daudet, et al. Signal reconstruction from stft magnitude: A state of the art. In *International conference on digital audio effects (DAFx)*, 2011.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in Neural Information Processing Systems*, 2015.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, 2013.

Yaniv Taigman, Lior Wolf, Adam Polyak, and Eliya Nachmani. Voice synthesis for in-the-wild speakers via a phonological loop. Preprint on arXiv:1707.06588, 2017.

Naoya Takahashi and Yuki Mitsufuji. Multi-scale multi-band densenets for audio source separation. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE, 2017.

Naoya Takahashi, Nabarun Goswami, and Yuki Mitsufuji. Mmdenselstm: An efficient combination of convolutional and recurrent neural networks for audio source separation. In *16th International Workshop on Acoustic Signal Enhancement (IWAENC)*. IEEE, 2018.

T. Tieleman and G. Hinton. Lecture 6.5 — rmsprop. COURSERA: Neural Networks for Machine Learning, 2012.

Stefan Uhlich, Franck Giron, and Yuki Mitsufuji. Deep neural network based instrument extraction from music. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015.

Stefan Uhlich, Marcello Porcu, Franck Giron, Michael Enenkl, Thomas Kemp, Naoya Takahashi, and Yuki Mitsufuji. Improving music source separation based on deep neural networks through data augmentation and network blending. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017.

Vladimir Vapnik. Principles of risk minimization for learning theory. In *Advances in Neural Information Processing Systems*, 1992.

Vladimir Vapnik. *Estimation of dependences based on empirical data*. Springer Science & Business Media, 2006.

Balaji Venu. Multi-core processors - an overview. Preprint on arXiv:1110.3535, 2011.

Emmanuel Vincent, Rémi Gribonval, and Cédric Févotte. Performance measurement in blind audio source separation. *IEEE transactions on audio, speech, and language processing*, 14(4):1462–1469, 2006.

Emmanuel Vincent, Hiroshi Sawada, Pau Bofill, Shoji Makino, and Justinian P Rosca. First stereo audio source separation evaluation campaign: data, algorithms and results. In *International Conference on Independent Component Analysis and Signal Separation*. Springer, 2007.

Tuomas Virtanen. Monaural sound source separation by nonnegative matrix factorization with temporal continuity and sparseness criteria. *IEEE transactions on audio, speech, and language processing*, 15(3):1066–1074, 2007.

DeLiang Wang and Guy J. Brown, editors. *Computational Auditory Scene Analysis*. IEEE Press, Piscataway, NJ, 2006.

Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, et al. Tacotron: Towards end-to-end speech synthesis. Preprint on arXiv:1703.10135, 2017.

Zhong-Qiu Wang, Jonathan Le Roux, DeLiang Wang, and John R Hershey. End-to-end speech separation with unfolded iterative phase reconstruction. Preprint on arXiv:1804.10204, 2018.

Rachel Ward, Xiaoxia Wu, and Leon Bottou. Adagrad stepsizes: Sharp convergence over nonconvex landscapes. In *International Conference on Machine Learning*, 2019.

Norbert Wiener. *Extrapolation, interpolation, and smoothing of stationary time series*. The MIT press, 1964.

Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, 1:433–486, 1995.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, 2017.

Tianbao Yang, Qihang Lin, and Zhe Li. Unified convergence analysis of stochastic momentum methods for convex and non-convex optimization. Preprint on arXiv:1604.03257, 2016.

Ozgur Yilmaz and Scott Rickard. Blind separation of speech mixtures via time-frequency masking. *IEEE Transactions on signal processing*, 52(7):1830–1847, 2004.

Heiga Ze, Andrew Senior, and Mike Schuster. Statistical parametric speech synthesis using deep neural networks. In *IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013.

Heiga Zen, Yannis Agiomyrgiannakis, Niels Egberts, Fergus Henderson, and Przemysław Szczepaniak. Fast, compact, and high quality lstm-rnn based statistical parametric speech synthesizers for mobile devices. Preprint on arXiv:1606.06061, 2016.

Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. Preprint on arXiv:1901.09321, 2019.

Martin Zinkevich, John Langford, and Alex J. Smola. Slow learners are fast. In *Advances in Neural Information Processing Systems*, 2009.

Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J. Smola. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems*, 2010.

Fangyu Zou, Li Shen, Zequn Jie, Weizhong Zhang, and Wei Liu. A sufficient condition for convergences of Adam and RMSprop. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019a.

FY Zou, Li Shen, ZQ Jie, Ju Sun, and Wei Liu. Weighted Adagrad with unified momentum. Preprint on arXiv:1808.03408, 2019b.

A. Supplementary results for Demucs

A.1. Audio samples

We provide audio samples taken from the test set of MusDB. They are available through the ICLR code sharing url[1] along with all the source code to reproduce our experiments. The audio files for the Wave-U-Net and MMDenseLSTM have been obtained from the SiSec Mus 2018 evaluation campaign results website[2]. For Open Unmix, we generated them from the pretrained UMX model using the reference PyTorch implementation[3]. We recommend listening to the audio samples with headphones, while being careful with the volume. An HTML page `index.html` is provided for easier comparison. The following folders are provided:

- Reference: ground truth,
- Open Unmix,
- WaveUNet,
- Demucs: trained only on MusDB,
- DemucsExtra: trained on MusDB and an extra 150 songs,
- ConvTasnet: trained only on MusDB,
- ConvTasnetExtra: trained on MusDB and an extra 150 songs,
- MMDenseNetLSTM, trained on MusDB and an extra 804 songs.

A.2. Results for all metrics with box plots

Reusing the notations from Vincent et al. [2006], let us take a source $j \in 1, 2, 3, 4$ and introduce P_{s_j} (resp $P_{\mathbf{s}}$) the orthogonal projection on s_j (resp on $\text{Span}(s_1, \ldots, s_4)$). We

[1] `https://ai.honu.io/papers/demucs/`
[2] `https://sisec18.unmix.app`
[3] `https://github.com/sigsep/open-unmix-pytorch`.

then take with $\hat{s}_j$ the estimate of source s_j

$$s_{\text{target}} := P_{s_j}(\hat{s}_j) \qquad e_{\text{interf}} := P_{\mathbf{s}}(\hat{s}_j) - P_{s_j}(\hat{s}_j) \qquad e_{\text{artif}} := \hat{s}_j - P_{\mathbf{s}}(\hat{s}_j)$$

We can now define various signal to noise ratio, expressed in decibels (dB): the source to distortion ratio

$$\text{SDR} := 10 \log_{10} \frac{\|s_{\text{target}}\|^2}{\|e_{\text{interf}} + e_{\text{artif}}\|^2},$$

the source to interference ratio

$$\text{SIR} := 10 \log_{10} \frac{\|s_{\text{target}}\|^2}{\|e_{\text{interf}}\|^2}$$

and the sources to artifacts ratio

$$\text{SAR} := 10 \log_{10} \frac{\|s_{\text{target}} + e_{\text{interf}}\|^2}{\|e_{\text{artif}}\|^2}.$$

As explained in the main paper, extra invariants are added when using the `museval` package. We refer the reader to Vincent et al. [2006] for more details. We provide box plots for each metric and each target on Figures A.1, A.2, A.3 and A.4, generated using the notebook provided specifically by the organizers of the SiSec Mus evaluation campaign[4]. Hereafter, we provide the equivalent of Table 1 in the main paper for both SIR and SAR.

			Test SIR in dB				
Architecture	**Wav?**	**Extra?**	`All`	`Drums`	`Bass`	`Other`	`Vocals`
IRM oracle	✗	N/A	15.53	15.61	12.88	12.84	20.78
Open-Unmix	✗	✗	10.49	11.12	10.93	6.59	13.33
Wave-U-Net	✓	✗	6.26	8.83	5.78	2.37	8.06
Demucs	✓	✗	10.39 ±.07	11.81 ±.27	10.55 ±.20	5.90 ±.04	13.31 ±.21
Conv-Tasnet	✓	✗	11.47 ±.09	12.31 ±.09	11.52 ±.15	7.76 ±.07	14.30 ±.32
Demucs	✓	150	11.95 ±.09	13.74 ±.25	13.03 ±.22	7.11 ±.10	13.94 ±.10
Conv-Tasnet	✓	150	12.24 ±.09	13.66 ±.14	13.18 ±.13	8.40 ±.08	13.70 ±.22
MMDenseLSTM	✗	804	12.24	11.94	11.59	8.94	16.48

[4] `https://github.com/sigsep/sigsep-mus-2018-analysis`

			Test SAR in dB				
Architecture	**Wav?**	**Extra?**	`All`	`Drums`	`Bass`	`Other`	`Vocals`
IRM oracle	✗	N/A	8.31	8.40	7.40	7.93	9.51
Open-Unmix	✗	✗	5.90	6.02	6.34	4.74	6.52
Wave-U-Net	✓	✗	4.49	5.29	4.64	3.99	4.05
Demucs	✓	✗	6.08 ±.01	6.18 ±.03	6.41 ±.05	5.18 ±.06	6.54 ±.04
Conv-Tasnet	✓	✗	6.13 ±.04	6.19 ±.05	6.60 ±.07	4.88 ±.02	6.87 ±.05
Demucs	✓	150	6.50 ±.02	7.04 ±.07	6.68 ±.04	5.26 ±.03	7.00 ±.05
Conv-Tasnet	✓	150	6.57 ±.02	7.35 ±.05	6.96 ±.08	4.76 ±.05	7.20 ±.05
MMDenseLSTM	✗	804	6.50	6.96	6.00	5.55	7.48

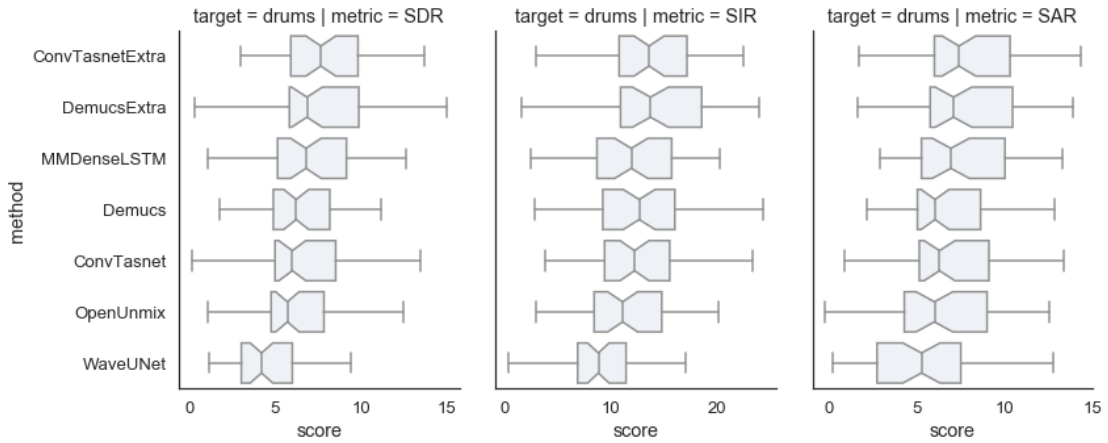


Figure A.1.: Boxplot showing the distribution of SDR, SIR and SAR over the tracks of the MusDB test for the `drums` source.

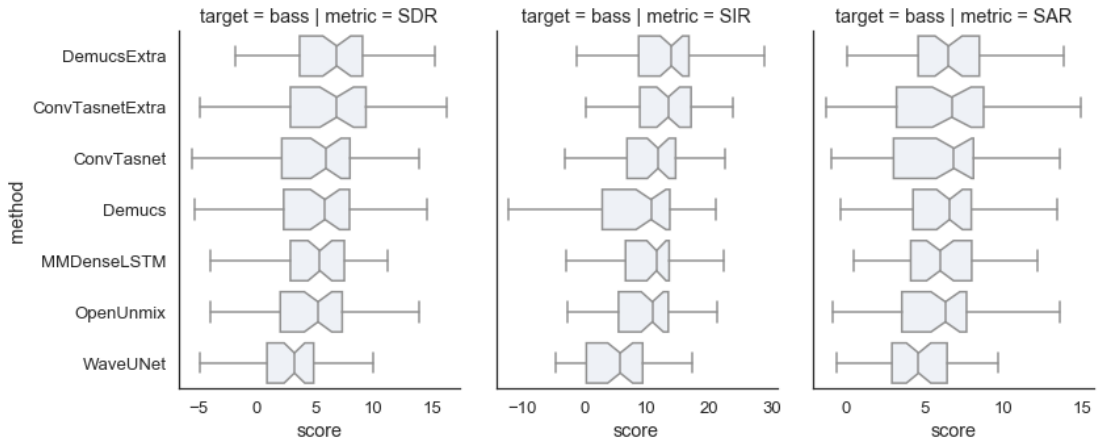


Figure A.2.: Boxplot showing the distribution of SDR, SIR and SAR over the tracks of the MusDB test for the `bass` source.

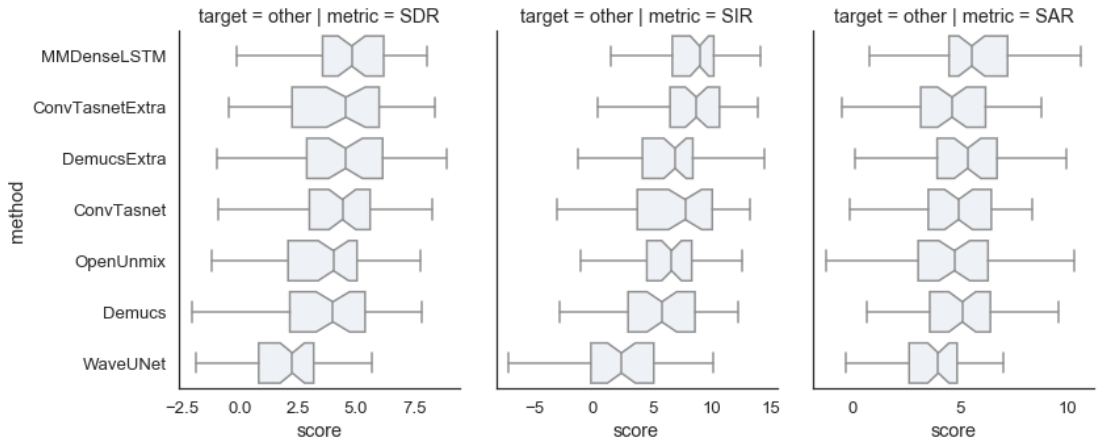


Figure A.3.: Boxplot showing the distribution of SDR, SIR and SAR over the tracks of the MusDB test for the `other` source.

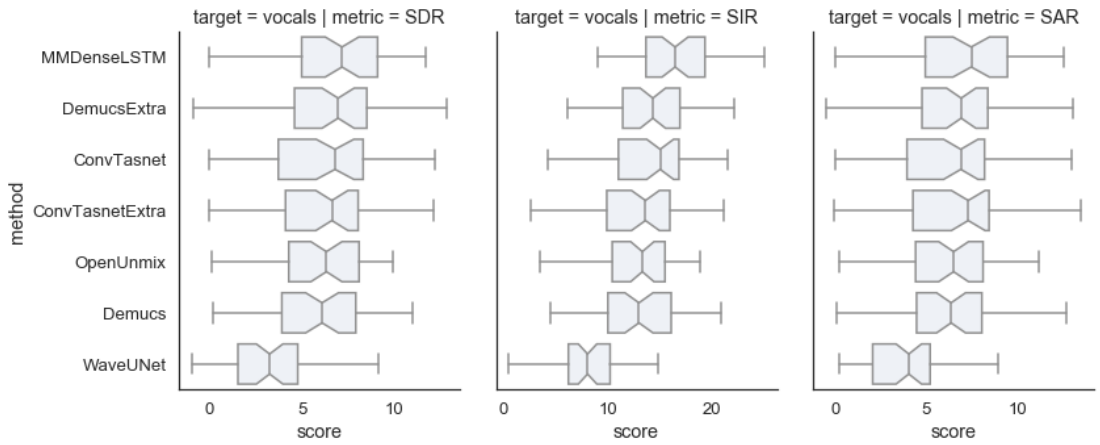


Figure A.4.: Boxplot showing the distribution of SDR, SIR and SAR over the tracks of the MusDB test for the `vocals` source.

B. Proofs of convergence of Adabatch

Introduction

We present in Section B.1 the pseudocode for AdaBatch and Wild AdaBatch. In Section B.2, we give two lemma from which we can derive the expectation and variance of the AdaBatch gradient update. In Section B.3 we study the convergence of regular mini-batch and AdaBatch for SGD as well as the convergence of reconditionned SGD. In Section B.4 we compare the convergence of regular mini-batch and AdaBatch for SVRG. Finally in Section B.5 we give convergence plots for Wild AdaBatch and SVRG on the remaining datasets.

B.1. Algorithms

We present two possible uses of AdaBatch. Algorithm 1 counts for each mini-batch the number of time each feature is non zero and use that to recondition the gradient. This is the algorithm that we study in Section 4.2.

Algorithm 2 is an Hogwild! inspired synchronous SGD method that we introduce in Section 4.4. Instead of counting the features, we directly use the reconditioning $\frac{1-(1-p^{(k)})^B}{p}$ where B is the batch size and $\forall k \in [d], p^{(k)} = \mathbb{P}\left[k \in \mathcal{S}\left(f\right)\right]$ i.e., the probability that feature k is active in a random training sample. We prove in section B.3.2 that this reconditioning benefit from the same convergence speed as regular AdaBatch and does not require to keep count of the features which is easier to implement in the parallel setting, although it requires to precompute the probabilities $p^{(k)}$.

B.2. Expectation and variance of the AdaBatch update

The AdaBatch update $g_{\mathrm{ab},n}$ is defined as

$$\forall k \in [d], g_{\mathrm{ab},n}^{(k)} = \begin{cases} \frac{\sum_{b:k\in\mathcal{S}(f)} f'_{n,b}(w_{n-1})^{(k)}}{\sum_{b:k\in\mathcal{S}(f)} 1} & \text{if } \sum_{b:k\in\mathcal{S}(f)} 1 \neq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{B.3}$$

Algorithm 1 AdaBatch

function AdaBatch(w_0, N, B, γ, f)
  for $n \in [N]$ **do**
    for $b \in [B]$ **do**
      Sample $f_{n,b}$ from the distribution of f
      Compute $f'_{n,b}(w)$
    end for
    for $b \in [B]$ **do**
      for $k \in \mathcal{S}(f_{n,b})$ **do**

$$w_n^{(k)} \leftarrow w_{n-1}^{(k)} - \gamma \frac{f'_{n,b}(w_{n-1})^{(k)}}{|\{b : k \in \mathcal{S}(f_{n,b})\}|} \tag{B.1}$$

      end for
    end for
  end for
end function

Algorithm 2 Wild AdaBatch

function Wild AdaBatch(w_0, N, B, γ, p, f)
  for $n \in [N]$ **do**
    parallel for $b \in [B]$ **do**
      Sample $f_{n,b}$ from the distribution of f
      Compute $f'_{n,b}(w)$
    end parallel for
    parallel for $b \in [B]$ **do**
      for $k \in \mathcal{S}(f_{n,b})$ **do**

$$w_n^{(k)} \leftarrow w_{n-1}^{(k)} - \frac{\gamma}{B} \frac{1 - (1 - p^{(k)})^B}{p} f'_{n,b}(w_{n-1})^{(k)} \tag{B.2}$$

      end for
    end parallel for
  end for
end function

so that we have the recurrence rule for SGD

$$w_n = w_{n-1} - \gamma g_{\mathrm{ab},n}.$$

$g_{\mathrm{ab},n}$ is a per coordinate stochastic average of only the subset of the gradients which have a non zero coordinate in that direction. We will need the following Lemma in order to get the expected value and the variance of $g_{\mathrm{ab},n}$.

Lemma B.1. *Let Z, $(Z_i)_{i\in[N]}$ N i.i.d. random variables with value in $\mathbb{R}$ for which the set $\{0\}$ is measurable with $p = \mathbb{P}[Z \neq 0] > 0$, and $A \in \mathbb{R}$ a random variable defined as*

$$\begin{cases} A := 0 \quad \textit{if} \quad \forall i \in [N], Z_i = 0 \\ A := \frac{\sum_{i\in[N]} Z_i}{\sum_{i\in[N]:Z_i!=0} 1} \quad \textit{otherwise.} \end{cases}.$$

We have

$$\mathbb{E}[A] = \frac{1-(1-p)^N}{p}\mathbb{E}[Z], \tag{B.4}$$

$$\mathbb{E}\left[A^2\right] = \frac{(1-(1-p)^N)^2}{p^2}\mathbb{E}[Z]^2 + \left(\sum_{i\in[N]}\binom{N}{i}p^i(1-p)^{N-i}\frac{1}{i}\right)\left(\frac{\mathbb{E}[Z^2]}{p} - \frac{\mathbb{E}[Z]^2}{p^2}\right) \tag{B.5}$$

$$\leq \frac{(1-(1-p)^N)^2}{p^2}\mathbb{E}[Z]^2 + \frac{(1-(1-p)^N)}{p}\mathbb{E}\left[Z^2\right]. \tag{B.6}$$

Proof. We introduce μ the measure of Z and μ^+ the measure of Z^+ defined for any measurable $A \subset \mathbb{R}$

$$\mu^+(A) = \frac{\mu(A \setminus \{0\})}{\mu(\mathbb{R} \setminus \{0\})}.$$

Intuitively Z^+ is the random variable we obtain if we drop all realizations where $Z = 0$. One can verify that $\mathbb{E}[Z^+] = \frac{\mathbb{E}[Z]}{p}$ and $\mathbb{E}[(Z^+)^2] = \frac{\mathbb{E}[Z^2]}{p}$.

Taking $Q \sim \mathcal{B}(p)$ a Bernoulli of parameter p independent from Z^+, one can readily notice that $Z \sim QZ^+$. We thus take N i.i.d. such copies $(Q_i, Z_i^+)_{i\in[N]}$. Let us take any $q \in \{0,1\}^N$ so that $|q| := \sum_{i\in[N]} q_i > 0$,

$$\begin{aligned} \mathbb{E}[A|Q=q] &= \mathbb{E}\left[\frac{\sum_{i\in[N]} q_i Z_i^+}{|q|}\right] \\ &= \frac{\sum_{i:q_i=1}\mathbb{E}\left[Z_i^+\right]}{|q|} \\ &= \mathbb{E}\left[Z^+\right]. \end{aligned}$$

Given that $\mathbb{E}\left[A|\sum_{i\in[N]}Q_i=0\right]=0$ and that $\mathbb{P}\left[\sum_{i\in[N]}Q_i\neq 0\right]=1-(1-p)^N$, we get (B.4).

We will denote $\mathbb{V}\left[A|Q=q\right]=\mathbb{E}\left[A^2|Q=q\right]-\mathbb{E}\left[A|Q=q\right]^2$. We study

$$\begin{aligned}\mathbb{E}\left[A^2|Q=q\right]&=\mathbb{V}\left[A|Q=q\right]+\mathbb{E}\left[A|Q=q\right]^2\\&=\mathbb{V}\left[\frac{\sum_{i:q_i=1}Z_i^+}{|q|}\right]+\frac{\mathbb{E}\left[Z\right]^2}{p^2}\\&=\frac{\mathbb{V}\left[Z^+\right]}{|q|}+\frac{\mathbb{E}\left[Z\right]^2}{p^2}\\&=\frac{\mathbb{E}\left[Z^2\right]}{p\,|q|}-\frac{\mathbb{E}\left[Z\right]^2}{p^2\,|q|}+\frac{\mathbb{E}\left[Z\right]^2}{p^2}.\end{aligned}$$

$$\begin{aligned}\mathbb{E}\left[A^2\right]&=\sum_{k\in[N]}\mathbb{E}\left[A^2|\,|Q|=k\right]\mathbb{P}\left[|Q|=k\right]\\&=\sum_{k\in[N]}\binom{N}{k}p^k(1-p)^{N-k}\frac{1}{k}\left(\frac{\mathbb{E}\left[Z^2\right]}{p}-\frac{\mathbb{E}\left[Z\right]^2}{p^2}\right)+\frac{\mathbb{E}\left[Z\right]^2}{p^2}(1-(1-p)^N)\\&\leq(1-(1-p)^N)\frac{\mathbb{E}\left[Z^2\right]}{p}+(1-(1-p)^N)\frac{\mathbb{E}\left[Z\right]^2}{p^2},\end{aligned}$$

as $\sum_{k\in[N]}\binom{N}{k}p^k(1-p)^{N-k}\frac{1}{k}\leq 1-(1-p)^N$ which gives us (B.5) and conclude this proof. □

Thanks to Lemma B.1 we get

$$\forall k\in[d],\mathbb{E}\left[g_{\mathrm{ab},n}^{(k)}\right]=\frac{1-(1-p^{(k)})^B}{p^{(k)}}\mathbb{E}\left[F'(w_{n-1})\right] \tag{B.7}$$

$$\forall k\in[d],\mathbb{E}\left[\left(g_{\mathrm{ab},n}^{(k)}\right)^2\right]\leq\frac{(1-(1-p^{(k)})^N)}{p}\mathbb{E}\left[\left(f'(w_{n-1})^{(k)}\right)^2\right]+\frac{(1-(1-p^{(k)})^N)}{p^2}\left\|F'(w_{n-1})^{(k)}\right\|^2. \tag{B.8}$$

We now present an improved bound for the second order moment of Z that can be better if than the previous one in the case where Np is large enough. Although we will not provide a full proof of convergence using this result for simplicity, we will comment on how this impact convergence in the proof of theorem 6.

Lemma B.2. *With the same notation as in lemma B.1, if $Np\geq 5$ we have*

$$\mathbb{E}\left[A^2\right]\leq\frac{5(1-(1-p)^N)\mathbb{E}\left[Z^2\right]}{Np^2}+\frac{(1-(1-p)^N)\mathbb{E}\left[Z\right]^2}{p^2}. \tag{B.9}$$

Proof. We reuse the notation from the proof of Lemma B.1. Let us define $M := |Q|$ which follows a binomial law of parameter N and p. Using Chernoff's inequality, we have for any $k \leq Np$,

$$\mathbb{P}\left[M \leq k\right] \leq \exp\left(-\frac{(Np-k)^2}{2Np}\right),$$

taking $k = \frac{Np}{2}$ we obtain

$$\mathbb{P}\left[M \leq \frac{Np}{2}\right] \leq \exp\left(-\frac{Np}{8}\right).$$

We have

$$\begin{aligned}\mathbb{E}\left[\frac{1}{M}|M>0\right] &\leq \mathbb{P}\left[M \leq \frac{Np}{2}|M>0\right] + \frac{2}{Np} \\ &= \frac{\mathbb{P}\left[M \leq \frac{Np}{2}\right]}{\mathbb{P}\left[M>0\right]} + \frac{2}{Np} \\ &\leq \frac{\exp\left(-\frac{Np}{8}\right)}{1-(1-p)^N} + \frac{2}{Np}.\end{aligned}$$

We have as $p \geq 5/N$ and using standard analysis techniques,

$$\frac{\exp\left(-\frac{Np}{8}\right)}{1-(1-p)^N} \leq \frac{3}{Np}.$$

We obtain

$$\mathbb{E}\left[\frac{1}{M}|M>0\right] \leq \frac{5}{Np}.$$

Plugging this result into (B.5), we immediately have

$$\mathbb{E}\left[A^2\right] \leq \frac{5(1-(1-p)^N)\mathbb{E}\left[Z^2\right]}{Np^2} + \frac{(1-(1-p)^N)\mathbb{E}\left[Z\right]^2}{p^2}.$$

□

B.3. Proof of convergence of AdaBatch and mini-batch SGD

B.3.1. Constant step size SGD with mini-batch

We will first give a convergence result for the regular mini-batch SGD, which is adapted from Needell and Ward [2016].

Assumption 2. *We assume there exists a convex compact set $\mathcal{D} \subset \mathbb{R}^d$ so that f and F verifies the following assumptions for μ, L and R strictly positive,*

1. *The hessian F'' of F is bounded from above and below as:*

$$\forall w \in \mathcal{D}, \mu \operatorname{Id} \preceq F''(w) \preceq L \operatorname{Id}, \tag{B.10}$$

   with $\mu > 0$ so that f is μ strongly convex and L smooth over $\mathcal{D}$.

2. *We assume f'' is almost surely bounded,*

$$\forall w \in \mathcal{D}, f''(w) \preceq R^2 \operatorname{Id}. \tag{B.11}$$

3. *Let $w_* := \arg\min_{w \in \mathcal{D}} F(w)$,*

$$F'(w_*) = 0, \tag{B.12}$$

   which means in particular that w_* is a global minimizer of F over $\mathbb{R}^d$.

This does not limit us to the case of *globally* strongly convex functions F as we only require it to be strongly convex on a compact subset that contains the global optimum w_*. In practice, this is often going to be the case, even when using a non strictly convex loss such as the logistic loss as soon as the problem is not perfectly separable, i.e., there is no hyperplane that perfectly separates the classes we are trying to predict.

We will now study the recursion for a given $w_0 \in \mathbb{R}^d$ given by

$$\forall n > 0, w_n = \Pi_{\mathcal{D}} \left[w_{n-1} - \frac{\gamma}{B} \sum_{b \in [B]} f'_{n,b}(w_{n-1}) \right], \tag{B.13}$$

where $\Pi_{\mathcal{D}}[w] := \arg\min_{x \in \mathcal{D}} \|w - x\|^2$ is the orthogonal projection on the set $\mathcal{D}$. This extra step of projection is required for this proof technique but experience shows that it is not needed.

Theorem 5 (Convergence of $F_n - F_*$ for SGD with mini-batch)**.** *If Assumptions 2 are verified and*

$$\gamma \left[L \left(1 - \frac{1}{B} \right) + \frac{2R^2}{B} \right] \leq 1, \tag{B.14}$$

then for any $N > 0$,

$$\|w_N - w_*\|^2 \leq (1 - \gamma\mu/2)^N \|w_0 - w_*\|^2 + \frac{4\gamma}{B\mu} \mathbb{E} \left[\|f'(w_*)\|^2 \right], \tag{B.15}$$

and introducing

$$\bar{w}_N = \frac{\sum_{n\in[N]}(1-\gamma\mu/2)^{N-n}w_n}{\sum_{n\in[N]}(1-\gamma\mu/2)^{N-n}},$$

we have

$$\mathbb{E}\left[F(\bar{w}_N)\right] - F_* \leq \gamma^{-1}(1-\gamma\mu/2)^N \left\|w_0 - w_*\right\|^2 + \frac{2\gamma}{B}\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right]. \tag{B.16}$$

Introducing $\bar{w}_N$ allows for an easier comparison directly on the objective function. This is made for qualitative analysis and we do not in practice perform this averaging.

We can see that the error given by (B.16) can be composed in two terms, one that measure how quickly we move away from the starting point and the second that depends on the stochastic noise around the optimum. We will call the former the *bias* term and the latter the *variance* term, following the terminology introduced by Bach and Moulines [2013].

Proof. We introduce $\forall n \in [N], \mathcal{F}_{n-1}$ the σ-field generated by $(f_{i,b})_{i\in[n-1],b\in[B]}$. Let us take $n \in [N]$ and introduce $\eta_n := w_n - w_*$ and $g_n := \frac{1}{B}\sum_{b\in[B]} f'_{n,b}(w_{n-1})$. We then proceed to bound $\left\|\eta_n\right\|^2$,

$$\begin{aligned}\left\|\eta_n\right\|^2 &\leq \left\|\eta_{n-1} - \gamma g_n\right\|^2 \quad \text{as } \Pi_{\mathcal{D}} \text{ is contractant for } \|\cdot\| \\ &= \left\|\eta_{n-1}\right\|^2 - 2\gamma g_n^T \eta_{n-1} + \gamma^2 \left\|g_n\right\|^2.\end{aligned}$$

Taking the expectation while conditioning on $\mathcal{F}_{n-1}$ we obtain

$$\mathbb{E}\left[\left\|\eta_n\right\|^2 |\mathcal{F}_{n-1}\right] \leq \left\|\eta_{n-1}\right\|^2 - 2\gamma F'(w_{n-1})^T\eta_{n-1} + \gamma^2\mathbb{E}\left[\left\|g_n\right\|^2 |\mathcal{F}_{n-1}\right], \tag{B.17}$$

$$\mathbb{E}\left[\left\|g_n\right\|^2 |\mathcal{F}_{n-1}\right] = \frac{\mathbb{E}\left[\left\|f'(w_{n-1})\right\|^2\right]}{B} + \left\|F'(w_{n-1})\right\|^2\left(1-\frac{1}{B}\right).$$

Injecting this in (B.17) gives us

$$\mathbb{E}\left[\left\|\eta_n\right\|^2 |\mathcal{F}_{n-1}\right] \leq \left\|\eta_{n-1}\right\|^2 - 2\gamma F'(w_{n-1})^T\eta_{n-1} + \frac{\gamma^2}{B}\mathbb{E}\left[\left\|f'(w_{n-1})\right\|^2\right] + \gamma^2\left\|F'(w_{n-1})\right\|^2\left(1-\frac{1}{B}\right). \tag{B.18}$$

As $F'' \preceq L\,\mathrm{Id}$ and using the co-coercivity of F' we have

$$\begin{aligned}\left\|F'(w_{n-1})\right\|^2 &= \left\|F'(w_{n-1}) - F'(w_*)\right\|^2 \\ &\leq L(F'(w_{n-1}) - F'(w_*))^T(w_{n-1} - w_*) \\ &= LF'(w_{n-1})^T(w_{n-1} - w_*).\end{aligned}$$

We have

$$\begin{aligned}\left\|f'(w_{n-1})\right\|^2 &\leq 2\left\|f'(w_{n-1})-f'(w_*)\right\|^2+2\left\|f'(w_*)\right\|^2\\ &\leq 2R^2(f'(w_{n-1})-f'(w_*))^T(w_{n-1}-w_*)+2\left\|f'(w_*)\right\|^2,\end{aligned}$$

and

$$\mathbb{E}\left[\left\|f'(w_{n-1})\right\|^2|\mathcal{F}_{n-1}\right]\leq 2R^2F'(w_{n-1})^T(w_{n-1}-w_*)+2\left\|f'(w_*)\right\|^2.$$

Injecting in (B.18) we get

$$\mathbb{E}\left[\left\|\eta_n\right\|^2|\mathcal{F}_{n-1}\right]\leq\left\|\eta_{n-1}\right\|^2-\gamma F'(w_{n-1})^T\eta_{n-1}\underbrace{\left(2-\gamma L\left(1-\frac{1}{B}\right)-\frac{2\gamma R^2}{B}\right)}_{A}+\frac{2\gamma^2\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right]}{B}.$$

We want A to be large enough, we will take

$$\gamma\left[L\left(1-\frac{1}{B}\right)+\frac{2R^2}{B}\right]\leq 1, \tag{B.19}$$

which gives us $A\geq 1$ and

$$\mathbb{E}\left[\left\|\eta_n\right\|^2|\mathcal{F}_{n-1}\right]\leq\left\|\eta_{n-1}\right\|^2-\gamma F'(w_{n-1})^T\eta_{n-1}+\frac{2\gamma^2\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right]}{B}.$$

As F is μ strictly convex, we have

$$F_*-F(w_{n-1})\geq F'(w_{n-1})^T(w_*-w_{n-1})+\frac{\mu}{2}\left\|\eta_{n-1}\right\|^2,$$

which allows to obtain

$$\mathbb{E}\left[\left\|\eta_n\right\|^2|\mathcal{F}_{n-1}\right]\leq(1-\gamma\mu/2)\left\|\eta_{n-1}\right\|^2-\gamma(F(w_{n-1})-F_*)+\frac{2\gamma^2}{B}\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right]. \tag{B.20}$$

Taking the full expectation gives us

$$\begin{aligned}\mathbb{E}\left[\left\|\eta_n\right\|^2\right]&\leq(1-\gamma\mu/2)\mathbb{E}\left[\left\|\eta_{n-1}\right\|^2\right]-\gamma(\mathbb{E}\left[F(w_{n-1})\right]-F_*)+\frac{2\gamma^2\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right]}{B},\\ &\leq(1-\gamma\mu/2)^n\left\|\eta_0\right\|^2+\frac{2\gamma^2\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right]}{B}\sum_{0\leq i<n}(1-\gamma\mu/2)^i\\ &\leq(1-\gamma\mu/2)^n\left\|\eta_0\right\|^2+\frac{4\gamma}{B\mu}\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right],\end{aligned}$$

which gives us (B.15).

Let us now take $\alpha := (1 - \gamma\mu/2)$, let us call $u_n := \mathbb{E}\left[\|\eta_n\|^2\right]$, we have using (B.20),

$$\gamma\delta_{n-1} \leq \alpha u_{n-1} - u_n + 2\gamma^2 R^2$$
$$\gamma\delta_{n-1}\alpha^{-n} \leq \alpha^{-n+1}u_{n-1} - u_n\alpha^{-n} + 2\gamma^2 R^2\alpha^{-n},$$

summing for n from 1 to N we obtain

$$\gamma \sum_{n\in[N]} \delta_{n-1}\alpha^{-n} \leq u_0 - u_N\alpha^{-N} + \frac{2\gamma^2}{B}\mathbb{E}\left[\|f'(w_*)\|^2\right] \sum_{n\in[N]} \alpha^{-n},$$

dividing by $\sum_{n\in[N]} \alpha^{-n}$ on each side and using the convexity of F we get

$$\mathbb{E}\left[F(\bar{w}_{N-1})\right] - F_* \leq \alpha^N u_0 + \frac{2\gamma^2}{B}\mathbb{E}\left[\|f'(w_*)\|^2\right],$$

which gives us (B.16) and concludes this proof. □

B.3.2. Convergence of reconditioned SGD

Let us now assume that we have for some matrices T and C definite positive so that

$$\forall w \in \mathcal{D}, \mu T \preceq F''(w) \preceq LT,$$
$$\forall w \in \mathcal{D}, f''(w) \preceq L\,\mathrm{Id} \quad a.s.$$

We now study w_n defined by the following recurence

$$w_n = w_{n-1} - \frac{\gamma}{B}C \sum_{b\in[B]} f'_{n,b}(w_{n-1}). \tag{B.21}$$

First let us introduce $v_0 := \sqrt{C}^{-1}w_0$, $v_* := \sqrt{C}^{-1}w_*$ and $h(w) := f(\sqrt{C}w)$ as well as $\forall n \in [N], b \in [B], h_{n,b}(w) := f_{n,b}(\sqrt{C}w)$, then we define

$$v_n := v_{n-1} - \frac{\gamma}{B} \sum_{b\in[B]} h'_{m,b}(v_{n-1}).$$

Multiplying by $\sqrt{C}$ we recover the same recurrence rule as (B.21) for $w_n = \sqrt{C}v_n$. Therefore, the convergence of v_n will give us the convergence of w_n.

Let us take $H(w) := \mathbb{E}\left[h(w)\right] = F(\sqrt{C}w)$. By definition we have

$$\forall w \in \mathcal{D}_C, \mu\sqrt{C}T\sqrt{C} \preceq F''(w) \preceq L\sqrt{C}T\sqrt{C},$$
$$\forall w \in \mathcal{D}_C, h''(w) \preceq R^2 L_C\,\mathrm{Id} \quad a.s.,$$

where $\mathcal{D}_C = \sqrt{C}^{-1}\mathcal{D}$, L_C is the largest eigen value of C. If we take $\mu_{C,T}$ (resp $L_{C,T}$) the smallest (resp largest) eigenvalue of $\sqrt{C}T\sqrt{C}$, then using Theorem 5, we have for

$$\gamma\left[LL_{C,T}\left(1-\frac{1}{B}\right)+\frac{2L_CR^2}{B}\right]\leq 1, \tag{B.22}$$

$$\|v_N-v_*\|^2\leq(1-\gamma\mu_{C,T})^N\|v_0-v_*\|^2+\frac{2\gamma}{B\mu}\mathbb{E}\left[\|h'(v_*)\|^2\right],$$

and introducing

$$\bar{v}_N=\frac{\sum_{n\in[N]}(1-\gamma\mu_{C,T}/2)^{N-n}v_n}{\sum_{n\in[N]}(1-\gamma\mu_{C,T}/2)^{N-n}},$$

we have

$$\mathbb{E}\left[H(\bar{v}_N)\right]-H_*\leq\gamma^{-1}(1-\gamma\mu_{C,T})^N\|v_0-v_*\|^2+\frac{2\gamma L_CR^2}{B}\mathbb{E}\left[\|h'(v_*)\|^2\right].$$

Using $w_n=\sqrt{C}v_n$, we obtain

$$\|w_N-w_*\|_{C^{-1}}^2\leq(1-\gamma\mu_{C,T})^N\|w_0-w_*\|_{C^{-1}}^2+\frac{4\gamma}{B\mu}\mathbb{E}\left[\|f'(w_*)\|_C^2\right],$$

$$\mathbb{E}\left[F(\bar{w}_N)\right]-F_*\leq\gamma^{-1}(1-\gamma\mu_{C,T})^N\|w_0-w_*\|_{C^{-1}}^2+\frac{2\gamma L_C}{B}\mathbb{E}\left[\|f'(w_*)\|_C^2\right]. \tag{B.23}$$

Application to sparse optimization. In the sparse setting, we have made the assumption that $T:=\mathrm{Diag}\,(p)$. We suggested two reconditioning strategies in such case. The first one is to take $C=\mathrm{Diag}\,(p)^{-1}$. In such case $\mu_{C,T}=L_{C,T}=1$ so that we have a perfect conditioning. However $L_C=p_{\min}^{-1}$ so that if $B\ll p_{\min}^{-1}$ we would have to take a much smaller step size and the term $\frac{1}{B}\mathbb{E}\left[\|f'(w_*)\|^2\right]_{\mathrm{Diag}(p)^{-1}}$ would explode.

The second one, $C:=\mathrm{Diag}\left(\frac{1-(1-p)^B}{p}\right)$ so that $\mu_{C,T}=1-(1-p_{\min})^B$ and $L_{C,T}=1-(1-p_{\max})^B$. Because the fonction $p\to 1-(1-p)^B$ increases faster for small probabilities, the conditioning of the problem is improved. If $p_{\max}$ is close to 1 and $p_{\min}$ close to 0, then $\mu_{C,T}\approx Bp_{\min}$ and $L_C\approx p_{\max}$. We have $L_C\leq B$ as $\forall p\in[0,1],(1-(1-p)^B)\leq Bp$, so that the increase due to L_C is perfectly balanced out by the batch size B in (B.22). Besides, as $\frac{C}{B}\preceq\mathrm{Id}$, we have $\frac{1}{B}\mathbb{E}\left[\|f'(w_*)\|_C^2\right]\leq\mathbb{E}\left[\|f'(w_*)\|^2\right]$.

B.3.3. Convergence of AdaBatch

We now make the following assumptions:

Assumption 3. *We assume there exists a convex compact set $\mathcal{D}\subset\mathbb{R}^d$, μ, L and R strictly positive so that we have the following assumptions verified.*

1. *The Hessian F'' (resp f'') of F (resp f) are bounded from above and below as:*

$$\forall w \in \mathcal{D}, \quad \mu \mathrm{Diag}\,(p) \preceq F''(w) \preceq L\mathrm{Diag}\,(p) \quad \textit{and} \quad \forall w \in \mathcal{D}, f''(w) \preceq R^2\,\mathrm{Id}\,. \tag{B.24}$$

2. *Let $w_* := \arg\min_{w\in\mathcal{D}} F(w)$,*

$$F'(w_*) = 0. \tag{B.25}$$

In particular, w_* is a global minimizer of F over $\mathbb{R}^d$.

We will now study convergence when we use the AdaBatch update. We are given $w_0 \in \mathbb{R}^d$ and we define $C := \mathrm{Diag}\left(\frac{1-(1-p)^B}{p}\right)$ and we define recursively,

$$\forall k \in [d], g^k_{\mathrm{ab},n} = \begin{cases} \frac{\sum_{b\in B_n^k} f'_{n,b}(w_{n-1})^k}{\sum_{b:k\in\mathcal{S}(f)} 1} & \text{if } \sum_{b:k\in\mathcal{S}(f)} 1 \neq 0 \\ 0 & \text{otherwise} \end{cases}$$
$$w_n = \Pi_{\mathcal{D},C}\left[w_{n-1} - \gamma g_{\mathrm{ab},n}\right],$$

where $\Pi_{\mathcal{D},C}[w] := \arg\min_{x\in\mathcal{D}} \|w-x\|^2_{C^{-1}}$ is the projection on the set $\mathcal{D}$ with respect to $\|\cdot\|_{C^{-1}}$. This extra step of projection is required for this proof technique but experience shows that it is not needed. We introduce $\forall p \in [0,1], p^{+B} := 1-(1-p)^B$.

Theorem 6 (Convergence of $F_n - F_*$ for AdaBatch)**.** *If Assumptions 3 are verified and*

$$\gamma\left(L + 2R^2\right) \leq 1, \tag{B.26}$$

then for any $N > 0$,

$$\|w_N - w_*\|^2 \leq (1-\gamma p_{\min}^{+B}\mu/2)^N \|w_0 - w_*\|^2 + \frac{4\gamma}{\mu}\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right], \tag{B.27}$$

and introducing

$$\bar{w}_N = \frac{\sum_{n\in[N]}(1-\gamma p_{\min}^{+B}\mu/2)^{N-n} w_n}{\sum_{n\in[N]}(1-\gamma p_{\min}^{+B}\mu/2)^{N-n}},$$

we have

$$\mathbb{E}\left[F(\bar{w}_N)\right] - F_* \leq \gamma^{-1}(1-\gamma p_{\min}^{+B}\mu/2)^N \|w_0 - w_*\|^2 + 2\gamma\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right]. \tag{B.28}$$

Proof. We introduce $\forall n \in [N], \mathcal{F}_{n-1}$ the σ-field generated by $(f_{i,b})_{i\in[n-1],b\in[B]}$. Let us take $n \in [N]$ and introduce $\eta_n := w_n - w_*$. We then proceed to bound $\|\eta_n\|^2_{C^{-1}}$,

$$\begin{aligned} \|\eta_n\|^2_{C^{-1}} &\leq \|\eta_{n-1} - \gamma g_n\|^2_{C^{-1}} \quad \text{as } \Pi_{\mathcal{D}} \text{ is contractant for } \|\cdot\|_{C^{-1}} \\ &= \|\eta_{n-1}\|^2_{C^{-1}} - 2\gamma g_n^T \eta_{n-1} + \gamma^2 \|g_n\|^2_{C^{-1}}\,. \end{aligned}$$

Taking the expectation while conditioning on $\mathcal{F}_{n-1}$ we obtain

$$\mathbb{E}\left[\|\eta_n\|^2_{C^{-1}}|\mathcal{F}_{n-1}\right] \leq \|\eta_{n-1}\|^2_{C^{-1}} - 2\gamma F'(w_{n-1})^T\eta_{n-1} + \gamma^2\mathbb{E}\left[\|g_n\|^2_{C^{-1}}|\mathcal{F}_{n-1}\right]. \quad \text{(B.29)}$$

Using Lemma B.1,

$$\mathbb{E}\left[\|g_n\|^2_{C^{-1}}|\mathcal{F}_{n-1}\right] \leq \mathbb{E}\left[\left\|f'(w_{n-1})\right\|^2\right] + \left\|F'(w_{n-1})\right\|^2_{\mathrm{diag}(p)^{-1}}. \quad \text{(B.30)}$$

Although we will not do it in the following, it is also possible to use Lemma B.2. Indeed, for any $k \in [d]$ such that $Bp^{(k)} \geq 5$, we have

$$\mathbb{E}\left[\left(g_n^{(k)}\right)^2 C^{-1}_{k,k}|\mathcal{F}_{n-1}\right] \leq \frac{5\mathbb{E}\left[(f'(w_{n-1})^{(k)})^2\right]}{Np} + \frac{(F'(w_{n-1})^{(k)})^2}{p^{(k)}},$$

which would be equivalent for the dimension k to having a regular batch size of $\frac{p^{(k)}B}{5}$. This shows that AdaBatch will benefit from a reduced variance for features that are frequent enough. For simplicity we will however stick with the simpler bound given by (B.30).

As $F'' \preceq L\mathrm{diag}\,(p)$ and using the co-coercivity of F' we have

$$\begin{aligned}\left\|F'(w_{n-1})\right\|^2_{\mathrm{diag}(p)^{-1}} &= \left\|F'(w_{n-1}) - F'(w_*)\right\|^2_{\mathrm{diag}(p)^{-1}} \\ &\leq L(F'(w_{n-1}) - F'(w_*))^T(w_{n-1} - w_*) \\ &= LF'(w_{n-1})^T(w_{n-1} - w_*).\end{aligned}$$

Injecting this in (B.29) gives us

$$\mathbb{E}\left[\|\eta_n\|^2|\mathcal{F}_{n-1}\right] \leq \|\eta_{n-1}\|^2_{C^{-1}} - 2\gamma F'(w_{n-1})^T\eta_{n-1} + \gamma^2\mathbb{E}\left[\left\|f'(w_{n-1})\right\|^2\right]. \quad \text{(B.31)}$$

We have

$$\begin{aligned}\left\|f'(w_{n-1})\right\|^2 &\leq 2\left\|f'(w_{n-1}) - f'(w_*)\right\|^2 + 2\left\|f'(w_*)\right\|^2 \\ &\leq 2R^2(f'(w_{n-1}) - f'(w_*))^T(w_{n-1} - w_*) + 2\left\|f'(w_*)\right\|^2\end{aligned}$$

and

$$\mathbb{E}\left[\left\|f'(w_{n-1})\right\|^2|\mathcal{F}_{n-1}\right] \leq 2R^2F'(w_{n-1})^T(w_{n-1} - w_*) + 2\left\|f'(w_*)\right\|^2.$$

Injecting in (B.31) we get

$$\mathbb{E}\left[\|\eta_n\|^2_{C^{-1}}|\mathcal{F}_{n-1}\right] \leq \|\eta_{n-1}\|^2_{C^{-1}} - \gamma F'(w_{n-1})^T\eta_{n-1}\underbrace{\left(2 - \gamma L - 2\gamma R^2\right)}_{A} + 2\gamma^2\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right].$$

We want A to be large enough, we will take

$$\gamma\left(L + 2R^2\right) \leq 1, \tag{B.32}$$

which gives us $A \geq 1$ and

$$\mathbb{E}\left[\|\eta_n\|_{C^{-1}}^2 |\mathcal{F}_{n-1}\right] \leq \|\eta_{n-1}\|_{C^{-1}}^2 - \gamma F'(w_{n-1})^T \eta_{n-1} + 2\gamma^2 \mathbb{E}\left[\left\|f'(w_*)\right\|^2\right].$$

As $F'' \succeq \mu \mathrm{Diag}\,(p)$ we have

$$\begin{aligned} F_* - F(w_{n-1}) &\geq F'(w_{n-1})^T(w_* - w_{n-1}) + \frac{\mu}{2}\|\eta_{n-1}\|_{\mathrm{Diag}(p)}^2 \\ &\geq F'(w_{n-1})^T(w_* - w_{n-1}) + \frac{p_{\min}^{+B}\mu}{2}\|\eta_{n-1}\|_{C^{-1}}^2, \end{aligned}$$

which allows to obtain

$$\mathbb{E}\left[\|\eta_n\|_{C^{-1}}^2 |\mathcal{F}_{n-1}\right] \leq (1 - \gamma\mu p_{\min}^{+B}/2)\|\eta_{n-1}\|_{C^{-1}}^2 - \gamma(F(w_{n-1}) - F_*) + 2\gamma^2\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right]. \tag{B.33}$$

Taking the full expectation gives us

$$\begin{aligned} \mathbb{E}\left[\|\eta_n\|_{C^{-1}}^2\right] &\leq (1 - \gamma\mu p_{\min}^{+B}/2)\mathbb{E}\left[\|\eta_{n-1}\|_{C^{-1}}^2\right] - \gamma(\mathbb{E}\left[F(w_{n-1})\right] - F_*) + 2\gamma^2\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right], \\ &\leq (1 - \gamma\mu p_{\min}^{+B}/2)^n \|\eta_0\|_{C^{-1}}^2 + 2\gamma^2\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right] \sum_{0\leq i<n} (1 - \gamma\mu p_{\min}^{+B}/2)^i \\ &\leq (1 - \gamma\mu p_{\min}^{+B}/2)^n \|\eta_0\|_{C^{-1}}^2 + \frac{4\gamma}{\mu}\mathbb{E}\left[\left\|f'(w_*)\right\|^2\right], \end{aligned}$$

which gives us (B.27). We obtain (B.28) in the exact same way as in the proof of Theorem 5. □

B.3.4. Sparse linear prediction

We will now show that Assumption 3 is easy to meet in the case of linear predictions. For simplicity, let us assume X is a random variable with values in $\{0,1\}^d$ with uncorrelated features, i.e.,

$$\forall k, k' \in [d] : k \neq k', \mathbb{E}\left[X^{(k)}X^{(k')}\right] = \mathbb{E}\left[X^{(k)}\right]X^{(k')},$$

and $\phi : \mathbb{R} \to \mathbb{R}$ a random convex function. Then one can take $f(w) := \phi(X^T w)$. For m, M and G strictly positive and $\mathcal{D} \subset \mathbb{R}^d$ a convex compact, We assume almost surely we have

$$\begin{aligned} &\forall w \in \mathcal{D}, m \leq \phi''(w) \leq M, \\ &\|X\|^2 \leq G^2 \quad a.s. \end{aligned}$$

Then, for any $w \in \mathcal{D}$ we have

$$\begin{aligned} F''(w) &= \mathbb{E}\left[f''(w)\right] \\ &= \mathbb{E}\left[\phi''(w) X X^T\right] \\ &\preceq M \mathbb{E}\left[X X^T\right] \\ &= M\left(\mathrm{Diag}\,(p) - \mathrm{Diag}\,(p)^2 + p p^T\right). \end{aligned}$$

Moreover, we have

$$p p^T = \sqrt{\mathrm{Diag}\,(p)}\left(\sqrt{p}\sqrt{p}^T\right)\sqrt{\mathrm{Diag}\,(p)}. \tag{B.34}$$

As $\sqrt{p}\sqrt{p}^T \preceq \left\|\sqrt{p}\right\|^2 \mathrm{Id} = \sum_{k\in[d]} p^{(k)}\,\mathrm{Id}$, we obtain

$$F''(w) \preceq M\left(1 + \sum_{k\in[d]} p^{(k)}\right)\mathrm{Diag}\,(p)\,.$$

Besides, we have

$$\begin{aligned} F''(w) &\succeq m\left(\mathrm{Diag}\,(p) - \mathrm{Diag}\,(p)^2\right) \\ &\succeq m(1 - p_{\max})\mathrm{Diag}\,(p)\,. \end{aligned}$$

Finally,

$$\begin{aligned} f''(w) &= \phi''(w) X X^T \\ &\preceq M G^2. \end{aligned}$$

As a conclusion, Assumptions 3 are verified for $\mu := m(1-p_{\max})$, $L := M\left(1 + \sum_{k\in[d]} p^{(k)}\right)$ and $R^2 := G^2 M$.

B.4. AdaBatch for SVRG

B.4.1. Mini-batch SVRG

We now only assume that F verifies the following inequalities for $\mu > 0$,

$$\forall w \in \mathbb{R}^d, \quad \mu \preceq F''(w) \quad \text{and} \quad f(w) \preceq L \text{ a.s.} \tag{B.35}$$

Let us take a starting point $y_0 \in \mathbb{R}^d$ and $m \in \mathbb{N}^*$. For all $s = 0, 1, \ldots$ we have $w_{s,0} := y_s$ and for all $n \in [m]$ let us define

$$\begin{aligned} w_{s,n} &:= w_{s,n-1} - \gamma g_{s,n}, \\ y_{s+1} &:= \frac{1}{m}\sum_{n\in[m]} w_{s,n}. \end{aligned}$$

with $g_{s,n}$ the SVRG update based on $(f_{s,n,b})_{b\in[B]}$ i.i.d samples of f. Let us introduce

$$\forall k \in [d], D_{s,n}^{(k)} := \left\{ b \in [B] : k \in \mathcal{S}\left(f'_{s,n,b}\right) \right\}.$$

For any dimension $k \in [d]$ we have

$$g_{s,n}^{(k)} := \frac{1}{B}\left(\sum_{b\in D_{s,n}^{(k)}} f'_{s,n,b}(w_{s,n-1})^{(k)} - f'_{s,n,b}(y_s)^{(k)} + F'(y_s)^{(k)}/p^{(k)} \right).$$

Theorem 7 (Convergence of SVRG with mini-batch)**.** *If Assumptions B.35 are verified and γ verifies*

$$\gamma L\left(1 - \frac{1}{B}\right) < 1,$$

then for all $s > 0$ we have

$$\mathbb{E}\left[F(y_s) - F_*\right] \leq \alpha^s (F(y_0) - F_*) \tag{B.36}$$

where

$$\alpha := \frac{1}{\mu\gamma\left(1 - \frac{\gamma L(3+B)}{2B}\right)m} + \frac{2L\gamma}{B\left(1 - \gamma L\frac{(3+B)}{2B}\right)}. \tag{B.37}$$

Proof. We will reuse the proof technique from [Bubeck et al., 2015, section 6.3]. We introduce $\forall n \in [N], \mathcal{F}_{s,n-1}$ the σ-field generated by $(f_{u,i,b})_{u\in[s],i\in[n-1],b\in[B]}$. For simplicity, we will drop all the s indices. We define $\Gamma_{n,b} := \operatorname{diag}\left(\left(\mathbb{1}_{k\in\mathcal{S}(f_{n,b})}/p^{(k)}\right)_{k\in[d]}\right)$ and $\Gamma := \operatorname{diag}\left(\left(\mathbb{1}_{k\in\mathcal{S}(f)}/p^{(k)}\right)_{k\in[d]}\right)$ so that

$$g_n = \frac{1}{B}\left(\sum_{b\in[B]} f'_{n,b}(w_{n-1}) - f'_{n,b}(y) + \Gamma_{n,b}F'(y_s) \right).$$

One can immediately notice that

$$\mathbb{E}\left[g_n|\mathcal{F}_{n-1}\right] = F'(w_{n-1}).$$

Besides, we have

$$\begin{aligned}
&\mathbb{E}\left[\left\|f'(w_{n-1}) - f'(y) + \Gamma F'(y)\right\|^2 |\mathcal{F}_{n-1}\right] \\
&\leq 2\mathbb{E}\left[\left\|f'(w_{n-1}) - f'(w_*)\right\|^2 + \left\|f'(y) - f'(w_*) + \Gamma F'(y)\right\|^2 |\mathcal{F}_{n-1}\right].
\end{aligned}$$

We reuse the same proof as in [Mania et al., 2015, Lemma 10]. Using the fact that $\mathbb{E}\left[(f'(y)-f'(w_*))^T\Gamma F'(y)|\mathcal{F}_{n-1}\right] = \|F'(y)\|^2_{\mathrm{diag}(p)^{-1}}$ and $\mathbb{E}\left[\|\Gamma F'(y)\|^2|\mathcal{F}_{n-1}\right] = \|F'(y)\|^2_{\mathrm{diag}(p)^{-1}}$ we have

$$\begin{aligned}\mathbb{E}\left[A^{(k)}|\mathcal{F}_{n-1}\right] &= \mathbb{E}\left[\|f'(y)-f'(w_*)\|^2|\mathcal{F}_{n-1}\right] - 2\|F'(y)\|^2_{\mathrm{diag}(p)^{-1}} + \|F'(y)\|^2_{\mathrm{diag}(p)^{-1}} \\ &\leq \mathbb{E}\left[\|f'(y)-f'(w_*)\|^2|\mathcal{F}_{n-1}\right].\end{aligned}$$

It follows that

$$\begin{aligned}\mathbb{E}\left[\|g_n\|^2|\mathcal{F}_{n-1}\right] &= \frac{1}{B}\mathbb{E}\left[\|f'(w_{n-1})-f'(y)+\Gamma F'(y)\|^2|\mathcal{F}_{n-1}\right] + \|F'(w_{t-1})\|^2\left(1-\frac{1}{B}\right) \\ &\leq \frac{2}{B}\left(\mathbb{E}\left[\|f'(w_{n-1})-f'(w_*)\|^2 + \|f'(y)-f'(w_*)\|^2|\mathcal{F}_{n-1}\right]\right) + \|F'(w_{t-1})\|^2\left(1-\frac{1}{B}\right) \\ &\leq \frac{4L}{B}(F(w_{n-1})-F_*+F(y)-F_*) + \|F'(w_{t-1})\|^2\left(1-\frac{1}{B}\right),\end{aligned}$$

using Lemma 6.4 from Bubeck et al. [2015]. We also have

$$\|F'(w_{t-1})\|^2 \leq LF'(w_{n-1})^T(w_{n-1}-w_*),$$

so that

$$\begin{aligned}\mathbb{E}\left[\|w_n-w_*\|^2|\mathcal{F}_{n-1}\right] \leq \|w_{n-1}-w_*\|^2 &- 2\gamma\left(1-\frac{\gamma L}{2}\left(1-\frac{1}{B}\right)\right)F'(w_{n-1})^T(w_{n-1}-w_*) \\ &+ \frac{4\gamma^2 L}{B}(F(w_{n-1})-F_*+F(y)-F_*).\end{aligned}$$

We choose γ so that

$$\gamma L\left(1-\frac{1}{B}\right) < 1,$$

and using that $F'(w_{n-1})^T(w_{n-1}-w_*) \geq F(w_{n-1})-F_*$ we have

$$\mathbb{E}\left[\|w_n-w_*\|^2|\mathcal{F}_{n-1}\right] \leq \|w_{n-1}-w_*\|^2 - \gamma\left(2-\frac{\gamma L(3+B)}{B}\right)(F(w_{n-1})-F_*) + \frac{4\gamma^2 L}{B}(F(y)-F_*).$$

Summing the above inequality for $n \in [m]$ and taking the expectation with respect to $\mathcal{F}_0$,

$$\mathbb{E}\left[\|w_m-w_*\|^2\right] \leq \|y-w_*\| - \gamma\left(2-\frac{\gamma L(3+B)}{B}\right)\mathbb{E}\left[\sum_{n\in[m]}F(w_n)-F_*|\mathcal{F}_0\right] + \frac{4L\gamma^2 m}{B}(F(y)-F_*).$$

Using the strong convexity of F we have $\|w_0-w_*\|^2 \leq \frac{2}{\mu}(F(y)-F_*)$ and finally

$$\mathbb{E}\left[F\left(\frac{1}{m}\sum_{n\in[m]}w_n\right)-F_*|\mathcal{F}_0\right] \leq \left(\frac{1}{\mu\gamma\left(1-\frac{\gamma L(3+B)}{2B}\right)m} + \frac{2L\gamma}{B\left(1-\gamma L\frac{(3+B)}{2B}\right)}\right)(F(y)-F(w_*)).$$

□

One can derive a simplified convergence result when we assume B large enough.

Corollary 1. *If we assume $B \gg 1$, then with $\gamma = \frac{1}{L}$ and $m = \frac{2BL}{\mu(0.9B-4)} \approx \frac{2.2L}{\mu}$, we have*

$$\mathbb{E}\left[F(y_s) - F_*\right] \leq 0.9^s (F(y_0) - F_*).$$

B.4.2. AdaBatch SVRG

Let us now assume that F verify the following inequalities for $\mu > 0$,

$$\forall w \in \mathbb{R}^d, \quad \mu \mathrm{diag}\,(p) \preceq F''(w) \quad \text{and} \quad f(w) \preceq L\mathrm{diag}\,(p) \text{ a.s.} \tag{B.38}$$

We now define for any dimension k such that $D_{s,n}^{(k)} \neq \emptyset$,

$$g_{s,n}^{(k)} := \frac{1}{\left|D_{s,n}^{(k)}\right|} \left(\sum_{b \in D_{s,n}^{(k)}} f'_{s,n,b}(w_{s,n-1})^{(k)} - f'_{s,n,b}(y_s)^{(k)} + F'(y_s)^{(k)}/p^{(k)} \right),$$

and $g_{s,n}^{(k)} := 0$ otherwise.

Theorem 8 (Convergence of SVRG with AdaBatch)**.** *If Assumptions B.38 are verified and γ verifies*

$$\gamma < \frac{L}{2},$$

then for all $s > 0$ we have

$$\mathbb{E}\left[F(y_s) - F_*\right] \leq \alpha^s (F(y_0) - F_*), \tag{B.39}$$

where

$$\alpha := \frac{1}{\mu(1 - (1 - p_{\min})^B)\gamma\,(1 - 2\gamma L)\,m} + \frac{2L\gamma}{1 - 2\gamma L}. \tag{B.40}$$

Proof. We reuse the same proof technique as previously and introduce the same operators Γ and $\Gamma_{n,b}$, again dropping all s indices for simplicity.

We introduce $C := \mathrm{Diag}\left(\frac{1-(1-p)^B}{p}\right)$ and using Lemma B.1 we have

$$\begin{aligned}
\mathbb{E}\left[\|g_n\|_{C^{-1}}^2 \,|\mathcal{F}_{n-1}\right] &\leq \mathbb{E}\left[\left\|f'(w_{n-1}) - f'(y) + \Gamma F'(y)\right\|^2 |\mathcal{F}_{n-1}\right] \\
&\leq 4L(F(w_{n-1}) - F_* + F(y) - F_*),
\end{aligned}$$

using similar arguments as for regular mini-batch. Therefore, we have

$$\begin{aligned}
\mathbb{E}\left[\|w_n - w_*\|_{C^{-1}}^2 \,|\mathcal{F}_{n-1}\right] &\leq \|w_{n-1} - w_*\|_{C^{-1}}^2 - 2\gamma F'(w_{n-1})^T (w_{n-1} - w_*) \\
&\quad + 4\gamma^2 L(F(w_{n-1}) - F_* + F(y) - F_*) \\
&\leq \|w_{n-1} - w_*\|_{C^{-1}}^2 - 2\gamma\,(1 - 2\gamma L)\,(F(w_{n-1}) - F_*) + 4\gamma^2 L\,(F(y) - F_*)\,.
\end{aligned}$$

Summing the above inequality for $n \in [m]$ and taking the expectation with respect to $\mathcal{F}_0$,

$$\mathbb{E}\left[\|w_m - w_*\|^2_{C^{-1}}\right] \leq \|y - w_*\|_{C^{-1}} - 2\gamma\,(1 - 2\gamma L)\,\mathbb{E}\left[\sum_{n\in[m]} F(w_n) - F_*|\mathcal{F}_0\right] + 4L\gamma^2 m(F(y) - F_*).$$

Using the strong convexity of F we have

$$\begin{aligned}\|w_0 - w_*\|_{C^{-1}} &\leq \frac{1}{1-(1-p_{\min})^B}\,\|w_0 - w_*\|^2_{\mathrm{diag}(p)} \\ &\leq \frac{2}{\mu(1-(1-p_{\min})^B)}\,(F(y) - F_*)\,,\end{aligned}$$

and finally

$$\mathbb{E}\left[F\left(\frac{1}{m}\sum_{n\in[m]} w_n\right) - F_*|\mathcal{F}_0\right] \leq \left(\frac{1}{\mu(1-(1-p_{\min})^B)\gamma\,(1-2\gamma L)\,m} + \frac{2L\gamma}{1-2\gamma L}\right)(F(y) - F(w_*)).$$

□

One can derive a simplified convergence result when we assume $p_{\min}$ small enough.

Corollary 2. *If we assume $p_{\min} \ll 1$ so that $(1-(1-p_{\min})^B) \approx Bp_{\min}$ then with $\gamma = \frac{1}{10L}$ and $m = \frac{20L}{Bp_{\min}\mu}$, we have*

$$\mathbb{E}\left[F(y_s) - F_*\right] \leq 0.9^s(F(y_0) - F_*).$$

B.4.3. Comparing the effect of regular mini-batch and AdaBatch for SVRG

If F verifies our sparse convexity condition

$$\forall w \in \mathbb{R}^d, \quad \mu\mathrm{diag}\,(p) \preceq F''(w) \preceq L\mathrm{diag}\,(p)\,, \tag{B.41}$$

we can apply Theorem 7 for

$$\forall w \in \mathbb{R}^d, \quad \mu p_{\min} \preceq F''(w) \preceq L, \tag{B.42}$$

We will assume that the batch size B is large enough (for instance $B = 50$), $p_{\min}$ is small enough so that $1-(1-p_{\min})^B \approx Bp_{\min}$. Then using corollary 1, in order to achieve a linear convergence rate of 0.9 for regular mini-batch we would need to have a number of inner iterations given by

$$m_{\mathrm{mb}} \approx \frac{2.2L}{p_{\min}\mu}.$$

This number is roughly constant with the batch size. However the cost of each single iteration is now B times larger, thus meaning that we would need to process B times more samples before reaching the same accuracy as when $B = 1$.

On the other hand, using Corollary 2, in order to achieve the same rate of convergence, we would require the number of inner iterations to be

$$m_{\text{ab}} \approx \frac{20L}{Bp_{\min}\mu},$$

thus the number of inner iterations is inversely proportionnal to the batch size, which balances perfectly the increased cost of each iteration. We will reach the same accuracy as for $B = 1$ without requiring to process more samples.

It should be noted that using Lemma B.2, it is possible to show that AdaBatch also benefits from variance reduction for the coordinates where $p^{(k)}B$ is large enough. This will depend on the datasets but we have observed such an effect in practice, which allows us to take a larger step-size and further improve convergence.

As for regular SGD, we have noticed experimentally that mini-batch SVRG will become more efficient than AdaBatch when we are close to the optimum. For instance, on datasets that are much smaller than *url* such as *news20* or *spam*, we observed that mini-batch SVRG will perform better than AdaBatch. Therefore, we would advice using AdaBatch for early optimization and regular mini-batch for fine tuning when close to the optimum.

B.5. Experimental results

B.5.1. Experimental results for AdaBatch Wild

We present here the same graphs as in Chapter 4 but for the *spam* and *url* dataset. We also provide the convergence with respect to the number of samples for *news20*. On both datasets, AdaBatch performs competitively with Hogwild! and significantly better than mini-batch SGD, especially when increasing the number of workers and batch-size.

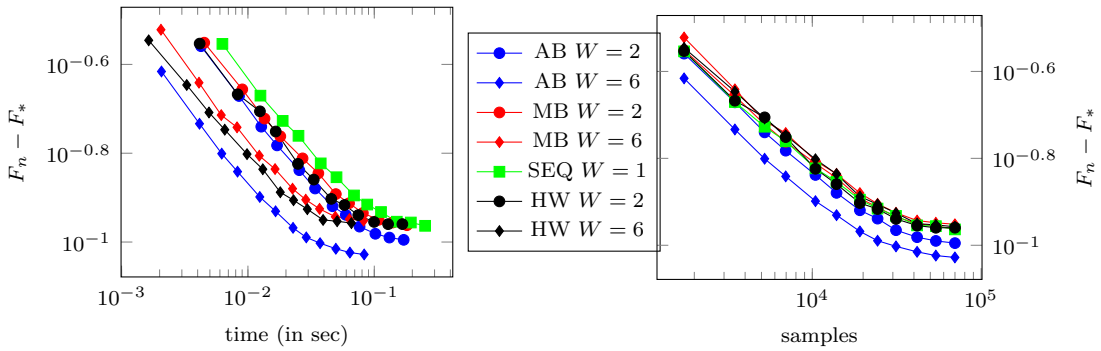


Figure B.1.: Convergence result for *news20*. The error is given either as a function of the wall-clock time (left) or of the number of samples processed (right).

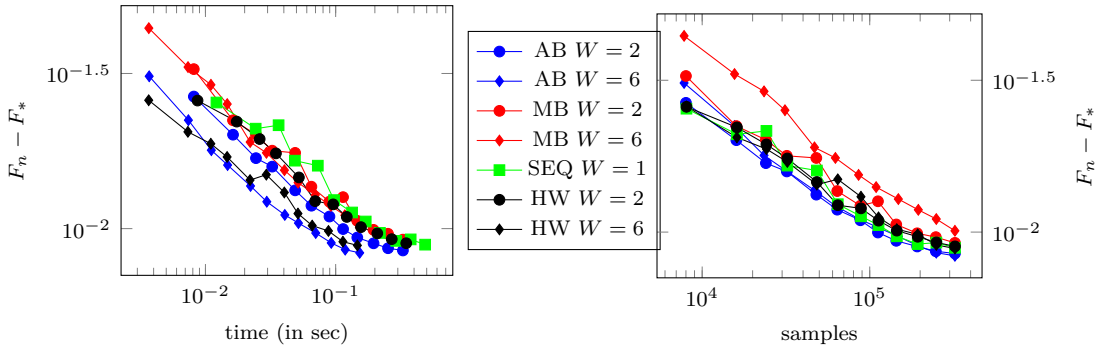


Figure B.2.: Convergence result for *spam*. The error is given either as a function of the wall-clock time (left) or of the number of samples processed (right).

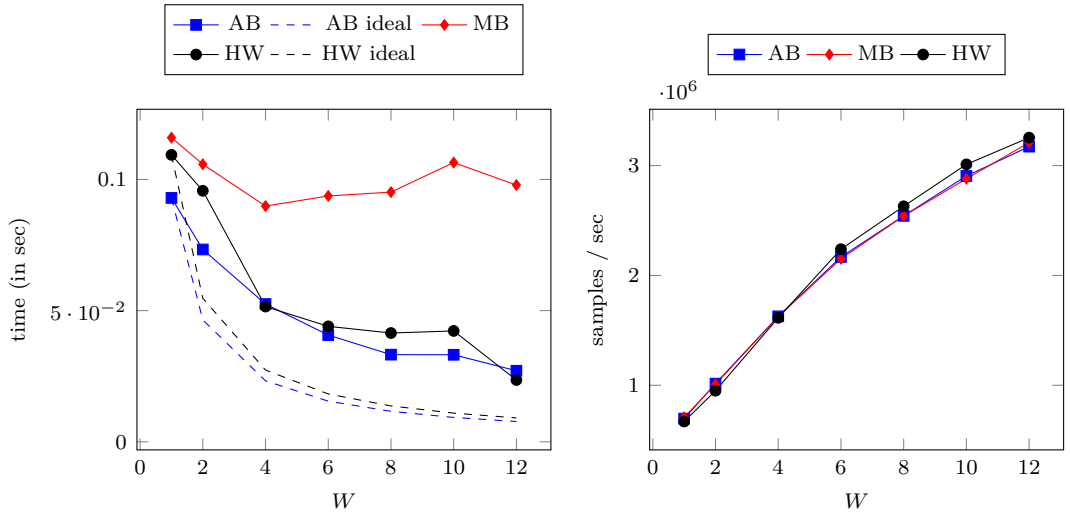


Figure B.3.: On the left, time to achieve a given test error when varying the number of workers for the *spam* dataset. The dashed line represents an ideal speedup dividing the time for 1 worker by W. On the right, number of process sampled per second as a function of W for the *spam* dataset.

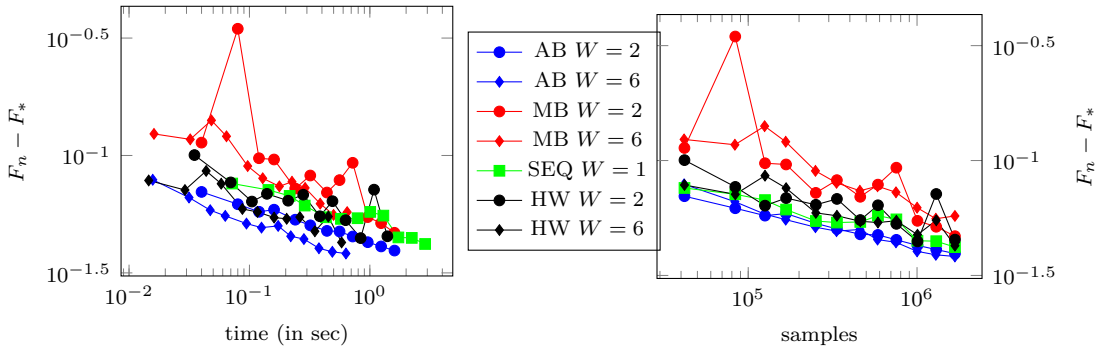


Figure B.4.: Convergence result for *url*. The error is given either as a function of the wall-clock time (left) or of the number of samples processed (right).

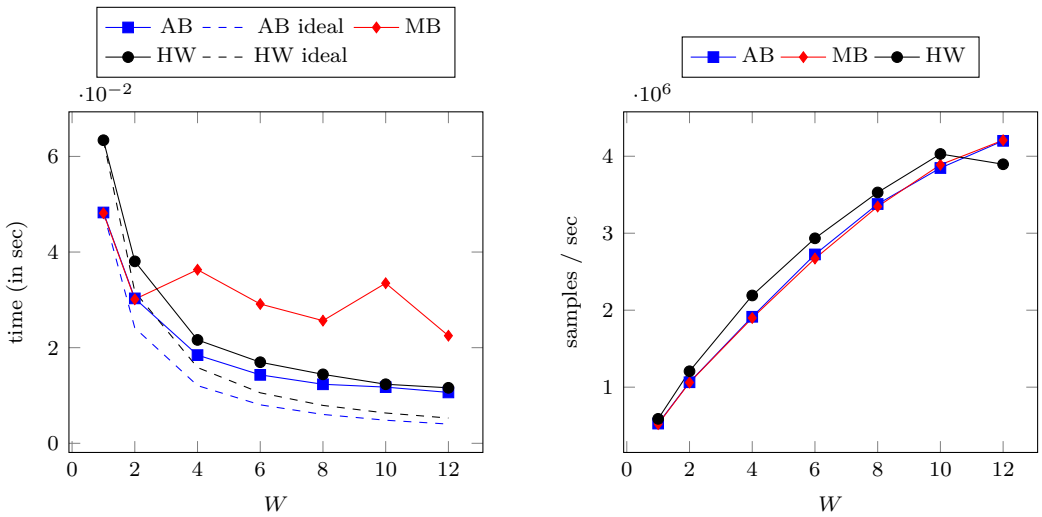
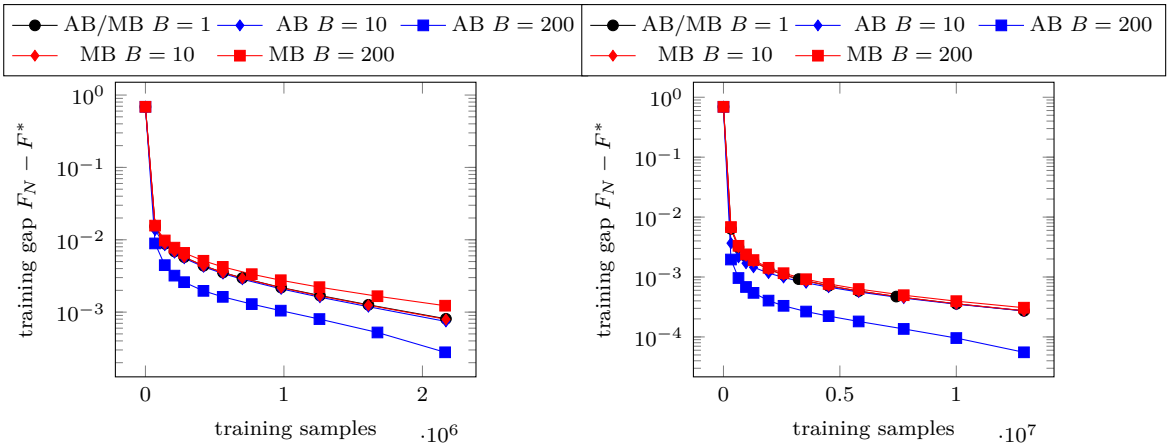


Figure B.5.: On the left, time to achieve a given test error when varying the number of workers on *url*. The dashed line represents an ideal speedup dividing the time for 1 worker by W. On the right, number of process sampled per second as a function of W on *url*.

B.5.2. Experimental results for SVRG

We give a comparison of regular mini-batch and AdaBatch on news20 and spam on figure B.6. The difference is less marked than on the url dataset which we believe is due to the relative simplicity of the optimization problem on such datasets. The L2 regularization was chosen in order to achieve relatively good validation error while retaining the good convergence of SVRG in the strongly convex case.



(a) Comparison of the training gap $F_n - F_*$ for regular mini-batch vs AdaBatch with SVRG on *news20* for the log loss with an L2 penalty of $\frac{10^{-5}}{2} \|w\|^2_{\text{diag}(p)}$.

(b) Comparison of the training gap $F_n - F_*$ for regular mini-batch vs AdaBatch with SVRG on *spam* for the log loss with an L2 penalty of $\frac{10^{-6}}{2} \|w\|^2_{\text{diag}(p)}$.

Figure B.6.: Comparison of regular mini-batch vs AdaBatch with SVRG on *news20* and *spam* dataset.

C. Convergence of adaptive methods with heavy-ball momentum

C.1. Setup and notations

We recall the dynamic system introduced in Section 5.2.3. In the rest of this section, we take an iteration $n \in \mathbb{N}^*$, and when needed, $i \in [d]$ refers to a specific coordinate. Given $x_0 \in \mathbb{R}^d$ our starting point, $m_0 = 0$, and $v_0 = 0$, we define

$$\begin{cases} m_{n,i} & = \beta_1 m_{n-1,i} + \nabla_i f_n(x_{n-1}), \\ v_{n,i} & = \beta_2 v_{n-1,i} + \left(\nabla_i f_n(x_{n-1})\right)^2, \\ x_{n,i} & = x_{n-1,i} - \alpha_n \frac{m_{n,i}}{\sqrt{\epsilon + v_{n,i}}}. \end{cases} \tag{C.1}$$

For Adam, the step size is given by

$$\alpha_n = \alpha(1-\beta_1)\sqrt{\sum_{j=0}^{n-1} \beta_2^j}. \tag{C.2}$$

For Adagrad (potentially extended with heavy-ball momentum), we have $\beta_2 = 1$ and

$$\alpha_n = \alpha(1-\beta_1). \tag{C.3}$$

Notice we include the factor $1-\beta_1$ in the step size rather than in (C.1), as this allows for a more elegant proof. The original Adam algorithm included compensation factors for both β_1 and β_2 [Kingma and Ba, 2014] to correct the initial scale of m and v which are initialized at 0. Adam would be exactly recovered by replacing (C.2) with

$$\alpha_n = \alpha \frac{\sqrt{\sum_{j=0}^{n-1} \beta_2^j}}{\sum_{j=0}^{n-1} \beta_1^j}. \tag{C.4}$$

However, the denominator $\sum_{j=0}^{n-1} \beta_1^j$ potentially makes $(\alpha_n)_{n\in\mathbb{N}^*}$ non monotonic, which complicates the proof. Thus, we instead replace the denominator by its limit value for $n \to \infty$. This has little practical impact as (i) early iterates are noisy because v is

averaged over a small number of gradients, so making smaller step can be more stable, (ii) for $\beta_1 = 0.9$ [Kingma and Ba, 2014], (C.2) differs from (C.4) only for the first few tens of iterations. We could have replaced the numerator by its limit value but chose not to as: (i) β_2 is typically much closer to 1 than β_1, thus our update rule would have differed from Adam for longer (ii) without this correction, the step size is either too large early on or too small at the end.

Throughout the proof we note $\mathbb{E}_{n-1}[\cdot]$ the conditional expectation with respect to $f_1, \dots, f_{n-1}$. In particular, x_{n-1}, v_{n-1} is deterministic knowing $f_1, \dots, f_{n-1}$. We introduce

$$G_n = \nabla F(x_{n-1}) \quad \text{and} \quad g_n = \nabla f_n(x_{n-1}). \tag{C.5}$$

Like in Section 5.6.2, we introduce the update $u_n \in \mathbb{R}^d$, as well as the update without heavy-ball momentum $U_n \in \mathbb{R}^d$:

$$u_{n,i} = \frac{m_{n,i}}{\sqrt{\epsilon + v_{n,i}}} \quad \text{and} \quad U_{n,i} = \frac{g_{n,i}}{\sqrt{\epsilon + v_{n,i}}}. \tag{C.6}$$

For any $k \in \mathbb{N}$ with $k < n$, we define $\tilde{v}_{n,k} \in \mathbb{R}^d$ by

$$\tilde{v}_{n,k,i} = \beta_2^k v_{n-k,i} + \mathbb{E}_{n-k-1}\left[\sum_{j=n-k+1}^{n} \beta_2^{n-j} g_{j,i}^2\right], \tag{C.7}$$

i.e. the contribution from the k last gradients are replaced by their expected value for know values of $f_1, \dots, f_{n-k-1}$. For $k = 1$, we recover the same definition as in (5.26).

C.2. Results

For any total number of iterations $N \in \mathbb{N}^*$, we define τ_N a random index with value in $\{0, \dots, N-1\}$, verifying

$$\forall j \in \mathbb{N}, j < N, \mathbb{P}[\tau = j] \propto 1 - \beta_1^{N-j}. \tag{C.8}$$

If $\beta_1 = 0$, this is equivalent to sampling τ uniformly in $\{0, \dots, N-1\}$. If $\beta_1 > 0$, the last few $\frac{1}{1-\beta_1}$ iterations are sampled rarely, and all iterations older than a few times that number are sampled almost uniformly. We bound the expected squared norm of the total gradient at iteration τ, which is standard for non convex stochastic optimization [Ghadimi and Lan, 2013].

Note that like in previous works, the bound worsen as β_1 increases, with a dependency in $(1-\beta_1)$ which is $(1-\beta_1)^{-1}$. This is a significant improvement over the existing bound for Adagrad with heavy-ball momentum, which scale as $(1-\beta_1)^{-3}$ [Zou et al., 2019b], or the best known bound for Adam which scale as $(1-\beta_1)^{-5}$ [Zou et al., 2019a].

Technical lemmas to prove the following theorems are introduced in Section C.4, while the proof of Theorems 9 and 10 are provided in Section C.5.

Theorem 9 (Convergence of Adam with momentum)**.** *Given the hypothesis introduced in Section 5.2.3, the iterates x_n defined in Section 5.2.2 with hyper-parameters verifying $0 < \beta_2 < 1$, $\alpha_n = \alpha(1-\beta_1)\sqrt{\sum_{j=0}^{n-1}\beta_2^j}$ with $\alpha > 0$ and $0 < \beta_1 < \beta_2$, we have for any $N \in \mathbb{N}^*$ such that $N > \frac{\beta_1}{1-\beta_1}$, taking τ defined by* (C.8)*,*

$$\mathbb{E}\left[\|\nabla F(x_\tau)\|^2\right] \leq 2R\frac{F(x_0)-F_*}{\alpha\tilde{N}} + \frac{E}{\tilde{N}}\left(\ln\left(1+\frac{R^2}{\epsilon(1-\beta_2)}\right) - N\ln(\beta_2)\right), \tag{C.9}$$

with

$$\tilde{N} = N - \frac{\beta_1}{1-\beta_1}, \tag{C.10}$$

and

$$E = \frac{\alpha dRL(1-\beta_1)}{(1-\beta_1/\beta_2)(1-\beta_2)} + \frac{12dR^2\sqrt{1-\beta_1}}{(1-\beta_1/\beta_2)^{3/2}\sqrt{1-\beta_2}} + \frac{2\alpha^2 dL^2\beta_1}{(1-\beta_1/\beta_2)(1-\beta_2)^{3/2}}. \tag{C.11}$$

Theorem 10 (Convergence of Adagrad with momentum)**.** *Given the hypothesis introduced in Section 5.2.3, the iterates x_n defined in Section 5.2.2 with hyper-parameters verifying $\beta_2 = 1$, $\alpha_n = (1-\beta_1)\alpha$ with $\alpha > 0$ and $0 < \beta_1 < 1$, we have for any $N \in \mathbb{N}^*$ such that $N > \frac{\beta_1}{1-\beta_1}$, taking τ as defined by* (C.8)*,*

$$\mathbb{E}\left[\|\nabla F(x_\tau)\|^2\right] \leq 2R\sqrt{N}\frac{F(x_0)-F_*}{\alpha\tilde{N}} + \frac{\sqrt{N}}{\tilde{N}}\left(\alpha dRL + \frac{12dR^2}{1-\beta_1} + \frac{2\alpha^2 dL^2\beta_1}{1-\beta_1}\right)\ln\left(1+\frac{NR^2}{\epsilon}\right), \tag{C.12}$$

with

$$\tilde{N} = N - \frac{\beta_1}{1-\beta_1}. \tag{C.13}$$

C.3. Analysis of the results with momentum

First notice that taking $\beta_1 \to 0$ in Theorems 9 and 10, we almost recover the same result as stated in 1 and 2, only losing on the term $4dR^2$ which becomes $12dR^2$.

Simplified expressions with momentum Assuming $N \gg \frac{\beta_1}{1-\beta_1}$ and $\beta_1/\beta_2 \approx \beta_1$, which is verified for typical values of β_1 and β_2 [Kingma and Ba, 2014], it is possible to simplify

the bound for Adam (C.9) as

$$\mathbb{E}\left[\|\nabla F(x_\tau)\|^2\right] \lessapprox 2R\frac{F(x_0)-F_*}{\alpha N} + \left(\frac{\alpha dRL}{1-\beta_2} + \frac{12dR^2}{(1-\beta_1)\sqrt{1-\beta_2}} + \frac{2\alpha^2 dL^2\beta_1}{(1-\beta_1)(1-\beta_2)^{3/2}}\right)\left(\frac{1}{N}\ln\left(1+\frac{R^2}{\epsilon(1-\beta_2)}\right) - \ln(\beta_2)\right). \tag{C.14}$$

Similarly, if we assume $N \gg \frac{\beta_1}{1-\beta_1}$, we can simplify the bound for Adagrad (C.12) as

$$\mathbb{E}\left[\|\nabla F(x_\tau)\|^2\right] \lessapprox 2R\frac{F(x_0)-F_*}{\alpha\sqrt{N}} + \frac{1}{\sqrt{N}}\left(\alpha dRL + \frac{12dR^2}{1-\beta_1} + \frac{2\alpha^2 dL^2\beta_1}{1-\beta_1}\right)\ln\left(1+\frac{NR^2}{\epsilon}\right), \tag{C.15}$$

Optimal finite horizon Adam is still Adagrad We can perform the same finite horizon analysis as in Section 5.5.3. If we take $\alpha = \frac{\tilde{\alpha}}{\sqrt{N}}$ and $\beta_2 = 1 - 1/N$, then (C.14) simplifies to

$$\mathbb{E}\left[\|\nabla F(x_\tau)\|^2\right] \lessapprox 2R\frac{F(x_0)-F_*}{\tilde{\alpha}\sqrt{N}} + \frac{1}{\sqrt{N}}\left(\tilde{\alpha} dRL + \frac{12dR^2}{1-\beta_1} + \frac{2\tilde{\alpha}^2 dL^2\beta_1}{1-\beta_1}\right)\left(\ln\left(1+\frac{NR^2}{\epsilon}\right)+1\right). \tag{C.16}$$

The term $(1-\beta_2)^{3/2}$ in the denominator in (C.14) is indeed compensated by the α^2 in the numerator and we again recover the proper $\ln(N)/\sqrt{N}$ convergence rate, which matches (C.15) up to a $+1$ term next to the log.

C.4. Technical lemmas

We first need an updated version of 5.1 that includes momentum.

Lemma C.1 (Adaptive update with momentum approximately follows a descent direction)**.** *Given $x_0 \in \mathbb{R}^d$, the iterates defined by the system* (C.1) *for $(\alpha_j)_{j\in\mathbb{N}^*}$ that is non-decreasing, and under the conditions* (5.5)*,* (5.7)*, and* (5.8)*, as well as $0 \le \beta_1 < \beta_2 \le 1$, we have for all iterations $n \in \mathbb{N}^*$,*

$$\mathbb{E}\left[\sum_{i\in[d]} G_{n,i}\frac{m_{n,i}}{\sqrt{\epsilon+v_{n,i}}}\right] \ge \frac{1}{2}\left(\sum_{i\in[d]}\sum_{k=0}^{n-1}\beta_1^k\mathbb{E}\left[\frac{G_{n-k,i}^2}{\sqrt{\epsilon+\tilde{v}_{n,k+1,i}}}\right]\right) - \frac{\alpha_n^2 L^2}{4R}\sqrt{1-\beta_1}\left(\sum_{l=0}^{n-1}\|u_{n-l}\|_2^2\sum_{k=l}^{n-1}\beta_1^l\sqrt{l}\right) - \frac{3R}{\sqrt{1-\beta_1}}\left(\sum_{k=0}^{n-1}\left(\frac{\beta_1}{\beta_2}\right)^k\sqrt{k+1}\,\|U_{n-k}\|_2^2\right). \tag{C.17}$$

Proof. We use multiple times (5.30) in this proof, which we repeat here for convenience,

$$\forall \lambda > 0,\, x, y \in \mathbb{R}, xy \leq \frac{\lambda}{2} x^2 + \frac{y^2}{2\lambda}. \tag{C.18}$$

Let us take an iteration $n \in \mathbb{N}^*$ for the duration of the proof. We have

$$\begin{aligned}\sum_{i\in[d]} G_{n,i} \frac{m_{n,i}}{\sqrt{\epsilon + v_{n,i}}} &= \sum_{i\in[d]} \sum_{k=0}^{n-1} \beta_1^k G_{n,i} \frac{g_{n-k,i}}{\sqrt{\epsilon + v_{n,i}}} \\ &= \underbrace{\sum_{i\in[d]} \sum_{k=0}^{n-1} \beta_1^k G_{n-k,i} \frac{g_{n-k,i}}{\sqrt{\epsilon + v_{n,i}}}}_{A} + \underbrace{\sum_{i\in[d]} \sum_{k=0}^{n-1} \beta_1^k \left(G_{n,i} - G_{n-k,i}\right) \frac{g_{n-k,i}}{\sqrt{\epsilon + v_{n,i}}}}_{B},\end{aligned} \tag{C.19}$$

Let us now take an index $0 \leq k \leq n-1$. We show that the contribution of past gradients G_{n-k} and g_{n-k} due to the heavy-ball momentum can be controlled thanks to the decay term β_1^k. Let us first have a look at B. Using (C.18) with

$$\lambda = \frac{\sqrt{1-\beta_1}}{2R\sqrt{k+1}},\; x = \left|G_{n,i} - G_{n-k,i}\right|,\; y = \frac{|g_{n-k,i}|}{\sqrt{\epsilon + v_{n,i}}},$$

we have

$$|B| \leq \sum_{i\in[d]} \sum_{k=0}^{n-1} \beta_1^k \left(\frac{\sqrt{1-\beta_1}}{4R\sqrt{k+1}} \left(G_{n,i} - G_{n-k,i}\right)^2 + \frac{R}{\sqrt{1-\beta_1}} \sqrt{k+1} \frac{g_{n-k,i}^2}{\epsilon + v_{n,i}} \right). \tag{C.20}$$

Notice first that for any dimension $i \in [d]$, $\epsilon + v_{n,i} \geq \epsilon + \beta_2^k v_{n-k,i} \geq \beta_2^k (\epsilon + v_{n-k,i})$, so that

$$\frac{g_{n-k,i}^2}{\epsilon + v_{n,i}} \leq \frac{1}{\beta_2^k} U_{n-k,i}^2 \tag{C.21}$$

Besides, using the L-smoothness of F given by (5.8), we have

$$\begin{aligned}\left\|G_n - G_{n-k}\right\|_2^2 &\leq L^2 \left\|x_{n-1} - x_{n-k-1}\right\|_2^2 \\ &= L^2 \left\| \sum_{l=1}^{k} \alpha_{n-l} u_{n-l} \right\|_2^2 \qquad \text{(C.22)} \\ &\leq \alpha_n^2 L^2 k \sum_{l=1}^{k} \left\|u_{n-l}\right\|_2^2, \qquad \text{(C.23)}\end{aligned}$$

using Jensen inequality and the fact that α_n is non-decreasing. Injecting (C.21) and

(C.23) into (C.20), we obtain

$$|B| \leq \left(\sum_{k=0}^{n-1} \frac{\alpha_n^2 L^2}{4R}\sqrt{1-\beta_1}\beta_1^k\sqrt{k}\sum_{l=1}^{k}\|u_{n-l}\|_2^2\right) + \left(\sum_{k=0}^{n-1}\frac{R}{\sqrt{1-\beta_1}}\left(\frac{\beta_1}{\beta_2}\right)^k\sqrt{k+1}\,\|U_{n-k}\|_2^2\right)$$
$$= \frac{\alpha_n^2 L^2}{4R}\sqrt{1-\beta_1}\left(\sum_{l=0}^{n-1}\|u_{n-l}\|_2^2\sum_{k=l}^{n-1}\beta_1^l\sqrt{l}\right) + \frac{R}{\sqrt{1-\beta_1}}\left(\sum_{k=0}^{n-1}\left(\frac{\beta_1}{\beta_2}\right)^k\sqrt{k+1}\,\|U_{n-k}\|_2^2\right). \tag{C.24}$$

Now going back to the A term in (C.19), we will study the main term of the summation, i.e. for $i \in [d]$ and $k < n$

$$\mathbb{E}\left[G_{n-k,i}\frac{g_{n-k,i}}{v_{n,i}}\right] = \mathbb{E}\left[\nabla_i F(x_{n-k-1})\frac{\nabla_i f_{n-k}(x_{n-k-1})}{\sqrt{\epsilon+v_{n,i}}}\right]. \tag{C.25}$$

Notice that we could almost apply Lemma 5.1 to it, except that we have $v_{n,i}$ in the denominator instead of $v_{n-k,i}$. Thus we will need to extend the proof to decorrelate more terms. We will further drop indices in the rest of the proof, noting $G = G_{n-k,i}$, $g = g_{n-k,i}$, $\tilde{v} = \tilde{v}_{n,k+1,i}$ and $v = v_{n,i}$. Finally, let us note

$$\delta^2 = \sum_{j=n-k}^{n}\beta_2^{n-j}g_{j,i} \qquad \text{and} \qquad r^2 = \mathbb{E}_{n-k-1}\left[\delta^2\right]. \tag{C.26}$$

In particular we have $\tilde{v} - v = r^2 - \delta^2$. With our new notations, we can rewrite (C.25) as

$$\mathbb{E}\left[G\frac{g}{\sqrt{\epsilon+v}}\right] = \mathbb{E}\left[G\frac{g}{\sqrt{\epsilon+\tilde{v}}} + Gg\left(\frac{1}{\sqrt{\epsilon+v}} - \frac{1}{\sqrt{\epsilon+\tilde{v}}}\right)\right]$$
$$= \mathbb{E}\left[\mathbb{E}_{n-k-1}\left[G\frac{g}{\sqrt{\epsilon+\tilde{v}}}\right] + Gg\frac{r^2-\delta^2}{\sqrt{\epsilon+v}\sqrt{\epsilon+\tilde{v}}(\sqrt{\epsilon+v}+\sqrt{\epsilon+\tilde{v}})}\right]$$
$$= \mathbb{E}\left[\frac{G^2}{\sqrt{\epsilon+\tilde{v}}}\right] + \mathbb{E}\left[\underbrace{Gg\frac{r^2-\delta^2}{\sqrt{\epsilon+v}\sqrt{\epsilon+\tilde{v}}(\sqrt{\epsilon+v}+\sqrt{\epsilon+\tilde{v}})}}_{C}\right]. \tag{C.27}$$

We first focus on C:

$$|C| \leq \underbrace{|Gg|\frac{r^2}{\sqrt{\epsilon+v}(\epsilon+\tilde{v})}}_{\kappa} + \underbrace{|Gg|\frac{\delta^2}{(\epsilon+v)\sqrt{\epsilon+\tilde{v}}}}_{\rho},$$

due to the fact that $\sqrt{\epsilon+v}+\sqrt{\epsilon+\tilde{v}} \geq \max(\sqrt{\epsilon+v}, \sqrt{\epsilon+\tilde{v}})$ and $\left|r^2-\delta^2\right| \leq r^2+g^2$.

Applying (C.18) to κ with

$$\lambda = \frac{\sqrt{1-\beta_1}\sqrt{\epsilon+\tilde{v}}}{2},\ x = \frac{|G|}{\sqrt{\epsilon+\tilde{v}}},\ y = \frac{|g|\,r^2}{\sqrt{\epsilon+\tilde{v}}\sqrt{\epsilon+v}},$$

we obtain

$$\kappa \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{v}}} + \frac{1}{\sqrt{1-\beta_1}} \frac{g^2 r^4}{(\epsilon+\tilde{v})^{3/2}(\epsilon+v)}.$$

Given that $\epsilon + \tilde{v} \geq r^2$ and taking the conditional expectation, we can simplify as

$$\mathbb{E}_{n-k-1}[\kappa] \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{v}}} + \frac{1}{\sqrt{1-\beta_1}} \frac{r^2}{\sqrt{\epsilon+\tilde{v}}} \mathbb{E}_{n-k-1}\left[\frac{g^2}{\epsilon+v}\right]. \tag{C.28}$$

Now turning to ρ, we use (C.18) with

$$\lambda = \frac{\sqrt{1-\beta_1}\sqrt{\epsilon+\tilde{v}}}{2r^2},\ x = \frac{|G\delta|}{\sqrt{\epsilon+\tilde{v}}},\ y = \frac{|\delta g|}{\epsilon+v},$$

we obtain

$$\rho \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{v}}} \frac{\delta^2}{r^2} + \frac{1}{\sqrt{1-\beta_1}} \frac{r^2}{\sqrt{\epsilon+\tilde{v}}} \frac{g^2\delta^2}{(\epsilon+v)^2}. \tag{C.29}$$

Given that $\epsilon + v \geq \delta^2$, and $\mathbb{E}_{n-k-1}\left[\frac{\delta^2}{r^2}\right] = 1$, we obtain after taking the conditional expectation,

$$\mathbb{E}_{n-k-1}[\rho] \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{v}}} + \frac{1}{\sqrt{1-\beta_1}} \frac{r^2}{\sqrt{\epsilon+\tilde{v}}} \mathbb{E}_{n-k-1}\left[\frac{g^2}{\epsilon+v}\right]. \tag{C.30}$$

Notice that in C.29, we possibly divide by zero. It suffice to notice that if $r^2 = 0$ then $\delta^2 = 0$ a.s. so that $\rho = 0$ and (C.30) is still verified. Summing (C.28) and (C.30), we get

$$\mathbb{E}_{n-k-1}[|C|] \leq \frac{G^2}{2\sqrt{\epsilon+\tilde{v}}} + \frac{2}{\sqrt{1-\beta_1}} \frac{r^2}{\sqrt{\epsilon+\tilde{v}}} \mathbb{E}_{n-k-1}\left[\frac{g^2}{\epsilon+v}\right]. \tag{C.31}$$

Given that $r \leq \sqrt{\epsilon+\tilde{v}}$ by definition of $\tilde{v}$, and that using (5.7), $r \leq \sqrt{k+1}R$, we have, reintroducing the indices we had dropped

$$\mathbb{E}_{n-k-1}[|C|] \leq \frac{G_{n-k,i}^2}{2\sqrt{\epsilon+\tilde{v}_{n,k,i}}} + \frac{2R}{\sqrt{1-\beta_1}}\sqrt{k+1}\mathbb{E}_{n-k-1}\left[\frac{g_{n-k,i}^2}{\epsilon+v_{n,i}}\right]. \tag{C.32}$$

Taking the complete expectation and using that by definition $\epsilon + v_{n,i} \geq \epsilon + \beta_2^k v_{n-k,i} \geq \beta_2^k(\epsilon + v_{n-k,i})$ we get

$$\mathbb{E}[|C|] \leq \frac{1}{2}\mathbb{E}\left[\frac{G_{n-k,i}^2}{\sqrt{\epsilon+\tilde{v}_{n,k,i}}}\right] + \frac{2R}{\sqrt{1-\beta_1}\beta_2^k}\sqrt{k+1}\mathbb{E}\left[\frac{g_{n-k,i}^2}{\epsilon+v_{n-k,i}}\right]. \tag{C.33}$$

Injecting (C.33) into (C.27) gives us

$$\begin{aligned}\mathbb{E}\left[A\right] &\geq \sum_{i\in[d]}\sum_{k=0}^{n-1}\beta_1^k\left(\mathbb{E}\left[\frac{G_{n-k,i}^2}{\sqrt{\epsilon+\tilde{v}_{n,k,i}}}\right]-\left(\frac{1}{2}\mathbb{E}\left[\frac{G_{n-k,i}^2}{\sqrt{\epsilon+\tilde{v}_{n,k,i}}}\right]+\frac{2R}{\sqrt{1-\beta_1}\beta_2^k}\sqrt{k+1}\mathbb{E}\left[\frac{g_{n-k,i}^2}{\epsilon+v_{n-k,i}}\right]\right)\right)\\ &= \frac{1}{2}\left(\sum_{i\in[d]}\sum_{k=0}^{n-1}\beta_1^k\mathbb{E}\left[\frac{G_{n-k,i}^2}{\sqrt{\epsilon+\tilde{v}_{n,k,i}}}\right]\right)-\frac{2R}{\sqrt{1-\beta_1}}\left(\sum_{i\in[d]}\sum_{k=0}^{n-1}\left(\frac{\beta_1}{\beta_2}\right)^k\sqrt{k+1}\mathbb{E}\left[\left\|U_{n-k}\right\|_2^2\right]\right).\end{aligned} \tag{C.34}$$

Injecting (C.34) and (C.24) into (C.19) finishes the proof.

□

Similarly, we will need an updated version of 5.2.

Lemma C.2 (sum of ratios of the square of a decayed sum and a decayed sum of square)**.** *We assume we have $0 < \beta_2 \leq 1$ and $0 < \beta_1 < \beta_2$, and a sequence of real numbers $(a_n)_{n\in\mathbb{N}^*}$. We define $b_n = \sum_{j=1}^n \beta_2^{n-j}a_j^2$ and $c_n = \sum_{j=1}^n \beta_1^{n-j}a_j$. Then we have*

$$\sum_{j=1}^n \frac{c_j^2}{\epsilon+b_j} \leq \frac{1}{(1-\beta_1)(1-\beta_1/\beta_2)}\left(\ln\left(1+\frac{b_n}{\epsilon}\right)-n\ln(\beta_2)\right). \tag{C.35}$$

Proof. Now let us take $j\in\mathbb{N}^*$, $j\leq n$, we have using Jensen inequality

$$c_j^2 \leq \frac{1}{1-\beta_1}\sum_{l=1}^j \beta_1^{j-l}a_l^2,$$

so that

$$\frac{c_j^2}{\epsilon+b_j} \leq \frac{1}{1-\beta_1}\sum_{l=1}^j \beta_1^{j-l}\frac{a_l^2}{\epsilon+b_j}.$$

Given that for $l\in[j]$, we have by definition $\epsilon+b_j \geq \epsilon+\beta_2^{j-l}b_l \geq \beta_2^{j-l}(\epsilon+b_j)$, we get

$$\frac{c_j^2}{\epsilon+b_j} \leq \frac{1}{1-\beta_1}\sum_{l=1}^j \left(\frac{\beta_1}{\beta_2}\right)^{j-l}\frac{a_l^2}{\epsilon+b_l}. \tag{C.36}$$

Thus, when summing over all $j\in[n]$, we get

$$\begin{aligned}\sum_{j=1}^n \frac{c_j^2}{\epsilon+b_j} &\leq \frac{1}{1-\beta_1}\sum_{j=1}^n\sum_{l=1}^j\left(\frac{\beta_1}{\beta_2}\right)^{j-l}\frac{a_l^2}{\epsilon+b_l}\\ &= \frac{1}{1-\beta_1}\sum_{l=1}^n\frac{a_l^2}{\epsilon+b_l}\sum_{j=l}^n\left(\frac{\beta_1}{\beta_2}\right)^{j-l}\\ &\leq \frac{1}{(1-\beta_1)(1-\beta_1/\beta_2)}\sum_{l=1}^n\frac{a_l^2}{\epsilon+b_l}.\end{aligned} \tag{C.37}$$

Applying Lemma 5.2, we obtain (C.35).

□

We also need two technical lemmas on the sum of series.

Lemma C.3 (sum of a geometric term times a square root)**.** *Given* $0 < a < 1$ *and* $Q \in \mathbb{N}$*, we have,*

$$\sum_{q=0}^{Q-1} a^q \sqrt{q+1} \leq \frac{1}{1-a}\left(1 + \frac{\sqrt{\pi}}{2\sqrt{-\ln(a)}}\right) \leq \frac{2}{(1-a)^{3/2}}. \tag{C.38}$$

Proof. We first need to study the following integral:

$$\begin{aligned}
\int_0^\infty \frac{a^x}{2\sqrt{x}}\mathrm{d}x &= \int_0^\infty \frac{\mathrm{e}^{\ln(a)x}}{2\sqrt{x}}\mathrm{d}x \quad \text{, then introducing } y = \sqrt{x}, \\
&= \int_0^\infty \mathrm{e}^{\ln(a)y^2}\mathrm{d}y \quad \text{, then introducing } u = \sqrt{-2\ln(a)}y, \\
&= \frac{1}{\sqrt{-2\ln(a)}}\int_0^\infty \mathrm{e}^{-u^2/2}\mathrm{d}u \\
\int_0^\infty \frac{a^x}{2\sqrt{x}}\mathrm{d}x &= \frac{\sqrt{\pi}}{2\sqrt{-\ln(a)}},
\end{aligned} \tag{C.39}$$

where we used the classical integral of the standard Gaussian density function.

Let us now introduce A_Q:

$$A_Q = \sum_{q=0}^{Q-1} a^q \sqrt{q+1},$$

then we have

$$\begin{aligned}
A_Q - aA_Q &= \sum_{q=0}^{Q-1} a^q\sqrt{q+1} - \sum_{q=1}^{Q} a^q\sqrt{q} \quad \text{, then using the concavity of } \sqrt{\cdot}, \\
&\leq 1 - a^Q\sqrt{Q} + \sum_{q=1}^{Q-1}\frac{a^q}{2\sqrt{q}} \\
&\leq 1 + \int_0^\infty \frac{a^x}{2\sqrt{x}}\mathrm{d}x \\
(1-a)A_Q &\leq 1 + \frac{\sqrt{\pi}}{2\sqrt{-\ln(a)}},
\end{aligned}$$

where we used (C.39). Given that $\sqrt{-\ln(a)} \geq \sqrt{1-a}$ we obtain (C.38). □

Lemma C.4 (sum of a geometric term times roughly a power 3/2)**.** *Given* $0 < a < 1$ *and* $Q \in \mathbb{N}$*, we have,*

$$\sum_{q=0}^{Q-1} a^q\sqrt{q}(q+1) \leq \frac{4a}{(1-a)^{5/2}}. \tag{C.40}$$

Proof. Let us introduce A_Q:

$$A_Q = \sum_{q=0}^{Q-1} a^q \sqrt{q}(q+1),$$

then we have

$$\begin{aligned}
A_Q - aA_Q &= \sum_{q=0}^{Q-1} a^q \sqrt{q}(q+1) - \sum_{q=1}^{Q} a^q \sqrt{q-1} q \\
&\leq \sum_{q=1}^{Q-1} a^q \sqrt{q} \left((q+1) - \sqrt{q}\sqrt{q-1} \right) \\
&\leq \sum_{q=1}^{Q-1} a^q \sqrt{q} \left((q+1) - (q-1) \right) \\
&\leq 2 \sum_{q=1}^{Q-1} a^q \sqrt{q} \\
&= 2a \sum_{q=0}^{Q-2} a^q \sqrt{q+1} \quad \text{, then using Lemma C.3,} \\
(1-a)A_Q &\leq \frac{4a}{(1-a)^{3/2}}.
\end{aligned}$$

□

C.5. Proof of Adam and Adagrad with momentum

Common part of the proof Let us a take an iteration $n \in \mathbb{N}^*$. Using the smoothness of F defined in (5.8), we have

$$F(x_n) \leq F(x_{n-1}) - \alpha_n G_n^T u_n + \frac{\alpha_n^2 L}{2} \|u_n\|_2^2 .$$

Taking the full expectation and using Lemma C.1,

$$\begin{aligned}
\mathbb{E}\left[F(x_n)\right] \leq \mathbb{E}\left[F(x_{n-1})\right] - \frac{\alpha_n}{2} \left(\sum_{i \in [d]} \sum_{k=0}^{n-1} \beta_1^k \mathbb{E}\left[\frac{G_{n-k,i}^2}{2\sqrt{\epsilon + \tilde{v}_{n,k+1,i}}} \right] \right) + \frac{\alpha_n^2 L}{2} \mathbb{E}\left[\|u_n\|_2^2 \right] \\
+ \frac{\alpha_n^3 L^2}{4R} \sqrt{1-\beta_1} \left(\sum_{l=0}^{n-1} \|u_{n-l}\|_2^2 \sum_{k=l}^{n-1} \beta_1^l \sqrt{l} \right) + \frac{3\alpha_n R}{\sqrt{1-\beta_1}} \left(\sum_{k=0}^{n-1} \left(\frac{\beta_1}{\beta_2} \right)^k \sqrt{k+1} \, \|U_{n-k}\|_2^2 \right).
\end{aligned} \tag{C.41}$$

Notice that because of the bound on the ℓ_∞ norm of the stochastic gradients at the iterates (5.7), we have for any $k \in \mathbb{N}$, $k < n$, and any coordinate $i \in [d]$, $\sqrt{\epsilon + \tilde{v}_{n,k,i}} \leq R\sqrt{\sum_{j=0}^{n-1} \beta_2^j}$. Introducing $\Omega_n = \sqrt{\sum_{j=0}^{n-1} \beta_2^j}$, we have

$$\mathbb{E}\left[F(x_n)\right] \leq \mathbb{E}\left[F(x_{n-1})\right] - \frac{\alpha_n}{2R\Omega_n} \sum_{k=0}^{n-1} \beta_1^k \mathbb{E}\left[\|G_{n-k}\|_2^2\right] + \frac{\alpha_n^2 L}{2} \mathbb{E}\left[\|u_n\|_2^2\right] + \frac{\alpha_n^3 L^2}{4R} \sqrt{1-\beta_1} \left(\sum_{l=0}^{n-1} \|u_{n-l}\|_2^2 \sum_{k=l}^{n-1} \beta_1^l \sqrt{l} \right) + \frac{3\alpha_n R}{\sqrt{1-\beta_1}} \left(\sum_{k=0}^{n-1} \left(\frac{\beta_1}{\beta_2}\right)^k \sqrt{k+1}\, \|U_{n-k}\|_2^2 \right). \tag{C.42}$$

Now summing over all iterations $n \in [N]$ for $N \in \mathbb{N}^*$, and using that for both Adam (C.2) and Adagrad (C.3), α_n is non-decreasing, as well the fact that F is bounded below by F_* from (5.5), we get

$$\begin{aligned} \underbrace{\frac{1}{2R} \sum_{n=1}^{N} \frac{\alpha_n}{\Omega_n} \sum_{k=0}^{n-1} \beta_1^k \mathbb{E}\left[\|G_{n-k}\|_2^2\right]}_{A} &\leq F(x_0) - F_* \\ &+ \underbrace{\frac{\alpha_N^2 L}{2} \sum_{n=1}^{N} \mathbb{E}\left[\|u_n\|_2^2\right]}_{B} \\ &+ \underbrace{\frac{\alpha_N^3 L^2}{4R} \sqrt{1-\beta_1} \sum_{n=1}^{N} \sum_{k=0}^{n-1} \mathbb{E}\left[\|u_{n-k}\|_2^2\right] \sum_{l=k}^{n-1} \beta_1^l \sqrt{l}}_{C} \\ &+ \underbrace{\frac{3\alpha_N R}{\sqrt{1-\beta_1}} \sum_{n=1}^{N} \sum_{k=0}^{n-1} \left(\frac{\beta_1}{\beta_2}\right)^k \sqrt{k+1} \mathbb{E}\left[\|U_{n-k}\|_2^2\right]}_{D}. \end{aligned} \tag{C.43}$$

First looking at B, we have using Lemma C.2,

$$B \leq \frac{\alpha_N^2 L}{2(1-\beta_1)(1-\beta_1/\beta_2)} \sum_{i \in [d]} \left(\ln\left(1 + \frac{v_{N,i}}{\epsilon}\right) - N \log(\beta_2) \right). \tag{C.44}$$

Then looking at C and introducing the change of index $j = n - k$,

$$\begin{aligned}
C &= \frac{\alpha_N^3 L^2}{4R}\sqrt{1-\beta_1}\sum_{n=1}^{N}\sum_{j=1}^{n}\mathbb{E}\left[\|u_j\|_2^2\right]\sum_{l=n-j}^{n-1}\beta_1^l\sqrt{l} \\
&= \frac{\alpha_N^3 L^2}{4R}\sqrt{1-\beta_1}\sum_{j=1}^{N}\mathbb{E}\left[\|u_j\|_2^2\right]\sum_{n=j}^{N}\sum_{l=n-j}^{n-1}\beta_1^l\sqrt{l} \\
&= \frac{\alpha_N^3 L^2}{4R}\sqrt{1-\beta_1}\sum_{j=1}^{N}\mathbb{E}\left[\|u_j\|_2^2\right]\sum_{l=0}^{N-1}\beta_1^l\sqrt{l}\sum_{n=j}^{j+l}1 \\
&= \frac{\alpha_N^3 L^2}{4R}\sqrt{1-\beta_1}\sum_{j=1}^{N}\mathbb{E}\left[\|u_j\|_2^2\right]\sum_{l=0}^{N-1}\beta_1^l\sqrt{l}(l+1) \\
&\leq \frac{\alpha_N^3 L^2}{R}\sum_{j=1}^{N}\mathbb{E}\left[\|u_j\|_2^2\right]\frac{\beta_1}{(1-\beta_1)^2},
\end{aligned} \tag{C.45}$$

using Lemma C.4. Finally, using Lemma C.2, we get

$$C \leq \frac{\alpha_N^3 d L^2 \beta_1}{R(1-\beta_1)^3(1-\beta_1/\beta_2)}\sum_{i\in[d]}\left(\ln\left(1+\frac{v_{N,i}}{\epsilon}\right) - N\log(\beta_2)\right). \tag{C.46}$$

Finally, introducing the same change of index $j = n - k$ for D, we get

$$\begin{aligned}
D &= \frac{3\alpha_N R}{\sqrt{1-\beta_1}}\sum_{n=1}^{N}\sum_{j=1}^{n}\left(\frac{\beta_1}{\beta_2}\right)^{n-j}\sqrt{1+n-j}\,\mathbb{E}\left[\|U_j\|_2^2\right] \\
&= \frac{3\alpha_N R}{\sqrt{1-\beta_1}}\sum_{j=1}^{N}\mathbb{E}\left[\|U_j\|_2^2\right]\sum_{n=j}^{N}\left(\frac{\beta_1}{\beta_2}\right)^{n-j}\sqrt{1+n-j} \\
&\leq \frac{3\alpha_N R}{\sqrt{1-\beta_1}}\sum_{j=1}^{N}\mathbb{E}\left[\|U_j\|_2^2\right]\frac{1}{(1-\beta_1/\beta_2)^{3/2}},
\end{aligned} \tag{C.47}$$

using Lemma C.3. Finally, using Lemma 5.2 or equivalently Lemma C.2 with $\beta_1 = 0$, we get

$$D \leq \frac{6\alpha_N d R}{\sqrt{1-\beta_1}(1-\beta_1/\beta_2)^{3/2}}\sum_{i\in[d]}\left(\ln\left(1+\frac{v_{N,i}}{\epsilon}\right) - N\ln(\beta_2)\right). \tag{C.48}$$

This is as far as we can get without having to use the specific form of α_N given by either (C.2) for Adam or (C.3) for Adagrad. We will now split the proof for either algorithm.

Adam For Adam, using (C.2), we have $\alpha_n = (1-\beta_1)\Omega_n\alpha$. Thus, we can simplify the A term from (C.43), also using the usual change of index $j = n-k$, to get

$$\begin{aligned}
A &= \frac{1}{2R}\sum_{n=1}^{N}\frac{\alpha_n}{\Omega_n}\sum_{j=1}^{n}\beta_1^{n-j}\mathbb{E}\left[\|G_j\|_2^2\right] \\
&= \frac{\alpha(1-\beta_1)}{2R}\sum_{j=1}^{N}\mathbb{E}\left[\|G_j\|_2^2\right]\sum_{n=j}^{N}\beta_1^{n-j} \\
&= \frac{\alpha}{2R}\sum_{j=1}^{N}(1-\beta_1^{N-j+1})\mathbb{E}\left[\|G_j\|_2^2\right] \\
&= \frac{\alpha}{2R}\sum_{j=1}^{N}(1-\beta_1^{N-j+1})\mathbb{E}\left[\|\nabla F(x_{j-1})\|_2^2\right] \\
&= \frac{\alpha}{2R}\sum_{j=0}^{N-1}(1-\beta_1^{N-j})\mathbb{E}\left[\|\nabla F(x_j)\|_2^2\right]. \qquad \text{(C.49)}
\end{aligned}$$

If we now introduce τ as in (C.8), we can first notice that

$$\sum_{j=0}^{N-1}(1-\beta_1^{N-j}) = N - \beta_1\frac{1-\beta_1^N}{1-\beta_1} \geq N - \frac{\beta_1}{1-\beta_1}. \qquad \text{(C.50)}$$

Introducing

$$\tilde{N} = N - \frac{\beta_1}{1-\beta_1}, \qquad \text{(C.51)}$$

we then have

$$A \geq \frac{\alpha\tilde{N}}{2R}\mathbb{E}\left[\|\nabla F(x_\tau)\|_2^2\right]. \qquad \text{(C.52)}$$

Further notice that for any coordinate $i \in [d]$, we have $v_{N,i} \leq \frac{R^2}{1-\beta_2}$, besides $\alpha_N \leq \alpha\frac{1-\beta_1}{\sqrt{1-\beta_2}}$, so that putting together (C.43), (C.52), (C.44), (C.46) and (C.48) we get

$$\mathbb{E}\left[\|\nabla F(x_\tau)\|_2^2\right] \leq 2R\frac{F_0 - F_*}{\alpha\tilde{N}} + \frac{E}{\tilde{N}}\left(\ln\left(1 + \frac{R^2}{\epsilon(1-\beta_2)}\right) - N\log(\beta_2)\right), \qquad \text{(C.53)}$$

with

$$E = \frac{\alpha dRL(1-\beta_1)}{(1-\beta_1/\beta_2)(1-\beta_2)} + \frac{2\alpha^2 dL^2\beta_1}{(1-\beta_1/\beta_2)(1-\beta_2)^{3/2}} + \frac{12dR^2\sqrt{1-\beta_1}}{(1-\beta_1/\beta_2)^{3/2}\sqrt{1-\beta_2}}. \qquad \text{(C.54)}$$

This conclude the proof of theorem 9.

Adagrad For Adagrad, we have $\alpha_n = (1-\beta_1)\alpha$, $\beta_2 = 1$ and $\Omega_n \leq \sqrt{N}$ so that,

$$\begin{aligned}
A &= \frac{1}{2R}\sum_{n=1}^{N}\frac{\alpha_n}{\Omega_n}\sum_{j=1}^{n}\beta_1^{n-j}\mathbb{E}\left[\|G_j\|_2^2\right] \\
&\geq \frac{\alpha(1-\beta_1)}{2R\sqrt{N}}\sum_{j=1}^{N}\mathbb{E}\left[\|G_j\|_2^2\right]\sum_{n=j}^{N}\beta_1^{n-j} \\
&= \frac{\alpha}{2R\sqrt{N}}\sum_{j=1}^{N}(1-\beta_1^{N-j+1})\mathbb{E}\left[\|G_j\|_2^2\right] \\
&= \frac{\alpha}{2R\sqrt{N}}\sum_{j=1}^{N}(1-\beta_1^{N-j+1})\mathbb{E}\left[\|\nabla F(x_{j-1})\|_2^2\right] \qquad \text{(C.55)} \\
&= \frac{\alpha}{2R\sqrt{N}}\sum_{j=0}^{N-1}(1-\beta_1^{N-j})\mathbb{E}\left[\|\nabla F(x_j)\|_2^2\right]. \qquad \text{(C.56)}
\end{aligned}$$

Reusing (C.50) and (C.51) from the Adam proof, and introducing τ as in (5.12), we immediately have

$$A \geq \frac{\alpha\tilde{N}}{2R\sqrt{N}}\mathbb{E}\left[\|\nabla F(x_\tau)\|_2^2\right]. \qquad \text{(C.57)}$$

Further notice that for any coordinate $i \in [d]$, we have $v_N \leq NR^2$, besides $\alpha_N = (1-\beta_1)\alpha$, so that putting together (C.43), (C.57), (C.44), (C.46) and (C.48) with $\beta_2 = 1$, we get

$$\mathbb{E}\left[\|\nabla F(x_\tau)\|_2^2\right] \leq 2R\sqrt{N}\frac{F_0 - F_*}{\alpha\tilde{N}} + \frac{E}{\tilde{N}}\ln\left(1 + \frac{NR^2}{\epsilon}\right), \qquad \text{(C.58)}$$

with

$$E = \alpha dRL + \frac{2\alpha^2 dL^2\beta_1}{1-\beta_1} + \frac{12dR^2}{1-\beta_1}. \qquad \text{(C.59)}$$

This conclude the proof of theorem 10.

RÉSUMÉ

Les récents progrès en apprentissage profond permettent désormais l'analyse détaillée de données audio ainsi que leur génération. Les applications sont multiples : transcription automatique de morceaux de musique, séparation de source, synthèse vocale avec différentes identités du locuteur, synthèse de nouveaux instruments etc. Pour cette thèse, l'objectif est de trouver une architecture simple, rapide et précise capable de résoudre des tâches comme la modélisation d'instruments de musique, ou la séparation de source. L'entraînement de tels modèles implique l'utilisation de technique d'optimisation stochastique dont nous chercherons également à couvrir les aspects théoriques.

MOTS CLÉS

Apprentissage machine, optimisation, intelligence artificielle, synthèse audio, séparation de sources.

ABSTRACT

Thanks to recent progress in deep learning, we are now able to analyse and generate complex audio data. Many new applications are possible: automatic transcription of music tracks, source separation, speech synthesis with different speakers, synthesis of new instruments etc. In this thesis, we aim at developing a simple architecture, that is both fast and accurate, able to solve different audio tasks such as the modelisation of musical instruments or source separation. The training of such models requires the use of stochastique optimizatoin techniques, of which we will cover the relevent theoretical aspects.

KEYWORDS

Machine learning, optimization, artificial intelligence, audio synthesis, source separation.