Efficient Bound of Lipschitz Constant for Convolutional Layers by Gram Iteration

Blaise Delattre [1 2] **Quentin Barthélemy** [1] **Alexandre Araujo** [3] **Alexandre Allauzen** [2 4]

Abstract

Since the control of the Lipschitz constant has a great impact on the training stability, generalization, and robustness of neural networks, the estimation of this value is nowadays a real scientific challenge. In this paper we introduce a precise, fast, and differentiable upper bound for the spectral norm of convolutional layers using circulant matrix theory and a new alternative to the Power iteration. Called the Gram iteration, our approach exhibits a superlinear convergence. First, we show through a comprehensive set of experiments that our approach outperforms other state-of-the-art methods in terms of precision, computational cost, and scalability. Then, it proves highly effective for the Lipschitz regularization of convolutional neural networks, with competitive results against concurrent approaches.

1. Introduction

Since the landmark result of convolutional neural networks on computer vision tasks (Krizhevsky et al., 2012), researchers have tried to develop various methods and techniques to improve the training stability, generalization, and robustness of neural networks. Recent work has shown that the Lipschitz constant of the network, which is a measure of regularity (Virmaux & Scaman, 2018), can be tied to all of these properties (Bartlett et al., 2017; Yoshida & Miyato, 2017; Tsuzuku et al., 2018). Indeed, controlling this constant facilitates training by avoiding exploding gradients, improves generalization on unseen data, and makes networks more robust against adversarial attacks. Unfortunately, computing the Lipschitz constant of a neural network is an NP-hard problem (Virmaux & Scaman, 2018), and while implementation exists (Fazlyab et al., 2019; Latorre et al., 2020) their prohibitive computational cost inhibits their use for deep learning training.

To efficiently constrain or regularize the Lipschitz constant, researchers have focused on individual layers. In fact, due to the composition property, the product of the Lipschitz of each layer is a (loose) upper bound of the global Lipschitz of the network. By considering the ℓ_2-norm, the Lipschitz for each linear transformation reduces to the spectral norm and is equal to the largest singular value of its weight matrix. Efficient iterative methods, called *power method* or *power iteration*, for computing the spectral norm have existed for decades with many variants (Golub et al., 2000).

For convolution neural networks, several approaches have been proposed to efficiently estimate this largest singular value of convolutional layers. Ryu et al. (2019) and Farnia et al. (2019) rely on a power iteration approach where the convolution is treated as a linear operator that performs the matrix-vector product. To trade speed for precision, only one iteration is carried out. Other works have leveraged the structure of convolution (*i.e.* Toeplitz or circular structure) and the Fourier transform to devise exact but computationally expensive methods (Sedghi et al., 2019; Bibi et al., 2019) or efficient upper bounds for the largest singular value (Singla et al., 2021; Araujo et al., 2021; Yi, 2021).

In this paper, we build upon recent approaches to introduce an efficient procedure for estimating a bound on the Lipschitz constant of convolutional layers. Precise and fast, this bound scales nicely with convolution parameters (the kernel size, channel, and input spatial dimensions). The procedure is based on *Gram iteration*, a new iterative method to compute the largest singular value with super-linear convergence. Furthermore, our estimate is guaranteed to be a strict upper bound at each iteration. In contrast, methods based on power iteration have no such guarantees and can provide a harmful lower bound. After presenting the background and theoretical results of our contribution, we perform an extensive set of experiments to demonstrate the superiority of our method over all previous concurrent approaches. Finally, we demonstrate the usefulness of our method on Lipschitz regularization and show that we can achieve similar accuracy while being more stable.

[1]FOXSTREAM, Vaulx-en-Velin, France [2]Miles Team, LAMSADE, Université Paris-Dauphine, PSL University, Paris, France [3]New York University [4]ESPCI PSL, Paris, France. Correspondence to: Blaise Delattre <blaise.delattre@dauphine.eu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

Table 1. Qualitative summary of estimations of Lipschitz constant of convolutional layers. Precision and speed qualifications come from experimental results. "deter" abbreviates "deterministic". Ryu et al. (2019) uses PI adapted to convolution to compute an estimate arbitrarily precise in a zero padding (a special case of constant padding) setting but suffers from slow convergence. Araujo et al. (2021) uses the Toeplitz matrix theory to devise a fast bound in spite of precision. Other approaches suppose circular padding for convolution which is a reasonable (comparable performance) and useful assumption to reduce computation complexity thanks to the Fourier transform. Sedghi et al. (2019) provides exact computation of singular values, however, the method is very slow and does not scale. Singla et al. (2021) proposes a faster using PI method to the detriment of precision. Methods using PI are not guaranteed to produce an upper bound on convolution spectral norm regarding constant or circular padding. Our method gathers the best of both worlds: being fast, precise, deterministic, well differentiable, and with a strict upper bound on the spectral norm.

Methods	Precision	Speed	Padding	Upper Bound	Algorithm	Convergence
Ryu et al.; Farnia et al.	++	-	zero	✗	PI: iterative, non-deter	linear
Sedghi et al.; Bibi et al.	++	- -	circular	exact	SVD: deter	-
Singla et al.; Yi	+	+	circular	✗	PI: iterative, non-deter	linear
Araujo et al.	-	++	zero	✓	close-form, deter	-
Ours	++	++	circular	✓	GI: iterative, deter	superlinear

2. Related Work

The Lipschitz constant for the ℓ_2-norm of a neural network f is defined as follow for an input space $\mathcal{X}$:

$$\mathrm{Lip}(f) = \sup_{\substack{x,x' \in \mathcal{X} \\ x \neq x'}} \frac{\|f(x) - f(x')\|_2}{\|x - x'\|_2} \ . \tag{1}$$

It measures network sensitivity toward an input perturbation ϵ: $\|f(x) - f(x+\epsilon)\|_2 \leq \mathrm{Lip}(f)\ \|\epsilon\|_2$, using Equation (1). If ℓ_2-norm is a well-spread choice, other approaches use ℓ_∞-norm (Anil et al., 2019; Zhang et al., 2022). In this work, we are interested in estimating and controlling the Lipschitz constant of neural networks which are compositions of layers *i.e.* $f = \phi_l \circ \phi_{l-1} \circ \cdots \circ \phi_1$. Computing the Lipschitz constant of f is a hard problem (Virmaux & Scaman, 2018) and current implementations to estimate it directly fail to scale for deep architecture. Some approaches compute the Lipschitz constant using bound propagation techniques in network verification (Zhang et al., 2019; Jordan & Dimakis, 2020; Shi et al., 2022). However, a simple bound on the network Lipschitz constant can be derived by considering the individual layers:

$$\mathrm{Lip}(f) \leq \prod_{i=1}^{l} \mathrm{Lip}(\phi_i) \ . \tag{2}$$

A technique to enforce control on the Lipschitz constant is orthogonalization of layer ϕ_i by design (Trockman et al., 2021), by regularization (Wang et al., 2020; Huang et al., 2020) or by optimizing on manifold (Anil et al., 2019). Another approach is to perform spectral normalization (Yoshida & Miyato, 2017; Miyato et al., 2018; Farnia et al., 2019) or regularization (Gouk et al., 2021).

For dense layers, the problem boils down to matrix spectral norm computation or equivalently computing the largest singular value. Power iteration (PI) is a classical solution that serves as the basis for many others, and it is described in Algorithm 1, where $*$ corresponds to the conjugate transpose operator. The convergence is linear with a rate of convergence of σ_1/σ_2. This ratio can be close to 1 and other variations of PI improve it (Lanczos, 1950; Arnoldi, 1951; Kublanovskaya, 1962) and more recently (Lehoucq et al., 1996). However, these methods suffer from important drawbacks. First, many iterations are required to ensure full convergence, which can be computationally expensive. Second, the method is not fully deterministic, as it starts from a randomly generated vector. Finally, intermediate iterations are not guaranteed to be a strict upper bound on the spectral norm.

Algorithm 1 : Power_iteration(G, N_{iter})

Inputs matrix: G, number of iterations: N_{iter}
Initialization: draw a random vector u
for $1 \ldots N_{\text{iter}}$
  $v \leftarrow Gu/\|Gu\|_2$
  $u \leftarrow G^* v/\|G^* v\|_2$
$\sigma_1 \leftarrow (Gu)^* v$
return σ_1

(Miyato et al., 2018) reshapes convolutional filter $c_{\text{out}} \times c_{\text{in}} \times k \times k$ to a matrix of size $c_{\text{out}} \times c_{\text{in}} \times k^2$ and computes its spectral norm as a loose proxy for convolution one. Works of (Ryu et al., 2019; Farnia et al., 2019) generalized PI to convolutional layers leveraging transpose convolution operator to produce an exact estimate. However, time cost increases strongly with input spatial size and kernel size, and differentiability is not complete as the gradient is not accumulated in vector u and v, as explained in (Singla et al., 2021).

Another line of work exploits the structure of convolution to calculate its spectral norm with reduced computational complexity (Sedghi et al., 2019; Bibi et al., 2019; Singla et al., 2021; Araujo et al., 2021). In Sedghi et al. (2019), circular padding convolutions are represented using properties of doubly-block circulant matrix. Then the Fourier transform can be used to express singular values of convolutions. This method leads to the calculation of n^2 spectral norms for $c_{\text{out}} \times c_{\text{in}}$ matrix using SVD. The result is exact but the method is very slow and Singla et al. (2021) proposes a method that further reduces computation but at the expense of a decreased precision on the Lipschitz constant.

Table 1 is a qualitative summary of different methods to estimate the Lipschitz constant of convolutional layers. Ryu et al. (2019) proposes a convolution-adapted PI method in which computational time scales with convolution product, it can be costly to derive a very precise spectral norm estimation as convergence is linear. Concerning Araujo et al. (2021) method, it is fast but only precise for $c_{\text{out}} \times 1 \times k \times k$ or $1 \times c_{\text{in}} \times k \times k$ convolutional filters. Sedghi et al. (2019) provides an exact estimate of the Lipschitz constant under circular convolution hypothesis by performing n^2 SVD on $c_{\text{out}} \times c_{\text{in}}$ matrices which is very slow and not scalable for large convolution filters. Using the same hypothesis, Singla et al. (2021) proposes a bound by taking a minimum of four $c_{\text{out}}k \times c_{\text{in}}k$ matrix spectral norms using PI to compute them, using PI makes the method faster but at the expense of degrading bound precision. Yi (2021) proves that this type of bounds also holds in zero padding case. It is important to note that all methods using PI (Ryu et al., 2019; Singla et al., 2021) produce do not offer upper bound guarantees. For experiences, we choose the Ryu et al. (2019) method with 100 iterations (to ensure convergence) as a reference value. Indeed, most convolutional layers of CNNs use constant padding. (Sedghi et al., 2019) gives an exact spectral norm for circular padding convolution. In recent works, researchers have mainly focused on speed but at the expense of precision (Singla et al., 2021; Araujo et al., 2021). Our method gathers both methods' strengths: differentiable, scalable, very fast with superlinear convergence, and with a precision matching Sedghi et al. (2019) method.

3. Background on Convolutional Layers

In this section, we present some properties of convolutional layers with circular padding based on the inherent matrix structure of the convolution operator.

Let $X \in \mathbb{R}^{n\times n}$ be the input, and $K \in \mathbb{R}^{k\times k}$ a convolution kernel. In the following, the kernel will be always considered as padded towards input size, *i.e.* $K \in \mathbb{R}^{n\times n}$. Convolution product $\star$ can be expressed as a matrix-vector product, where the structure of matrix $W \in \mathbb{R}^{n^2\times n^2}$ takes into account the weight sharing of convolution (Jain, 1989):

$$\text{vec}(K \star X) = W\text{vec}(X)\ . \tag{3}$$

Using a *circular padding*, W is a doubly-block circulant matrix in which each row block is a circulant shift of the previous one and where each block is a circulant matrix (Jain, 1989). Denoting $\text{circ}(\cdot)$ the operator generating a circulant matrix form a vector, W is fully determined by the kernel K as:

$$W = \begin{pmatrix} \text{circ}(K_1) & \text{circ}(K_2) & \cdots & \text{circ}(K_n) \\ \text{circ}(K_n) & \text{circ}(K_1) & \ddots & \vdots \\ \vdots & \ddots & \text{circ}(K_1) & \text{circ}(K_2) \\ \text{circ}(K_2) & \cdots & \text{circ}(K_n) & \text{circ}(K_1) \end{pmatrix}.$$

In the following, we denote $W = \text{blkcirc}(K_1, \ldots, K_n)$. However the choice of padding for the convolution has a crucial effect; indeed, with zero padding, the structure of the convolution matrix is not doubly-block circulant but *doubly-block Toeplitz*, and the following results with the Fourier diagonalization do not hold.

Now, based on the doubly-block circulant structure of convolution with circular padding, we present some useful properties. First, we formally introduce the discrete Fourier matrix and some known results.

Definition 1 (Fourier Transform)**.** *The discrete Fourier transform matrix $U \in \mathbb{C}^{n\times n}$ is defined such that, for $1 \leq u, v \leq n$:*

$$U_{u,v} = e^{-\frac{2\pi \mathbf{i} uv}{n}}$$

and its inverse is defined as $U^{-1} = \frac{1}{n}\, U^*$.

Operating on a vectorized 2D input X, 2D discrete Fourier transform is expressed as $\text{vec}(U^\top XU) = (U \otimes U)\text{vec}(X)$, where $\otimes$ is the Kronecker product. We denote $F = U \otimes U$ in the following of the paper.

Theorem 1 (Section 5.5 of Jain (1989))**.** *Let $K \in \mathbb{R}^{n\times n}$ a convolutional kernel and $W \in \mathbb{R}^{n^2\times n^2}$ be the doubly-block circulant matrix such that $W = \text{blkcirc}\,(K_1, \ldots, K_n)$, then, W can be diagonalized as follows:*

$$W = F\text{diag}(\lambda)F^{-1} \tag{4}$$

where $\lambda = F\text{vec}(K)$ are the eigenvalues of W.

In the context of deep learning, convolutions are applied with multiple channels: c_{out} convolutions are applied on input with c_{in} channels. Therefore, the matrix associated with the convolutional filter $K \in \mathbb{R}^{c_{\text{out}}\times c_{\text{in}}\times n\times n}$ becomes a block matrix, where each block is a doubly-block circulant matrix: $W = (W_{i,j})$, for $1 \leq i \leq c_{\text{out}}$ and $1 \leq j \leq c_{\text{in}}$, with $W_{i,j} = \text{blkcirc}(K_{i,j,1}, \ldots, K_{i,j,n})$. As also studied

by Sedghi et al. (2019) and Yi (2021), this type of block doubly-block circulant matrix can be *block* diagonalized as follows.

Theorem 2 (Corollary A.1.1. of Trockman et al. (2021))**.** *Let $P_{out} \in \mathbb{R}^{c_{out}n^2 \times c_{out}n^2}$ and $P_{in} \in \mathbb{R}^{c_{in}n^2 \times c_{in}n^2}$ be permutation matrices and let $W \in \mathbb{R}^{c_{out}n^2 \times c_{in}n^2}$ be the matrix equivalent of the multi-channel circular convolution defined by filter $K \in \mathbb{R}^{c_{out} \times c_{in} \times n \times n}$, then W can be block diagonalized as follows:*

$$W = (I_{c_{out}} \otimes F) P_{out} D P_{in}^\top (I_{c_{in}} \otimes F^{-1}) \tag{5}$$

where D is a block diagonal matrix with n^2 blocks of size $c_{out} \times c_{in}$ and where

$$D = P_{out}^\top \begin{pmatrix} F\mathrm{vec}(K_{1,1}) & \cdots & F\mathrm{vec}(K_{1,c_{in}}) \\ \vdots & & \vdots \\ F\mathrm{vec}(K_{c_{out},1}) & \cdots & F\mathrm{vec}(K_{c_{out},c_{in}}) \end{pmatrix} P_{in} \ .$$

Matrices P_{out} and P_{in} reshaping matrix D into a block diagonal matrix correspond to an alternative vectorization of layer input (Sedghi et al., 2019; Yi, 2021) while keeping singular values identical (Henderson & Searle, 1981).

Based on this result and on the properties of block diagonal matrices, it is easy to compute the largest singular value σ_1 of W from the block diagonal matrix D. Let $(D_i)_{i=1,\dots,n^2}$ be the diagonal blocks of the matrix D, then:

$$\sigma_1(W) = \sigma_1(D) = \max_i \sigma_1(D_i) \ . \tag{6}$$

It requires calculating the spectral norm of each D_i, one could use SVD as in (Sedghi et al., 2019) or PI as in (Yi, 2021), however, these algorithms do not scale up nor are efficient. In the following section, we present a more precise, fast, and scalable approach.

4. Lipschitz Constant of Convolutional Layers

4.1. Gram iteration

In this part, we derive a new method to estimate an upper bound of the spectral norm, deferring all the proofs to Appendix B. We consider a general matrix $G \in \mathbb{C}^{p \times q}$ and its singular values $\sigma(G)$.

Definition 2 (Schatten ρ-norms (Bhatia, 2013))**.** *For $\rho \in [1, +\infty)$, Schatten ρ-norms of matrix G are defined as:*

$$s_\rho(G) = \|\sigma(G)\|_\rho = \left(\sum_{i=1}^{\min\{p,q\}} \sigma_i^\rho(G) \right)^{1/\rho} ,$$

where $\rho = 2$ gives Frobenius norm $\|G\|_F$ and $\rho = +\infty$ gives spectral norm $\sigma_1(G)$.

We define the recurrent sequence $(G^{(t)})$ as, $\forall t \geq 1$:

$$\begin{cases} G^{(1)} & = G \\ G^{(t+1)} & = G^{(t)^*} G^{(t)} \ , \end{cases} \tag{7}$$

where $G^{(t+1)}$ is the Gram matrix of $G^{(t)}$ (Bhatia, 2013).

We observe that the 2^t-th root of the squared Frobenius norm of $G^{(t)}$ is the Schatten 2^t-norm of G:

$$\sqrt[2^t]{\|G^{(t)}\|_F^2} = \|G^{(t)}\|_F^{2^{1-t}} = \|\sigma(G)\|_{2^t} \ , \tag{8}$$

which converges superlinearly (almost quadratically) to $\|\sigma(G)\|_\infty$, which is equal to the spectral norm of G, *i.e.* $\sigma_1(G)$. Those results are formalized in the following theorem.

Theorem 3 (Main result). *Let $G \in \mathbb{C}^{p \times q}$ and define the recurrent sequence $G^{(t+1)} = G^{(t)^*} G^{(t)}$, with $G^{(1)} = G$. Let $(\|G^{(t)}\|_F^{2^{1-t}})$ be another sequence based on $G^{(t)}$, then, $\forall t \geq 1$, we have the following results:*

- *The sequence is an upper bound on spectral norm:*

$$\sigma_1(G) \leq \|G^{(t)}\|_F^{2^{1-t}} \ ;$$

- *The sequence converges to the spectral norm:*

$$\|G^{(t)}\|_F^{2^{1-t}} \underset{t \to +\infty}{\longrightarrow} \sigma_1(G) \ ,$$

and this convergence is Q-superlinear of order $2 - \epsilon$, for ϵ arbitrary small.

We denote the iterative method described in Theorem 3 as *Gram iteration* (GI) due to the recurrence relation: $G^{(t+1)}$ is the Gram matrix of $G^{(t)}$. A pseudo-code is given in Algorithm 2. Note that to avoid overflow, matrix G is rescaled at each iteration and scaling factors are cumulated in variable r, in order to unscale the result at the end of the method, see Appendix A.3. Unscaling is crucial as it is required to remain a strict upper bound on the spectral norm at each iteration of the method. One can note that GI gives a proper norm at each iteration, which is a deterministic upper bound on the spectral norm, and which converges quasi-quadratically. However, GI is not a matrix-free method, contrary to PI and its variants.

Given our interest in optimizing within this bound, differentiability is a crucial property. In the following, we study the differentiability of the method and provide an explicit gradient formulation.

Proposition 1. *Bound sequence $(\|G^{(t)}\|_F^{2^{1-t}})$ is differentiable regarding G:*

$$\frac{\partial(\|G^{(t)}\|_F^{2^{1-t}})}{\partial G} = \frac{G(G^*G)^{2^{t-1}-1}}{(\|G^{(t)}\|_F)^{2(1-2^{-t})}} \ . \tag{9}$$

Algorithm 2 : Lip_dense(G, N_{iter})

Inputs matrix: G, number of iterations: N_{iter}
$r \leftarrow 0$ // initialize rescaling
for $1 \dots N_{\text{iter}}$
  $G \leftarrow G/\|G\|_F$ // rescale to avoid overflow
  $G \leftarrow G^* G$ // Gram iteration
  $r \leftarrow 2(r + \log\|G\|_F)$ // cumulate rescaling
$\sigma_1 \leftarrow \|G\|_F^{2^{-N_{\text{iter}}}} \exp(2^{-N_{\text{iter}}} r)$
return σ_1

This proposition is key as it enables the implementation of the bound with explicit gradient for optimization, and thus provides an efficient way to backpropagate during training. Note that the gradient in GI is complete, whereas the gradient in PI vectors is not accumulated during iterations.

4.2. Spectral norm of convolutional layers

GI described in Algorithm 2 can be used directly to estimate the spectral norm of dense layers. For convolutional layers, we use Equation (6) and apply Theorem 3 on sequences $(D_i^{(t)})$.

Proposition 2. *An upper bound of the spectral norm of a circular convolutional layer is estimated as:*

$$\sigma_1(W) \leq \max_{1 \leq i \leq n^2} \|D_i^{(t)}\|_F^{2^{1-t}} \underset{t \to +\infty}{\longrightarrow} \sigma_1(W) \,. \tag{10}$$

Because $\|D_i^{(t)}\|_F^{2^{1-t}} = s_{2^t}(D_i)$ is a Schatten norm, it is convex and a maximum over norms establishes a convex function. Hence the upper bound is subdifferentiable everywhere.

Algorithm 3 : Lip_conv(K, N_{iter})

Inputs Filter: K, number of iterations: N_{iter}
$D \leftarrow \texttt{FFT2}(K)$ // FFT
$r \leftarrow 0_{n^2}$ // initialize rescaling
for $1 \dots N_{\text{iter}}$
  for i **in** $1 \dots n^2$ // for-loop in parallel
    $D_i \leftarrow D_i/\|D_i\|_F$ // rescale to avoid overflow
    $D_i \leftarrow D_i^* D_i$ // Gram iteration
    $r_i \leftarrow 2(r_i + \log\|D_i\|_F)$ // cumulate rescaling
$\sigma_1 \leftarrow \max_i\{\|D_i\|_F^{2^{-N_{\text{iter}}}} \exp(2^{-N_{\text{iter}}} r_i)\}$
return σ_1

Using Proposition 2 we devise Algorithm 3 to compute spectral norm for a convolutional layer defined by a convolutional filter K padded towards input size. This algorithm is deterministic and precise due to the properties of Gram iteration, as well as tractable and scalable for convolutions as we consider a batch of n^2 relatively small matrices of size $c_{\text{out}} \times c_{\text{in}}$. The batch computation on GPU is well optimized hence the speed of the algorithm.

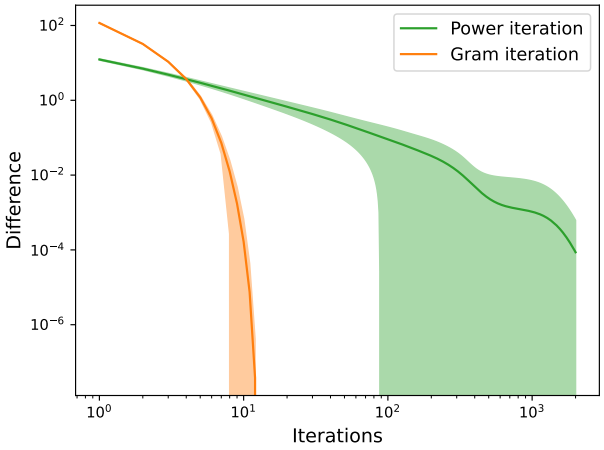


Figure 1. Convergence plot in log-log scale for spectral norm computation, comparing Power iteration and Gram iteration, one standard deviation shell is represented in a light color. Difference at each iteration is defined as $|\sigma_{1\text{method}} - \sigma_{1\text{ref}}|$. Gram iteration converges to numerical 0 in less than 12 iterations while Power iteration has not yet converged with 2,000 iterations and has a larger dispersion through runs.

It is possible to consider smaller $n_0 \leq n$ in Equation (10) to further reduce computation for large input spatial size and still compute a bound as explained in Appendix C.1. Approximation between circular and zero paddings is discussed in Appendix C.2.

5. Numerical experiments

For research reproducibility, the code is available https://github.com/blaisedelattre/lip4conv. All experiences were done on one NVIDIA RTX A6000 GPU.

5.1. Computation of spectral norms

We compare PI and GI methods for spectral norms computation, and a set of 100 random Gaussian matrices 2000×1000 is generated. The reference value for the spectral norm of a matrix $\sigma_{1\text{ref}}$ is obtained from PyTorch using SVD, and Algorithm 2 is used for GI. Estimation error is defined as $|\sigma_{1\text{method}} - \sigma_{1\text{ref}}|$. We observe in Figure 1 that PI needs more than 2,000 iterations to fully converge to numerical 0 and at minimum 90 iterations. Runs have a much larger standard deviation in comparison to GI. For that experiment, full convergence required up to 5,000 for some realizations, see Figure 6 in Appendix. This highlights how PI convergence can be very slow from one run to another for different generated matrices. Furthermore, in Figure 7 in Appendix we show convergence of methods when a generated matrix is fixed: variance PI observed for PI is only due to its non-deterministic behavior, unlike GI which is deterministic. This shows how the variance of PI is sensitive due to

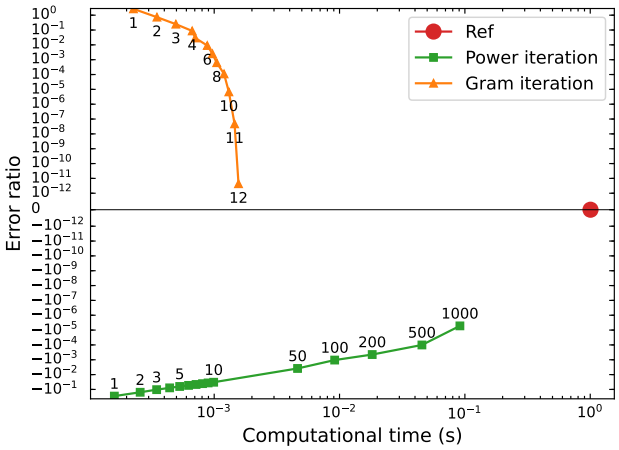


Figure 2. Error ratios over computational times of methods for spectral norm computation. Error ratio is defined as $\sigma_{1\text{method}}/\sigma_{1\text{ref}} - 1$. Points are annotated with the number of iterations. GI converges in $\pm 10^{-3}$s to numerical 0, while PI convergence is slower.

two causes: random generation of input matrices and the method in itself with random vector initialization at the start of the algorithm. On the contrary, GI converges under 15 iterations in every case.

To capture the sign of the error, *i.e.* if the current estimator is an upper bound or not, we define the error ratio as $\sigma_{1\text{method}}/\sigma_{1\text{ref}} - 1$, depicted in Figure 2 also reported in Table 5. We observe that the error ratio for PI is negative at each iteration, meaning that PI estimation approaches the reference while being inferior to it, thus being a lower bound. GI empirically enjoys a very fast convergence, with 10 iterations: doing one more iteration divide the error ratio by approximately $1e2$ and two more iterations by $1e4$. In contrast, PI has a much slower convergence: passing from 10 to 100 iterations only divide the ratio by a factor of 10. This faster convergence rate from GI makes the method much faster in terms of computational time than PI and SVD for the same precision. Those results motivate the use of GI for dense layers as well.

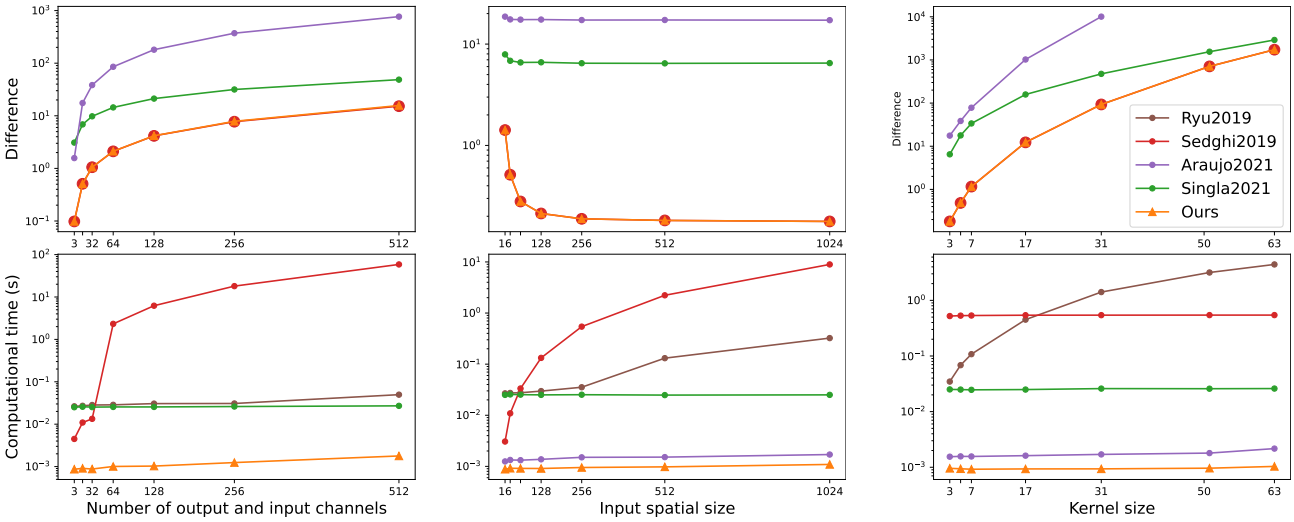
5.2. Estimation on simulated convolutional layers

This experiment assesses the properties of bounds from different methods in terms of precision and computational time. We consider convolutional layers with constant padding and make vary three parameters: number of input and output channels (Figure 3a), spatial input size n (Figure 3b) and kernel size k (Figure 3c). For each experiment, random kernels are generated from Gaussian, and uniform distribution is repeated 50 times. We take average of errors, defined $\sigma_{1\text{method}} - \sigma_{1\text{Ryu2019}}$, and computational time (in seconds). Method of Ryu et al. (2019) is taken as a reference with 100 iterations to have precise estimation, (Singla et al., 2021) with 50 iterations as in their code base, (Araujo et al., 2021) with 50 samples (10 taken in paper experiments description), and ours is Algorithm 3 with 5 iterations. We observe that our method gives a Lipschitz bound which matches the Lipschitz constant under the circular convolution hypothesis given by (Sedghi et al., 2019), which gives the exact spectral norm for circular padding convolution. Furthermore, our method is the fastest of the considered methods. We see that (Ryu et al., 2019) computational time scales very fast with n and k whereas other methods having circular hypothesis remain quasi constant regarding computational times. We note that the Lipschitz difference increases as kernel size varies and the difference between circular and zero padding is more pronounced. These experiments highlight the scalability of our method with respect to the parameters of convolutional layers.

5.3. Estimation on convolutional layers of CNNs

Inspired by (Singla et al., 2021), this experiment estimates the Lipschitz constant for each convolutional layer of a ResNet18 (He et al., 2016), pre-trained on the ImageNet-1k dataset. We take the same method's parameters as in Section 5.2. Figure 4 reports the difference between reference and Lipschitz bounds for all convolutional layers of ResNet18 for different methods. We observe that our method gives a bound very close to (Sedghi et al., 2019) and always above the reference value given by (Ryu et al., 2019). For the filter of index 3, we see that the value given by (Singla et al., 2021), 2.03, is below the reference value 2.05: this illustrated the issue with using PI in the pipeline to obtain an upper bound. Numerical results are detailed in Table 7.

Contrary to the first part studying only convolutional layers, the overall Lipschitz constant of the network is assessed here. We consider several CNNs pre-trained on the ImageNet-1k dataset: VGG (Simonyan & Zisserman, 2014) and ResNet (He et al., 2016) of different sizes. Using rules from Appendix E to compute the Lipschitz constant of each individual layer in CNN, we compute a bound B on the overall Lipschitz constant of the network. Then we compare this produced bound for each method to the one obtained by (Ryu et al., 2019) by considering the ratio: $B_{\text{method}}/B_{\text{Ryu2019}}$. We take 100 iterations for (Ryu et al., 2019) to have a precise reference, 50 iterations for (Singla et al., 2021), 20 samples for (Araujo et al., 2021) and 7 iterations for our method, as the task requires increased precision. Results are reported in Table 2, we see that our method gives results similar to (Sedghi et al., 2019) and overall network Lipschitz bound is tighter than (Singla et al., 2021) and (Araujo et al., 2021) in comparison to (Ryu et al., 2019). This experience illustrates that small errors in the estimation of a single Lipschitz constant layer can lead to major errors on the Lipschitz



(a) Varying number of channels c_{in} and c_{out}, with $n = 32$ and $k = 3$.

(b) Varying input size n, with $c_{\text{in}} = 16$, $c_{\text{in}} = 16$ and $k = 3$.

(c) Varying kernel size k, with $c_{\text{in}} = 16$, $c_{\text{in}} = 16$ and $n = 252$.

Figure 3. For each subplot, we vary a convolution layer parameter and display the difference and computational times of Lipschitz bounds. Difference is expressed as $\sigma_{1\,\text{method}} - \sigma_{1\,\text{Ryu2019}}$. Sedghi et al. (2019) time cost overall is prohibitive for its use, regarding Ryu et al. (2019) it scales strongly with input and kernel size. Araujo et al. (2021) and Singla et al. (2021) are intermediate trade offs between precision and speed. Our method performs best in terms of speed and precision regarding reference Ryu et al. (2019) and gives the same values as Sedghi et al. (2019) ones.

bound of the overall network and that acute precision is crucial. Moreover, methods using PI such as (Singla et al., 2021) have a huge standard deviation in this task which is problematic for deep networks whereas ours and (Sedghi et al., 2019) have standard deviations that remain small. Our method offers the same precision quality as (Sedghi et al., 2019) but significantly faster.

5.4. Regularization of convolutional layers of ResNet

Regularization of spectral norms of convolutional layers has been studied in previous works (Yoshida & Miyato, 2017; Miyato et al., 2018; Gouk et al., 2021). The goal of this experiment is to assess how different bounds control the Lipschitz constant L_j of each convolution layer ϕ_j along the training process. To measure this control, a target Lipschitz constant L is set for all convolutions. The loss optimized during training becomes: $\mathcal{L}_{\text{train}} + \mu_{\text{reg}} \sum_{j=1}^{l} \mathcal{L}_{\text{reg}}(\phi_j)$, with

$$\mathcal{L}_{\text{reg}}(\phi_j) = \sigma_{1\,\text{method}}(\phi_j)\ \mathbb{1}_{\sigma_{1\,\text{method}}(\phi_j) > L}\ . \tag{11}$$

This regularization loss penalizes the Lipschitz constant of the convolutional layer only if it is above the target Lipschitz constant L, else regularization is deactivated. Function $x \mapsto \mathbb{1}_{x>L}$ is not differentiable in L, but the probability for $\sigma_1(\phi_j)$ to be numerically equal to L is close to 0.

We use ResNet18 architecture (He et al., 2016), trained on the CIFAR-10 dataset for 200 epochs, and with a batch size of 256. We use SGD with a momentum of 0.9 and an initial learning rate of 0.1 with a cosine annealing schedule. The baseline is trained without regularizations. Different regularizations are compared: Lipschitz regularization by different methods with $\mu_{\text{reg}} = 1e\text{-}1$; and the usual weight decay (WD) applied on convolutional filters with $\mu_{\text{reg}} = 5e\text{-}3$, which has been picked to bound layers spectral norm around the target as good as possible. We only compare methods under the circular padding hypothesis and use (Sedghi et al., 2019) as the reference since it provides an exact value. We take 6 iterations for Gram iteration in our method, 10 for power iteration in (Singla et al., 2021), and 10 samples in (Araujo et al., 2021) to have similar training times. For our method, we implement an explicit backward differentiation using Equation (9) to speed up the gradient computation and reduce the memory footprint.

Figure 5 presents results on Lipschitz constant control of convolutional layers for ResNet18. The goal here is to have a Dirac distribution at target value $L = 1$, *i.e.* we want each convolutional layer to have a spectral norm of 1. We observe that the more precise a method is, the more well-controlled the resulting spectrum resulting from training is. Our regularization method matches each layer's target spectral norm of 1. Weight decay gives gross estimations and (Araujo et al., 2021) overestimates spectral norms and thus results in over-constrained Lipschitz constant for some layers. (Singla et al., 2021) produces a maximum spectral of 1.1 norm which is above target L but overall Lipschitz constant is centered around the target with dispersion. This suggests

Table 2. Comparison of networks bound Lipschitz ratio with standard deviation for several CNNs, reference for computing the ratio is the bound given by Ryu et al. (2019) method. Overall Lipschitz constant bound B_{method} is estimated for each method. Ratio of network Lipschitz bound is defined as $B_{\text{method}}/B_{\text{Ryu2019}}$. Results are averaged over 100 runs. Ratio standard deviations of ours, Araujo et al. (2021) and Sedghi et al. (2019) are induced by Ryu et al. (2019) method. We give the same ratio as Sedghi et al. (2019) in a much lower time.

Model	Ratio of Network Lipschitz Bound (Total Running Time (s))				
	Ours	Singla et al.	Araujo et al.	Sedghi et al.	Ryu et al.
VGG16	1.14 ± 0.020 (0.100)	23.90 ± 6.80 (0.47)	4.88e+09 $\pm$ 4.83e+09 (0.027)	1.14 ± 0.020 (525)	(0.64)
VGG19	1.16 ± 0.005 (0.110)	30.63 ± 0.30 (0.53)	9.70e+09 $\pm$ 4.84e+07 (0.030)	1.16 ± 0.005 (639)	(0.71)
ResNet18	1.47 ± 0.007 (0.039)	87.93 ± 0.88 (0.50)	3.03e+10 $\pm$ 1.56e+08 (0.039)	1.47 ± 0.007 (185)	(0.71)
ResNet34	1.82 ± 0.350 (0.060)	4982 ± 4894 (0.88)	7.86e+21 $\pm$ 7.86e+21 (0.050)	2.15 ± 0.350 (237)	(1.16)
ResNet50	1.68 ± 0.350 (0.100)	3338 ± 4622 (1.05)	6.66e+31 $\pm$ 9.42e+31 (0.050)	1.67 ± 0.350 (377)	(1.49)
ResNet101	1.74 ± 0.320 (0.173)	4026 ± 4178 (1.50)	6.55e+64 $\pm$ 1.13e+65 (0.100)	1.74 ± 0.320 (551)	(2.20)
ResNet152	1.92 ± 0.460 (0.260)	8.39e+4 $\pm$ 1.60e+5 (2.05)	2.85e+96 $\pm$ 5.71e+96 (0.120)	1.92 ± 0.460 (725)	(3.01)

that loosely bound under-constraints convolution's spectral norm and that the differentiation of our bound behaves correctly. Detailed spectral norm histograms over epochs for each Lipschitz regularization method are presented in Figure 10, showing that all spectral norms are below the target Lipschitz constant at epoch 120 for our method and accounts for fast convergence in Lipschitz regularization in comparison to other methods.

5.5. Image classification with regularized ResNet

Finally, to study the impact of regularization on image classification accuracy and training time, we conduct experiments using the same experimental training framework as previous Section 5.4 for CIFAR-10. That is to say with a high learning rate and regularization parameter to discriminate the different bounds. Trainings are repeated 10 times. Accuracy of (Sedghi et al., 2019) was omitted because the training was too long to finish, around 6750s per epoch. We vary the number of iterations for our bound, from 3 to 6 at the end of the training, because we observe reduced standard deviation by doing so. Our training time is reported for 6 Gram iterations. Results of experiments are reported in Table 3. Most of the bounds for Lipschitz regularization (Ryu et al., 2019; Araujo et al., 2021; Singla et al., 2021) tend to slightly degrade accuracy in comparison to baseline, contrary to our bound where a small accuracy gain can be observed: our Lipschitz regularization is not a trade-off on accuracy under this training setting. Furthermore, the low standard deviation of our accuracies suggests that our regularization stabilizes training contrary to all other approaches. The computational cost of our method is the lightest of all Lipschitz regularization approaches.

To demonstrate the scalability of our method, a ResNet18 is trained on the ImageNet-1k dataset, processing 256×256 images. ResNet18 baseline (with WD) is compared with two regularized ResNet18: one trained from scratch on 88 epochs, and one initialized on a pre-trained baseline and fined-tuned on 10 epochs. Trainings are repeated four times on 4 GPU V100. The results of experiments are reported in Table 4. This experiment shows that our bound remains efficient when dealing with large images. Moreover, it is not required to train a new ResNet from scratch: any trained ResNet can be fine-tuned on a few epochs to be regularized.

Table 3. This table shows test accuracies and training times for ResNet18 on CIFAR-10, repeated 10 times. Our method has a much narrow standard deviation, slightly better accuracy, and lower training time in comparison to other bounds for Lipschitz regularization.

	Test accuracy (%)	**Time per epoch (s)**
Baseline	93.32 ± 0.12	11.4
Baseline WD	93.30 ± 0.19	11.4
Ryu et al.	93.27 ± 0.13	95.0
Sedghi et al.	✗	6750
Araujo et al.	90.66 ± 0.26	37.4
Singla et al.	93.29 ± 0.15	38.4
Ours	93.48 ± 0.08	31.9

Table 4. This table shows test accuracies and training times for ResNet18 on ImageNet-1k, repeated 4 times.

	Test accuracy (%)	**Time per epoch (s)**
Baseline WD	69.76	746
Ours	70.77 ± 0.12	782
Ours (fine-tuned)	70.50 ± 0.05	782

6. Conclusion

In this paper, we introduce an efficient upper bound on the spectral norm of circular convolutional layers. It uses Gram iteration, a new and promising alternative to the power method, exhibiting quasi-quadratic convergence. Our comprehensive set of experiments shows that our method is

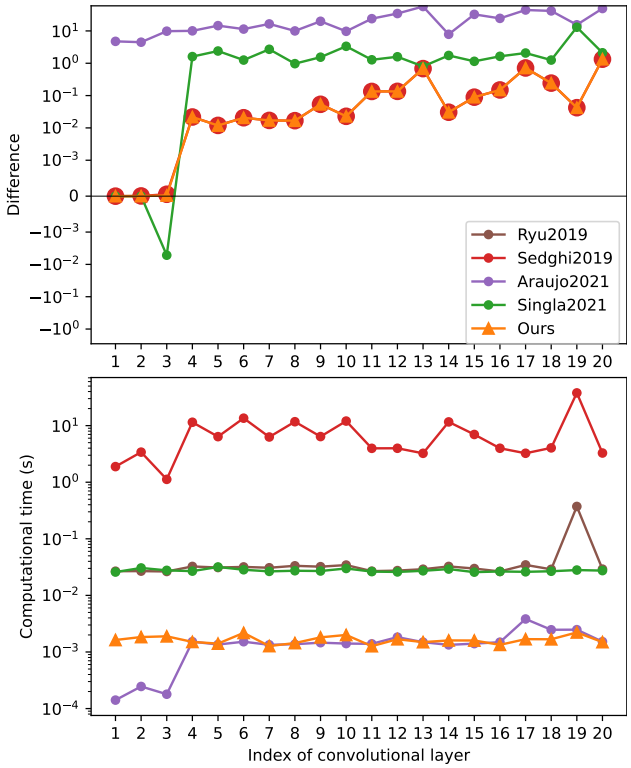


Figure 4. Comparison of Lipschitz bounds for all convolutional layers pre-trained ResNet18, reference is given by Ryu et al. (2019) method. Each convolutional layer in ResNet18 has a different number of output and input channels, kernel, and input spatial size. This experiment reproduces the usage of bounds on real kernel filters, we observe that our method gives the best precision of all and comparable time with Araujo et al. (2021).

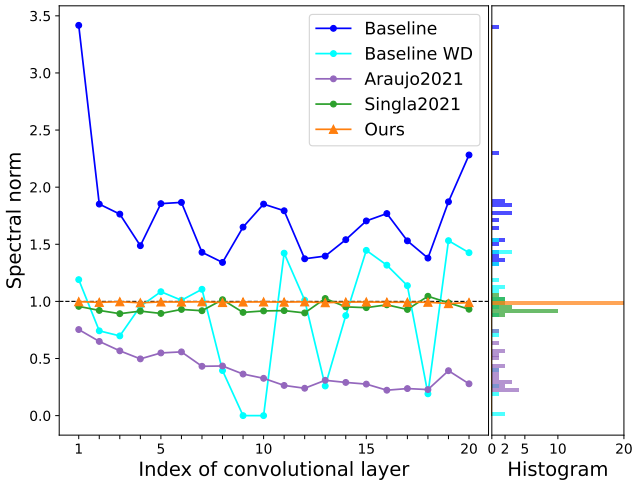


Figure 5. Plot and histogram of spectral norms for each convolution layer in ResNet18 at the end of training on CIFAR-10, for different regularization methods. We observe that the histogram of spectral norms regularized with our method is all at the target Lipschitz constant, meanwhile, other methods' histograms are scattered.

accurate (with low variance), fast and tractable for large input sizes and convolutional filters. Our result provides upper bound guarantees on the spectral norm. Furthermore, it allows the Lipschitz constant of convolutional layers to be precisely controlled by differentiation. It is also suitable for Lipschitz regularization during the training process of deep neural networks, bringing stability along with better classification accuracy.

Instead of focusing on the largest singular values, some works orthogonalize the weights (Li et al., 2019; Trockman et al., 2021), constraining all singular values to be equal to 1. We believe that our approach provides better expressiveness than orthogonalizing layers, as the spectrum can retain more information. Further work will consist of a theoretical study of the differences between the two approaches.

Acknowledgments

This work was performed using HPC resources from GENCI- IDRIS (Grant 2023-AD011014214) and funded by the French National Research Agency (ANR SPEED-20-CE23-0025).

References

Anil, C., Lucas, J., and Grosse, R. Sorting out lipschitz function approximation. In *International Conference on Machine Learning*, 2019.

Araujo, A., Negrevergne, B., Chevaleyre, Y., and Atif, J. On lipschitz regularization of convolutional layers using toeplitz matrix theory. *AAAI Conference on Artificial Intelligence*, 2021.

Arnoldi, W. E. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of applied mathematics*, 9(1):17–29, 1951.

Bartlett, P. L., Foster, D. J., and Telgarsky, M. J. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, 2017.

Bhatia, R. *Matrix analysis*, volume 169. Springer Science & Business Media, 2013.

Bibi, A., Ghanem, B., Koltun, V., and Ranftl, R. Deep layers as stochastic solvers. In *International Conference on Learning Representations*, 2019.

Cisse, M., Bojanowski, P., Grave, E., Dauphin, Y., and Usunier, N. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, 2017.

Farnia, F., Zhang, J., and Tse, D. Generalizable adversarial training via spectral normalization. In *International Conference on Learning Representations*, 2019.

Fazlyab, M., Robey, A., Hassani, H., Morari, M., and Pappas, G. Efficient and accurate estimation of lipschitz constants for deep neural networks. In *Advances in Neural Information Processing Systems*, 2019.

Golub, G. H. et al. Eigenvalue computation in the 20th century. *Journal of Computational and Applied Mathematics*, 2000.

Gouk, H., Frank, E., Pfahringer, B., and Cree, M. J. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 2021.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Henderson, H. V. and Searle, S. R. The vec-permutation matrix, the vec operator and kronecker products: A review. *Linear & Multilinear Algebra*, 9:271–288, 1981.

Huang, L., Liu, L., Zhu, F., Wan, D., Yuan, Z., Li, B., and Shao, L. Controllable orthogonalization in training dnns. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Jain, A. K. Fundamentals of digital image processing. In *Fundamentals of digital image processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

Jordan, M. and Dimakis, A. G. Exactly Computing the Local Lipschitz Constant of ReLU Networks. In *Advances in Neural Information Processing Systems*, 2020.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

Kublanovskaya, V. On some algorithms for the solution of the complete eigenvalue problem. *USSR Computational Mathematics and Mathematical Physics*, 1(3):637–657, 1962.

Lanczos, C. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 1950.

Latorre, F., Rolland, P., and Cevher, V. Lipschitz constant estimation of neural networks via sparse polynomial optimization. In *International Conference on Learning Representations*, 2020.

Lehoucq, R. B. et al. Deflation techniques for an implicitly restarted Arnoldi iteration. *SIAM Journal on Matrix Analysis and Applications*, 17(4):789–821, 1996.

Li, Q., Haque, S., Anil, C., Lucas, J., Grosse, R. B., and Jacobsen, J.-H. Preventing gradient attenuation in lipschitz constrained convolutional networks. In *Advances in Neural Information Processing Systems*. 2019.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

Pfister, L. and Bresler, Y. Bounding multivariate trigonometric polynomials. *IEEE Transactions on Signal Processing*, 67(3):700–707, 2019.

Ryu, E. K., Liu, J., Wang, S., Chen, X., Wang, Z., and Yin, W. Plug-and-play methods provably converge with properly trained denoisers. In *International Conference on Machine Learning*, 2019.

Sedghi, H., Gupta, V., and Long, P. The singular values of convolutional layers. In *International Conference on Learning Representations*, 2019.

Shi, Z., Wang, Y., Zhang, H., Kolter, J. Z., and Hsieh, C.-J. Efficiently Computing Local Lipschitz Constants of Neural Networks via Bound Propagation. *Advances in Neural Information Processing Systems*, 2022.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Singla, S. et al. Fantastic four: Differentiable and efficient bounds on singular values of convolution layers. In *International Conference on Learning Representations*, 2021.

Trockman, A. et al. Orthogonalizing convolutional layers with the cayley transform. In *International Conference on Learning Representations*, 2021.

Tsuzuku, Y., Sato, I., and Sugiyama, M. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *Advances in Neural Information Processing Systems*, 2018.

Virmaux, A. and Scaman, K. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Advances in Neural Information Processing Systems*, 2018.

Wang, J., Chen, Y., Chakraborty, R., and Yu, S. X. Orthogonal convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Yi, X. Asymptotic singular value distribution of linear convolutional layers. *arXiv preprint arXiv:2006.07117*, 2021.

Yoshida, Y. and Miyato, T. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941*, 2017.

Zhang, B., Jiang, D., He, D., and Wang, L. Rethinking Lipschitz Neural Networks and Certified Robustness: A Boolean Function Perspective. In *Advances in Neural Information Processing Systems*, 2022.

Zhang, H., Zhang, P., and Hsieh, C.-J. RecurJac: An Efficient Recursive Algorithm for Bounding Jacobian Matrix of Neural Networks and Its Applications. *AAAI Conference on Artificial Intelligence*, 2019.

A. Iterative methods to estimate spectral norm

Input G is a rectangular complex matrix, $G \in \mathbb{C}^{p \times q}$. Concerning the termination criterion, N_{iter} is the maximal number of iterations, but it could be combined with a threshold on the update norm.

A.1. Power iteration

Power iteration is described in Algo. 1: it is a simple but non-deterministic method that may converge slowly depending on input G and on initialization of random vector $u \in \mathbb{C}^q$.

A.2. Gram iteration

A naive implementation of Gram iteration is given in Algo. 4: this simple method is deterministic. If each Gram iteration is more complex than the power one, the matrix-matrix product is highly optimized on GPU, providing a fast method. If we compare PI and GI, we see that in the latter we iterate on the operator represented by matrix G and modify it, whereas in PI operator remains the same.

Algorithm 4 : Gram_iteration_naive(G, N_{iter})

1: **Inputs** matrix: G, number of iterations: N_{iter}
2: **for** $1 \dots N_{\text{iter}}$
3: $\quad G \leftarrow G^* G$
4: $\sigma_1 \leftarrow \|G\|_F^{2^{-N_{\text{iter}}}}$
5: **return** σ_1

A.3. Gram iteration with Rescaling

Due to the fast increase of values, we observe overflow in Algo. 4. To avoid it, we introduce a rescaling by Frobenius norm at each iteration: $G \leftarrow G / \|G\|_F$.

Denoting $G^{(t)}$ the Gram iterate at iteration t, the result of Gram iteration with rescaling is:

$$\|G^{(N_{iter})}\|_F^{2^{-N_{\text{iter}}}} = \|(G^* G)^{2^{N_{\text{iter}}-1}}\|_F^{2^{-N_{\text{iter}}}} \;/\; \prod_{t=1}^{N_{iter}} \|G^{(t)}\|_F^{2^{-t+1}} = s_{2^{N_{\text{iter}}}}(G) \;/\; \prod_{t=1}^{N_{iter}} \|G^{(t)}\|_F^{2^{-t+1}} \;,$$

which is no longer equal to the expected $2^{N_{\text{iter}}}$-Schatten norm of G.

In order to unscale the result at the end of the method, scaling factors are cumulated into a variable r. Denoting $r^{(t)}$ the cumulated scaling factor at iteration t, we have for the last iteration:

$$r^{(N_{iter})} = \sum_{t=1}^{N_{iter}} 2^{N_{iter}-t+1} \log \|G^{(t)}\|_F \;.$$

At the end of the method, we must unscale by:

$$\exp\left(2^{-N_{\text{iter}}}\, r^{(N_{iter})}\right) = \prod_{t=1}^{N_{iter}} \|G^{(t)}\|_F^{2^{-t+1}} \;.$$

Finally, the result of Gram iteration with rescaling at each iteration and final unscaling is the right $2^{N_{\text{iter}}}$-Schatten norm of G:

$$\|G^{(N_{iter})}\|_F^{2^{-N_{\text{iter}}}} \exp\left(2^{-N_{\text{iter}}}\, r^{(N_{iter})}\right) = \|(G^* G)^{2^{N_{\text{iter}}-1}}\|_F^{2^{-N_{\text{iter}}}} = s_{2^{N_{\text{iter}}}}(G) \;.$$

Unscaling is crucial as it is required to obtain a Schatten norm, which remains a strict upper bound on the spectral norm at each iteration of the method.

Gram iteration with rescaling is described in Algo. 2 and can be applied on a dense layer as it is.

A.4. Largest singular vectors by Gram iteration

At the end of the Gram iteration, the largest singular vectors can be estimated. The largest right-singular vector $u \in \mathbb{C}^q$ can be estimated using the final matrix $G \in \mathbb{C}^{q \times q}$, as it is a projector on the space of maximal singular value. So any non-zero column of index i, $G_{:,i}$ will be multiples of u:

$$u \leftarrow G_{:,i} / \|G_{:,i}\|_2 \ . \tag{12}$$

Then, defining G_0 the initial matrix G, the largest left-singular vector $v \in \mathbb{C}^p$ is:

$$v \leftarrow G_0 u / \|G_0 u\|_2 \ . \tag{13}$$

Note that using deflation techniques with GI, it is possible to estimate all other singular values and associated singular vectors, as done classically with PI.

A.5. Largest eigenpair by Gram iteration

Now, input G is a square complex matrix, $G \in \mathbb{C}^{p \times p}$. We want to compute its largest eigenpair (λ_1, u_1), where λ_1 is the largest eigenvalue and u_1 the largest eigenvector.

Power iteration described in Algorithm 1 can be simply adapted to output this largest eigenpair. Steps 4 and 5 are replaced by: $u \leftarrow Gu/\|Gu\|_2$, and step 6 is replaced by: $\lambda_1 \leftarrow u^* G u / \|u\|_2$. At the end of PI, u is u_1.

Similarly, it is also possible to adapt the Gram iteration described in Algorithm 2. Step 5 is replaced by: $G \leftarrow GG$. Given the eigenvalue decomposition of $G = Q \Lambda Q^{-1}$, the tth iterate $G^{(t)} = G^{2^{t-1}} = Q \Lambda^{2^{t-1}} Q^{-1}$. The same results applied for this alternative iteration as for Gram iteration, indeed proof of Section B still holds.

Since Gram matrix $G^{(2)} = {G^{(1)}}^* G^{(1)}$ is Hermitian, *i.e.* ${G^{(2)}}^* = G^{(2)}$, both steps 5 are equivalent from the second iteration: ${G^{(t)}}^* G^{(t)} = G^{(t)} G^{(t)}$, $\forall t \geq 2$.

B. Proofs of Section 4.2

Singular value decomposition of $G \in \mathbb{C}^{p \times q}$ is written $V_1 \Sigma V_2^*$, where $\Sigma = \mathrm{diag}(\sigma)$, $\sigma = (\sigma_i)_{1 \leq i \leq m}$ and $m = \min(p, q)$. We use the decreasing order indexation convention: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m$.
We have: $\sigma_1(G) = \sigma_1 = \|\sigma\|_\infty$.
We get: $G^{(1)} = G$, and $\forall t \geq 2$: $G^{(t)} = V_2 \Sigma^{2^{t-1}} V_2^*$.

B.1. Proof of Theorem 3

Proposition. *An upper bound of the spectral norm is:*

$$\sigma_1(G) \leq \|G^{(t)}\|_F^{\ 2^{1-t}} \ .$$

Proof. We start using this property on the largest singular value:

$$\begin{aligned} \sigma_1(G)^2 &= \sigma_1(G^* G) \\ \sigma_1(G) &= \sigma_1(G^* G)^{\frac{1}{2}} \ . \\ \text{Iterating } t \text{ times:} \quad \sigma_1(G) &= \sigma_1(G^{(t)})^{2^{1-t}} \\ \sigma_1(G) &\leq \|G^{(t)}\|_F^{\ 2^{1-t}} \ . \end{aligned}$$

□

Proposition. *The bound sequence converges to the spectral norm:*

$$\|G^{(t)}\|_F^{2^{1-t}} \underset{t \to +\infty}{\longrightarrow} \sigma_1(G) \ .$$

Proof. For $t \geq 1$,

$$\begin{aligned}
\|G^{(t)}\|_F^2 &= \mathrm{Tr}({G^{(t)}}^* G^{(t)}) \\
\|G^{(t)}\|_F^2 &= \mathrm{Tr}(G^{(t+1)}) \\
\|G^{(t)}\|_F^2 &= \mathrm{Tr}(V_2 |\Sigma|^{2^t} V_2^*) \\
\|G^{(t)}\|_F^2 &= \mathrm{Tr}(V_2^* V_2 |\Sigma|^{2^t}) \\
\|G^{(t)}\|_F^2 &= \mathrm{Tr}(|\Sigma|^{2^t}) \\
\|G^{(t)}\|_F^2 &= \sum_{i=1}^{m} |\sigma_i|^{2^t}
\end{aligned}$$

$$\begin{aligned}
(||G^{(t)}||_F^2)^{2^{-t}} &= \left(\sum_{i=1}^{m} |\sigma_i|^{2^t} \right)^{2^{-t}} \\
||G^{(t)}||_F^{2^{1-t}} &= \|\sigma\|_{2^t} \underset{t \to +\infty}{\longrightarrow} \|\sigma\|_\infty
\end{aligned}$$

and $\|\sigma\|_\infty = \sigma_1(G)$. □

Definition 1. *Q-convergences.*
Suppose sequence $(u_t)_{t \in \mathbb{N}}$ converges to $\hat{u}$, it converges Q-linearly if $\lim_{t \to +\infty} \frac{|u_{n+1} - \hat{u}|}{|u_n - \hat{u}|} = \mu < 1$.
It converges Q-superlinearly if $\mu = 0$. The number μ is called the rate of convergence.
It converges Q-quadratically if $\lim_{t \to +\infty} \frac{|u_{n+1} - \hat{u}|}{|u_n - \hat{u}|^2} = 0$.

Proposition. *The convergence of bound sequence is Q-superlinear of order $2 - \epsilon$, for ϵ arbitrary small (quasi quadratic convergence in practice). When $\sigma_1(G) = \sigma_2(G)$, convergence is Q-linear of rate $\frac{1}{2}$.*

Proof. To analyze convergence we compute the quotient and look at its limit to conclude on convergence type.

$$\begin{aligned}
\|\sigma\|_{2^t} - \|\sigma\|_\infty &= \left(\sum_{i=1}^{m} |\sigma_i|^{2^t} \right)^{2^{-t}} - \|\sigma\|_\infty \\
\|\sigma\|_{2^t} - \|\sigma\|_\infty &= \left(\left(\sum_{i=1}^{m} \left| \frac{\sigma_i}{\sigma_1} \right|^{2^t} \right)^{2^{-t}} - 1 \right) \|\sigma\|_\infty .
\end{aligned}$$

$$\begin{aligned}
\text{We define} \quad \epsilon_t &= \left(\sum_{i=1}^{m} \left| \frac{\sigma_i}{\sigma_1} \right|^{2^t} \right)^{2^{-t}} - 1 \\
\epsilon_t &= \exp\left(2^{-t} \log \left(\sum_{i=1}^{m} \left| \frac{\sigma_i}{\sigma_1} \right|^{2^t} \right) \right) - 1 \\
\epsilon_t &= \exp\left(2^{-t} \log \left(1 + \sum_{i=2}^{m} \left| \frac{\sigma_i}{\sigma_1} \right|^{2^t} \right) \right) - 1 .
\end{aligned}$$

With $\alpha = \sum_{i=2}^{m} \left| \frac{\sigma_i}{\sigma_1} \right|^{2^t} \underset{t \to +\infty}{\longrightarrow} 0$, using series expansion:

$$\begin{aligned}
\epsilon_t &\underset{t \to +\infty}{=} \exp\left(2^{-t} \left(\alpha + o(\alpha) \right) \right) - 1 \\
\epsilon_t &\underset{t \to +\infty}{=} 2^{-t} \alpha + o(2^{-t} \alpha) .
\end{aligned}$$

For $q > 0$,

$$\frac{\epsilon_{t+1}}{\epsilon_t^q} \underset{t\to+\infty}{\sim} \frac{2^{-(t+1)} \sum_{i=2}^m \left|\frac{\sigma_i}{\sigma_1}\right|^{2^{t+1}}}{2^{-tq}\left(\sum_{i=2}^m \left|\frac{\sigma_i}{\sigma_1}\right|^{2^t}\right)^q}$$

$$\frac{\epsilon_{t+1}}{\epsilon_t^q} \underset{t\to+\infty}{\sim} 2^{t(q-1)-1} \frac{\left|\frac{\sigma_2}{\sigma_1}\right|^{2^{t+1}}}{\left|\frac{\sigma_2}{\sigma_1}\right|^{q2^t}}$$

$$\frac{\epsilon_{t+1}}{\epsilon_t^q} \underset{t\to+\infty}{\sim} 2^{t(q-1)-1} \left|\frac{\sigma_2}{\sigma_1}\right|^{2^t(2-q)} .$$

When $\sigma_1 > \sigma_2$, for $0 < q < 2$, $\lim_{t\to+\infty} \frac{\epsilon_{t+1}}{\epsilon_t^q} = 0$. Hence convergence is Q-superlinear of order $q = 2 - \epsilon$, where ϵ is arbitrarily small, providing a quasi-quadratic convergence in practice.

For the special case $\sigma_1 = \sigma_2$, convergence is Q-linear ($q = 1$) of rate $\frac{1}{2}$.

□

B.2. Proof of Proposition 1

To implement explicit differentiation instead of auto-differentiation, we need to compute the gradient of the spectral bound given by Gram iteration: $||G^{(t)}||_F^{2^{1-t}}$.

Compute the gradient of $G \mapsto ||G^{(t)}||_F^{2^{1-t}}$.

For $t = 1$, $G^{(1)} = G$,

$$\frac{\partial \|G\|_F}{\partial G} = \frac{G}{\|G\|_F} .$$

For $t \geq 2$, note that $G^{(t)} = (G^*G)^{2^{(t-2)}}$. First, we calculate:

$$\frac{\partial \|G^{(t)}\|_F^2}{\partial G} = 2^t G (G^*G)^{2^{t-1}-1} .$$

Then,

$$\frac{\partial (\|G^{(t)}\|_F^2)^{1/2^t}}{\partial G} = \frac{1}{2^t} (\|G^{(t)}\|_F^2)^{1/2^t-1} 2^t \; G(G^*G)^{2^{t-1}-1}$$

$$\frac{\partial (\|G^{(t)}\|_F^2)^{1/2^t}}{\partial G} = (\|G^{(t)}\|_F)^{1/2^{t-1}-2} \; G(G^*G)^{2^{t-1}-1}$$

$$\frac{\partial (\|G^{(t)}\|_F^2)^{1/2^t}}{\partial G} = \frac{G(G^*G)^{2^{t-1}-1}}{(\|G^{(t)}\|_F)^{2(1-2^{-t})}} .$$

Finally using the chain rule, the gradient is given for $t \geq 1$:

$$\frac{\partial (\|G^{(t)}\|_F^2)^{1/2^t}}{\partial G} = \frac{G(G^*G)^{2^{t-1}-1}}{(\|G^{(t)}\|_F)^{2(1-2^{-t})}} . \tag{14}$$

To avoid overflow in differentiation, one can rescale G at the start of the gradient calculation. As the spectral norm $\|G\|_2$ has been computed during the forward pass, it can be used for rescaling. Then, calculate the gradient with Equation (14), and finally unscale the result by the factor: $1/\|G\|_2$.

C. Approximations for convolutional layers

C.1. Approximation for lower input spatial size $n_0 \leq n$

A bound on spectral norm can be produced with Gram iteration as in Proposition 2:

$$\sigma_1(W) = \max_{1 \leq i \leq n^2} \sigma_1(D_i) \leq \max_{1 \leq i \leq n^2} \|D_i^{(t)}\|_F^{2^{1-t}} .$$

To consider the very large input spatial size n, we can use our bound by approximating the spatial size for $n_0 \leq n$. It means we pad the kernel K to match the spatial dimension $n_0 \times n_0$, instead of $n \times n$. To compensate for the error committed by the sub-sampling approximation, we multiply the bound by a factor α. The work of Pfister & Bresler (2019) analyzes the quality of approximation depending on n_0 and gives an expression for factor $\alpha = \frac{2\lfloor k/2 \rfloor}{n_0}$ to compensate and ensure to remain an upper-bound, as studied in (Araujo et al., 2021).

C.2. Approximation between circular and zero paddings

In CNN architectures, zero padding is more commonly used than circular padding. To evaluate the potential approximation gap between these two methods, we compared our approach with the zero padding approach of (Ryu et al., 2019) in terms of error. As shown in Figure 3a and Figure 3b of our paper, our results demonstrate that the error with respect to the zero padding case remains low. However, for high kernel sizes, as depicted in Figure 3c, our approach may not be as accurate as that of (Ryu et al., 2019). It is important to note that the typical kernel size for deep learning applications is relatively small, usually lower than 7.

An empirical comparison between zero and circular padding can be obtained by examining the table of Lipschitz constant estimations for all convolutional layers of ResNet18 in Table 7 of Appendix F. We compare the values given by both methods: zero padding (Ryu et al., 2019) used as reference and, circular padding (Sedghi et al., 2019)/Ours. We observe that the Lipschitz constant values obtained from both methods are quite similar. This provides a strong assessment of the approximation gap between circular and zero padding for convolutional layers encountered in ResNet.

(Yi, 2021) provides an asymptotic bound on the singular values of circular convolution and zero padding convolutions in relation to the input spatial size. This, along with the findings from (Araujo et al., 2021), can aid in establishing an upper bound on the spectral norm of Toeplitz matrices, which represents zero padding convolutions, by leveraging the spectral norm of circulant matrices, which represents circular padding ones. Based on empirical results, the difference in the Lipschitz constant between circular and zero padding increases with kernel size and decreases with input size. Specifically, Figure 3b shows that as the input spatial size increases, the gap between (Ryu et al., 2019) and (Sedghi et al., 2019)/our approach decreases, while the gap widens as kernel size increases.

D. Additional figures and tables for computation of spectral norms on GPU

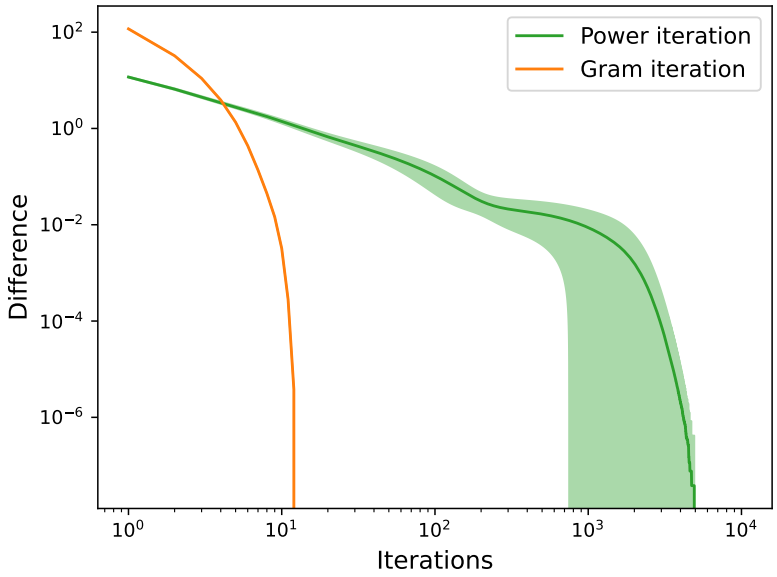


Figure 6. Convergence plot in log-log scale for spectral norm computation, comparing Power iteration and Gram iteration, one standard deviation shell is represented in a light color. Difference at each iteration is defined as $|\sigma_{1\,\text{method}} - \sigma_{1\,\text{ref}}|$ where reference is taken from PyTorch. Gram iteration converges to numerical 0 in less than 12 iterations while Power iteration requires up to 5,000.

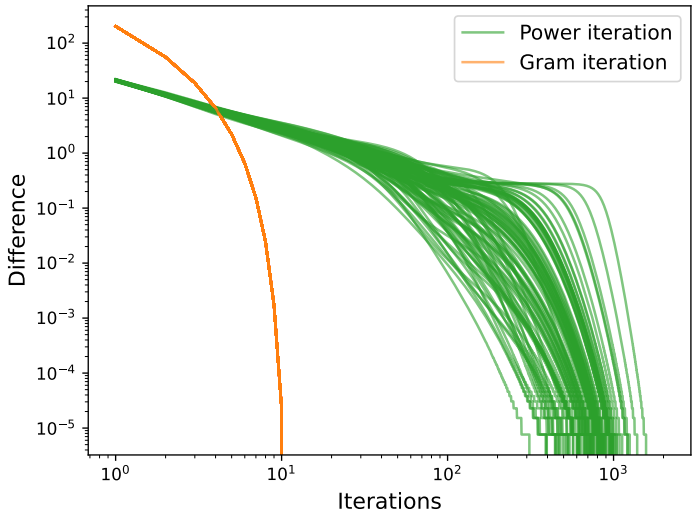


Figure 7. Convergence plot in log-log scale for spectral norm computation, comparing power iteration and Gram iteration, the same matrix is used here, each line corresponds to one run of Power iteration and Gram iteration. The difference is defined as $|\sigma_{1\,\text{method}} - \sigma_{1\,\text{ref}}|$ where reference is taken from PyTorch. We see that Power iteration has inherent randomness by design whereas Gram iteration is fully deterministic.

Table 5. Numerical values associated to Figure 2: error ratios and computational times of methods for spectral norm computation. Error ratio is defined as $\sigma_{1_{\text{method}}}/\sigma_{1_{\text{ref}}} - 1$.

Method	Error ratio	Computational time (s)
Reference	0	1.47e+0
PI (10 iters)	-3.27e-2 ± 3.55e-3	1.12e-3
PI (100 iters)	-1.41e-3 ± 1.75e-3	1.03e-2
PI (1000 iters)	-2.43e-6 ± 4.01e-6	1.04e-1
Ours GI (10 iters)	6.75e-6	1.83e-3
Ours GI (11 iters)	4.73e-8	1.99e-3
Ours GI (12 iters)	4.33e-12	2.13e-3

Table 6. Times and difference to reference, for different $p \times p$ matrices, averaged 10 times. Error is defined as $\sigma_{\text{method}} - \sigma_{\text{ref}}$. To have comparable time and precision, 100 iterations were taken for Power iteration and 10 for Gram iteration. Gram iteration is overall more competitive than Power iteration for all matrix dimensions considered.

Dimension p	PI time (s)	GI time (s)	PI error	GI error	Ref
10	0.0093	0.0012	0.0006	0	5.28
100	0.0083	0.0010	-0.0231	0	19.31
1000	0.0093	0.0017	-0.0426	0.0005	62.95
10000	0.016	0.0019	-0.6004	0.0037	199.77
50000	0.017	0.0040	-1.70	0.1	447.21

E. Lipschitz constant of CNN

Works of (Tsuzuku et al., 2018; Cisse et al., 2017) studied the Lipschitz constant of neural network by proposing for each kind of layer a formula or method to bound its Lipschitz constant and then the network overall Lipschitz constant using Equation (2).

A dense layer can be represented as matrix $W \in \mathbb{R}^{m \times n}$. The Lipschitz constant for the ℓ_2-norm equals to $\sigma_1(W)$. One can use SVD, power iteration, or Gram iteration to compute it. Most of the activation functions used in deep learning such as ReLu, sigmoid, and hyperbolic tangent are 1-Lipschitz. As mentioned in Appendix D.3 of (Tsuzuku et al., 2018) the max-pooling can be seen as a max operation followed by a pooling operation. The Lipschitz constant of the max function is bounded by 1, hence bound on the Lipschitz constant of max-pooling is bounded by the Lipschitz constant of the pooling operation. The Lipschitz constant of the pooling operation is bounded by the maximum number of repetitions of a pixel in the input. For a kernel of spatial size k the number of maximum repetitions is k^2, taking into account the spatial size n of the input and the stride the Lipschitz constant is bounded by (Tsuzuku et al., 2018):

$$\left\lceil \frac{\min(k, n-k+1)}{\text{stride}} \right\rceil^2 .$$

Lipschitz constant of batch normalization is estimated in Appendix C.4.1 of (Tsuzuku et al., 2018):

$$L = \max_i \{ |\gamma_i| / \sqrt{\sigma_i^2 + \epsilon} \} ,$$

where, in this equation, γ_i and σ_i^2 are respectively the scale parameter and the variance of the batch. This Lipschitz constant is rarely considered in the global constant of networks.

Residual connections are present in ResNet (He et al., 2016) architectures. In order to compute a bound on the Lipschitz constant of such networks, one can use the following rules as in (Tsuzuku et al., 2018) and (Gouk et al., 2021). For g, f functions of Lipschitz constant of L_1, L_2 respectively, the Lipschitz constant for an addition $f + g$ is bounded by $L_1 + L_2$, and for composition $f \circ g$, it is bounded by $L_1 L_2$. We call Φ_l the l-th composition of layers separated by residual connections, *i.e.* $\Phi_l = \phi_{l,1} \circ \phi_{l,2} \circ \dots$ and $\text{Lip}(\Phi_l) \leq \text{Lip}(\phi_{l,1}) \, \text{Lip}(\phi_{l,2}) \dots$. We call L_l the Lipschitz constant of the truncated network before block Φ_l, then we have $L_{l+1} \leq L_l + \text{Lip}(\Phi_l)$.

F. Additional table for estimation on convolutional layers of CNNs

Table 7. Comparison of Lipschitz bounds for all convolutional layers ResNet18, reference for real Lipschitz bound is given by (Ryu et al., 2019) method. We observe that our method gives close value to Sedghi et al. (2019) up to two decimal digits while being significantly faster. We remark that convolution filter $512 \times 512 \times 1 \times 1$ Singla et al. (2021) value 2.03 underestimates reference value 2.05.

Filter Shape	**Lipschitz Bound**					**Running Time (s)**				
	Ours	Singla2021	Araujo2021	Sedghi2019	Ryu2019	Ours	Singla2021	Araujo2021	Sedghi2019	Ryu2019
$64 \times 3 \times 7 \times 7$	15.91	28.89	22.25	15.91	15.91	0.0024	0.0507	0.0003	18.06	0.7579
$64 \times 64 \times 3 \times 3$	6.00	9.32	17.59	6.00	6.00	0.0014	0.0264	0.0003	7.47	1.07
$64 \times 64 \times 3 \times 3$	5.34	6.28	15.31	5.34	5.33	0.0015	0.025	0.0012	7.29	1.07
$64 \times 64 \times 3 \times 3$	7.00	8.71	16.46	7.00	7	0.0011	0.026	0.0003	7.27	1.08
$64 \times 64 \times 3 \times 3$	3.82	5.40	13.74	3.81	3.82	0.0014	0.025	0.0003	7.19	1.08
$128 \times 64 \times 3 \times 3$	4.71	5.98	16.35	4.71	4.71	0.0014	0.026	0.0012	9	0.473
$128 \times 128 \times 3 \times 3$	5.72	7.21	23.58	5.72	5.72	0.0011	0.026	0.0013	4.85	0.78
$128 \times 64 \times 1 \times 1$	1.91	1.91	6.39	1.91	1.91	0.0014	0.0333	0.0001	9.25	0.042
$128 \times 128 \times 3 \times 3$	4.39	6.78	21.93	4.39	4.41	0.0011	0.026	0.0003	4.844	0.78
$128 \times 128 \times 3 \times 3$	4.89	7.57	19.79	4.89	4.88	0.0014	0.027	0.0004	4.78	0.78
$256 \times 128 \times 3 \times 3$	7.39	8.45	35.86	7.39	7.39	0.0014	0.026	0.0017	5.36	0.46
$256 \times 256 \times 3 \times 3$	6.58	8.03	37.98	6.58	6.58	0.0014	0.026	0.0025	3.46	0.85
$256 \times 128 \times 1 \times 1$	1.21	1.21	5.97	1.21	1.21	0.0014	0.036	0.0001	5.96	0.04
$256 \times 256 \times 3 \times 3$	6.36	7.57	30.07	6.36	6.35	0.001	0.026	0.0004	3.48	0.85
$256 \times 256 \times 3 \times 3$	7.68	9.17	29.19	7.68	7.67	0.0011	0.027	0.0004	3.48	0.85
$512 \times 256 \times 3 \times 3$	9.99	11	46.24	9.99	9.92	0.0011	0.026	0.004	3.55	0.45
$512 \times 512 \times 3 \times 3$	9.09	10.45	47.59	9.09	9.04	0.0014	0.026	0.0001	2.8	0.79
$512 \times 256 \times 1 \times 1$	2.05	2.03	11.87	2.05	2.05	0.0012	0.024	0.0001	3.77	0.052
$512 \times 512 \times 3 \times 3$	17.60	18.37	59.04	17.60	17.5	0.001	0.027	0.0005	2.79	0.79
$512 \times 512 \times 3 \times 3$	7.48	7.6	57.33	7.48	7.43	0.0014	0.027	0.0006	2.83	0.79

G. Additional table and figure for regularization of convolutional layers of ResNet

The naive auto differentiation indeed uses a lot amount of GPU memory, but the explicit differentiation using Proposition 1 mitigates this matter as the computational graph for the gradient is smaller, as reported in Table 8.

Table 8. Differentiation memory footprints in order to perform regularization for all convolutional layers in the model, for an input image of size 224×224. The number of Gram iterations is 6.

Model	Autodiff Mem GPU (MB)	Explicit Mem GPU (MB)
ResNet18	9770	4322
ResNet34	17152	7088
ResNet50	$< 42\,000$	27535

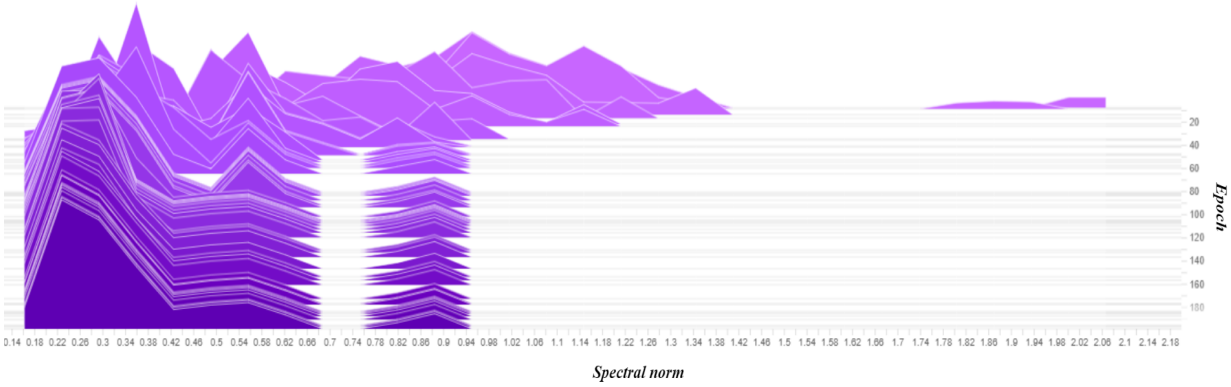


Figure 8. Spectral norm histograms in the function of epoch for training on CIFAR10 with regularization using Araujo2021.

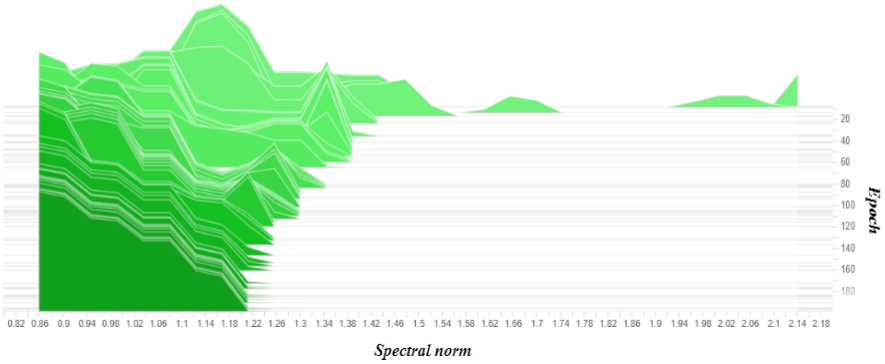


Figure 9. Spectral norm histograms in the function of epoch for training on CIFAR10 with regularization using Singla2021.

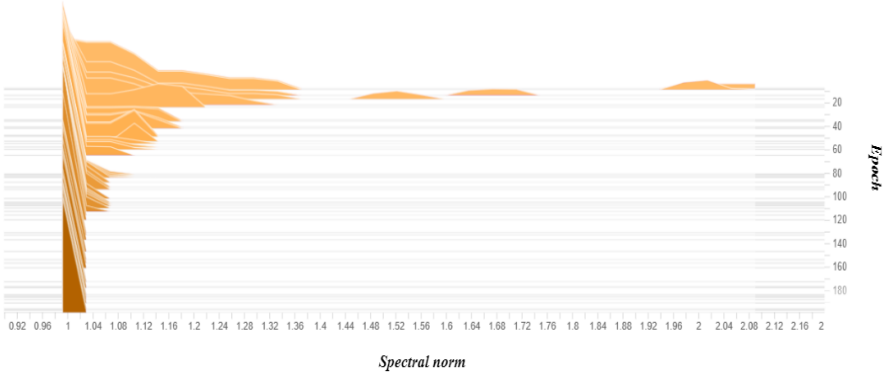


Figure 10. Spectral norm histograms in the function of epoch for training on CIFAR10 with regularization using our method.