Towards an Evaluation of Lipschitz Constant Estimation Algorithms by building Models with a Known Lipschitz Constant

William Piat, Jalal Fadili, Frédéric Jurie, Sébastien da Veiga

Towards an Evaluation of Lipschitz Constant Estimation Algorithms by building Models with a Known Lipschitz Constant

William Piat[1,2], Jalal Fadili[2], Frédéric Jurie[1], and Sébastien Da Veiga[1]

[1] Safran Tech, Digital Sciences and Technologies Department, Rue des Jeunes Bois, Châteaufort, 78114 Magny-Les-Hameaux, France.
[2] Normandie Univ, ENSICAEN, CNRS, GREYC, Caen, France

Abstract. We present a consistent framework for evaluating algorithms assessing the Lipschitz constant of neural networks. Lipschitz constants are tightly linked to robustness in the sense that they bound the overall amplification of the vector processed through the network, in other words it directly correlates the changes on the prediction with changes on the input. Controlling it precisely during the training of statistical model is still an open problem and is key to certifying neural networks. Our research is motivated by the fact that numerous algorithms exist for computing Lipschitz constant and that papers tend to only compare the proposed algorithms against each other without the knowledge of the actual target value. This is sound if one wants to showcase the differences in performance and accuracy between algorithms, however there exist no reference test where the target is known beforehand. We provide here, to the best of our knowledge, the first procedure to generate general networks with a known target Lipschitz constant. This methodology enables to gain some insight on the results of Lipschitz assessing algorithms, both on the behavior with respect to dimension and their behavior when probed near the boundaries of their validity domain.

1 Introduction and related works

Robustness of neural networks, in particular their robustness to adversarial examples, has been the subject of many recent works, given their increased usage in critical applications (applications that involve a high level of validation). It has indeed been shown that deep neural networks are very sensitive to adversarial attacks [1,2], which are small crafted noise, usually unable to fool a human, that are able to completely perturb the predictions made by deep neural networks. Ultimately the lack neural networks robustness leads to little or no use in critical applications to date.

A measure of the robustness to such input perturbations is the Lipschitz constant as it play an important role in generalization bounds [3] that describe the generalization capacity of a broad class of function: The smaller the Lipschitz constant the smaller the gap is bound to be between the training error and the testing error. Computing the Lipschitz constant answers two specific needs: the first one is the need for certification where the goal is to insure that around specific datapoints the prediction does not change. The second one is the need for robustness that is to insure more stability to perturbation

on any sample of the data distribution. Measuring and controlling the Lipschitz constant have hence become major challenges to characterize the robustness or to build robust neural networks [4]. Indeed, the lack of robustness can be seen as the consequence of uncontrolled Lipschitz constant during training, which allows a small perturbation to cause large changes in the decision function.

Estimating the Lipschitz constant of a neural network is a NP hard problem [5] therefore different approaches are often facing an accuracy/complexity dilemma. Loose estimations of the Lipschitz constant [6] used simple product of operator norms. It was then refined: Seqlip [5] is reformulating the estimation into a maximisation problem on activation variables that if solved exactly provides an upper bound for the Lipschitz constant. SDP relaxation such as LipSDP [7] or [8] give a true certified upper bound with a heavy computational cost. Computing the Lipschitz constant of ReLU networks can also be cast as polynomial optimization, for which the sparse version of the Lasserre's SDP-hierarchy was proposed in [9] to provide upper bounds that are sometimes strict improvement over previously known upper bounds.

A common ground of approaches for estimating the constant concerns the tests they are performing: given random or trained networks the methods are probed one against another and the one providing the lowest constant has be the one performing the best (only if the estimation is a true upper bound). Although interesting for comparing methods this simply does not answer basic questions: how close the estimation is to the real Lipschitz constant? What are the dependencies of the error?

The main motivation for this paper is to provide procedures for designing diverse test cases (i.e., diverse neural networks) for which the Lipschitz constant is known. This will allow us to quantitatively compare and qualify algorithms for estimating Lipschitz constants. The main difficulty is that building a general network that has a specific Lipschitz constant is an NP hard problem.

2 Contributions

Section 4, provides an exact formulation of the problem of prescribing a Lipschitz constant that can be solved approximately, this is to the best of our knowledge the first attempt at designing a prescribed Lipschitz network without the use of Lipschitz layers [10,11]. We probe the effectiveness of our approach by crafting neural networks with different values of Lipschitz constant and show small error despite the lack of convergence guaranties of our algorithm. In Section 5 we present a more naive approach that has the advantage of giving an exact Lipschitz constant but covers a smaller class of models this method is efficient, computationally speaking, and exact but it lacks the generality of the former. This method is then used to appreciate the approximation properties of Lipschitz assessing algorithms: we are then able to highlight effectively their respective properties and their dependencies to either input dimension and/or parameter dimension.

3 Notations and definitions

3.1 Neural network

Let $\mathcal{Z} \subset \mathbb{R}^p \times \mathbb{R}^q$ be a dataset with elements $(x, y) \in \mathcal{Z}$ where x is the input (explanatory variables) and y is the target. We wish to build a predictor in order to predict y from x.

Definition 1. *Given a non linear function $\phi : \mathbb{R} \to \mathbb{R}$ that can be applied element-wise on a vector, and a set of $n \in \mathbb{N}$ matrices $(W_i)_{i=1}^n$ with compatible dimensions so that the matrix product $W_1 W_2 ... W_n$ is well defined. A linear neural network with biases is a function $f : \Theta \times \mathcal{X} \to \mathbb{R}^q$, where $\mathcal{X} \to \mathbb{R}^p$ and Θ is the space of parameters ($\theta := (W_i, b_i)_{i \in [n]}$)*

$$f_1(\theta, x) := W_1 x + b_1, f_i(\theta, x) := W_i \phi(f_{i-1}(\theta, x)) + b_i, \forall i = 1, .., n, f(\theta, x) := f_n(\theta, x). \tag{1}$$

This function can be used to approximate y using x.

3.2 Lipschitz smoothness

The Lipschitz smoothness is a property that quantifies the variations of a non smooth function: the smaller the constant is, the less the function varies. This is closely intertwined with the notion of robustness as it give a direct way of upper bounding the changes of a function with respect to the changes in its arguments. This is overall a suitable indicator of a well regularized decision process and it would ideally be enforced or conditioned throughout a dedicated training.

Definition 2. *Let $X \subset \mathbb{R}^p, p \in \mathbb{N}$ and $Y \subset \mathbb{R}^q, q \in \mathbb{N}$, equipped with the usual norms, a function $f : \mathcal{X} \to \mathcal{Y}$ is called Lipschitz if there exists a constant L such that for all $(x, y) \in \mathcal{X} \times \mathcal{Y}$*

$$\|f(x) - f(y)\|_2 \leq L \|x - y\|_2. \tag{2}$$

Of course, any $L' > L$ is also a Lipschitz constant of f, for simplicity the minimal Lipschitz constant $L_{f,\mathcal{X}}$ of the function f is called the Lipschitz constant of f.

$L_{f,\mathcal{X}}$ can be expressed also as the upper bound of the differential quotient:

$$L_{f,\mathcal{X}} = \sup_{y,x \in X^2} \frac{\|f(x) - f(y)\|_2}{\|x - y\|_2}. \tag{3}$$

4 Enforcing a Lipschitz smoothness to a given network

In this section, we propose an approach that initializes a model and then optimize its parameters in order to fit a fixed target Lipschitz constant.

Let f be a neural network as defined in (1) In the case of a multi layer perceptron the Lipschitz constant of f over $\mathcal{X}$ in the norm $\|\cdot\|$ is:

$$L_{f,\mathcal{X}}(\theta) := \sup_{x \in \mathcal{X}} \left\| \left(\prod_{i=1}^{d-1} W_i^\top \operatorname{diag}(G_\phi(z_i)) \right) W_d^\top \right\|, \tag{4}$$

where $z_1 = x$, $z_i = W_i x_{i-1} + b_i$ for all $i \in \{2, ..., n\}$, and $G_\phi(z_i)$ is an appropriate generalized Jacobian of ϕ evaluated at z_i. Typically G is a conservative field [12] in which case $L_{f,\mathcal{X}}(\theta)$ is indeed a Lipschitz constant whenever $\mathcal{X}$ is convex [9]. When ϕ is differentiable this generalized gradient is the singleton of the Jacobian.

Our goal is the following:

$$\begin{array}{c} \text{Given a prescribed target Lipschitz constant } \bar{L} \\ \text{Find } \theta^* \text{ such that } L_{f,\mathcal{X}}(\theta^*) = \bar{L}. \end{array} \tag{5}$$

It is intended in (5) that such θ^* exists. Observe also that θ^* is not unique in general by obvious ambiguities (scale for instance). Thus, problem (5) can be solved only in an equivalence class.

Let us chose a loss function $\ell : \mathbb{R}^2_+ \to \mathbb{R}^+$ such that $\ell(a, b) = 0 \iff a = b$. One can then also formulate (5) as

$$\begin{array}{c} \min\limits_{\theta \in \Theta} \mathcal{L}(\theta, x, \bar{L}) \\ \text{where} \quad \mathcal{L}(\theta, x, \bar{L}) := \ell(L_{f,\mathcal{X}}(\theta), \bar{L}). \end{array} \tag{6}$$

If θ^* exists in the first problem then the minimum value in (6) is indeed 0. One can show using standard arguments that if $G_\phi(\cdot)$ is continuous (hence ϕ is C^1), that Θ is compact, the minimization problem (6) that the set of minimizers is a nonempty compact set (which does not mean that it is attained at 0). The C^1 assumption might be weakened but we will stick with this assumption for now. The compactness assumption on Θ also makes sense at least for some of the parameters to remove the ambiguities.

Remark 1. In any approach above, one has to appeal to some second-order information on ϕ, i.e., at some point to differentiate $G_\phi(z)$ wrt z. In turn, this necessitates a higher order smoothness on ϕ compared to our work on robust optimization [13] (which is not the case for the usual piece-wise linear activation functions).

4.1 Algorithm for solving (5)

A heuristic way to solve (5) is to alternatively maximize on x and then then apply a generalized (sub-)gradient descent step in θ. This is summarized in algorithm 1, where the inner maximization step is replaced in practice by a projected gradient ascent.

Algorithm 1: PGD-like pseudo-algorithm for approximating the prescription of a Lipschitz constant using spectral norm maximizing

Input: $\theta_1 = (W_i^1, b_i^1)_{i \in \{1,...,d\}}$ initial parameters of the network
Input: $\bar{L}$ the target Lipschitz constant
Input: $(\gamma_k)_{k \in \mathbb{N}}$ the decaying learning rate
for $k = 1, n$ **do**
  $x_k^* \in \text{Argmax}_{x \in \mathcal{X}} \, || \left(\prod_{i=1}^{d-1} {W_i^k}^\top \text{diag}(G_\phi(z_i)) {W_d^k}^\top \right) ||$;
  $\mathcal{L}(\theta_k, x_k^*, \bar{L}) = \ell \left(|| \left(\prod_{i=1}^{d-1} {W_i^k}^\top \text{diag}(G_\phi(z_{i,k}^*)) \right) {W_d^k}^\top ||, \bar{L} \right)$;
  $u_k \in G_{\mathcal{L}(\cdot, x_k^*, \bar{L})}(\theta_k)$;
  $\theta_{k+1} = \theta_k - \gamma_k u_k$;
return θ_n;

In algorithm 1, $G_{\mathcal{L}(\cdot,x,\bar{L})}(\theta)$ is a generalized derivative of $\mathcal{L}$ wrt variable θ evaluated at $(\theta, x, \bar{L})$. Clearly, it is hoped that algorithm (1) provides an approximation to a solution of problem (5), though we do not provide any guarantee on this. We are working towards establishing some of these guarantees.

In the next section, we provide an algorithm that allows to have a guaranteed structure of network with exactly $\bar{L}$ as a Lipschitz constant. It is however only valid for the very specific ReLU activation function.

5 Constructing known Lipschitz networks

Let us take the notation of equation (3): we can simply upper bound the Lipschitz constant as done in [6] by the product of the spectral norm in case the activation function ϕ is 1 Lipschitz:

$$L_{f,\mathcal{X}}(\theta) \leq \prod_{i=1}^{n} \|W_i\|_2 \tag{7}$$

Our goal is to tailor $(W_1, ..., W_n)$ so that this inequality become an equality.

5.1 Construction

Let us constraint our weights with the following conditions:

- for all $i \in [n]$, $W_i = O_i S_i O_{i-1}^\top \in \mathbb{R}^{m \times m}$, $m \in \mathbb{N}$, where S_i are diagonal matrices and O_i are square orthonormal matrices also of dimension m.
- Without loss of generality we can assume that the greatest value $\lambda_i^{max} = \max_j((S_i)_{j,j})$ is along the first dimension. We need to add another construction constraint here: we choose λ_i^{max} to be positive and to be the greatest eigenvalue in absolute norm (this ensures that no product of 2 negative eigenvalues will be greater that the products of the λ_i^{max}).

We have to make slight changes to the matrices $(O_k)_{k\in[n]}$ in order to have their main directions aligned along a positive dimension:

Let us write $O_i = (V_i^1, V_i^2, ...V_i^m)$ with column vectors, we have by definition that $W_i V_{i-1}^1 = \lambda_i^{max} V_i^1$.

We would like to have that for all $i \in \{1, ..., n\}$, V_i^1 has positive or null components (note that O_0 is not considered here) so that the ReLU activations do not change V_i^1 when it is applied element-wise. To achieve so we simply define:

$$O_i' := \text{diag}(\text{sign}^*(V_i^1))O_i = \text{diag}(\text{sign}^*(O_i e_1))O_i \tag{8}$$

where e_i is the i vector of the standard base of $\mathbb{R}^m$, sign^* is the element-wise function defined by:

$$\text{sign}^*(x) = \begin{cases} -1 & \text{if} \quad x < 0 \\ 1 & \text{otherwise} \end{cases}$$

For all $i \in \{1, ..., n\}$, O'_i is orthonormal as the transformation (8) is only a symmetry on carefully chosen axis. For the special case of O_0 we select $O'_0 = O_0$. Let us consider then the matrices $W'_i = O'_i S_i O'^{\top}_{i-1}$ and $f^*(x)$ neural network composed with the W'_i weight matrices (and null biases). We will now check that $L_{f^*,\mathcal{X}} = \prod_{i=1}^{n} \|W'_i\|_2$ with $\sigma = \text{ReLU}$.

5.2 Verification

Let $(W'_n ... W'_1)$ be the matrices as defined in the section 5.1 we will now check that we know the value of the Lipschitz constant of the crafted network and its main direction:

$$\|\prod_{i=1}^{n} W'_i\|_2 = \|O'_n \prod_{i=1}^{n} S_i O_0^{\top}\|_2 = \|\prod_{i=1}^{n} S_i\|_2 = \prod_{i=1}^{n} \lambda^i_{max} = \prod_{i=1}^{n} \|S_i\|_2 \geq L_{f'}$$

On top of that for all $i \in \{1, ..., n\}$ we have by construction $V_i^{1\prime}$ (ie the main direction of matrice i for the eigenvalue λ_i^{max}) whose components are positive or null and thus stable by the ReLU operation σ.

On the other hand we have for $x^* = V_0^1$:

$$\begin{aligned} f^*(x^*) &= W'_n \sigma(W'_{n-1} \sigma(W'_{n-2} ... \sigma(W'_1 x^*)) \\ &= W'_n \sigma(W'_{n-1} \sigma(W'_{n-2} ... \sigma(\lambda_1^{max} V'^1_1)) \\ &= W'_n \sigma(W'_{n-1} \sigma(W'_{n-2} ... \lambda_1^{max} V'^1_1) \\ &= \prod_{i=1}^{n} \lambda_i^{max} V'^1_n \end{aligned}$$

thus $L_{f^*} \geq \frac{\|f^*(x^*)\|_2}{\|x^*\|_2} = \prod_{i=1}^{n} \lambda^i_{max}$

Then $L_{f^*} = \prod_{i=1}^{n} \lambda^i_{max}$, we have successfully constructed a neural network with a known Lipschitz constant. The subspace of maximum amplification is a half straight line of the form $x = t.V_0^1$, $t > 0$.

Remark 2. We could have limited the construction by aligning only the first eigenvectors of the weight matrices and not the other ones, the verification would have remained the same. However the construction and algorithm 2 (presented in section 5.3) would have been more complex for little benefit on the class of models used.

5.3 Algorithm for designing Lipschitz test cases

The algorithm presented here is composed of 2 main components:

- A sampling operator in the set of orthonormal matrices [14] named Orthonormal_Sampling.
- An procedure capable to change an orthogonal matrix in order to make a column indexed i positive (it is only the implementation of equation (8) for a parametrizable column i). We named Positivation_by_symmetry in the algorithm

Algorithm 2: Building a 1-Lipschitz network with n layers

Input: Number of neurones per layer m
Input: number of layers n
$O_{old} = \text{Orthonormal\_Sampling}(m)$;
$V_0 = O_{old,1}$;
for $k = 1, n$ **do**
  $O_{new} = \text{Orthonormal\_Sampling}(m)$;
  $O_{new} = \text{Positivation\_by\_symmetry}(O_{new}, 1)$;
  $D = \text{Diag}((u_i)_{i\in\{1,2,...,m\}})$, $u_i \sim U(0,1)$;
  $D_{1,1} = 1$;
  $W_k = O_{new}.D.O_{old}^{\top}$;
  $O_{old} = O_{new}$;
return $V_0, W_1, W_2, .., W_n$;

Algorithm 2 gives the main directions, and gives matrices aligned along a positive direction making ReLUs applied between each layer unable to alter the output vectors. This allows the Lipschitz constant to be equal to the product of the spectral norms of the weight matrices: 1.

Remark 3. This algorithm can easily be changed for increasing the resulting model class broadness:

- Having any Lipschitz constant L by changing the eigenvalues
- Not aligning the last layer can add an extra difficulty for algorithms while not changing the main direction of the network, to that extend one should ensure that the eigenvalues of the last layer do not shrink the main direction compared to other directions which is feasible as there are no ReLUs after the last layer and thus the problem is linear. This does changes the Lipschitz constant that can be computed by processing the main direction and checking the norm increase
- Making different layer size on each layer to make more diverse structures

We kept the algorithm minimal to make it more understandable but some of these improvements may seem necessary for increasing the diversity of the networks generated however the most general networks are the one made by solving (6) which we discussed in the previous part

Remark 4. This is not an ideal case as this is not consistant with the behavior of random normalized layers (see [5] lemma 2). We are creating a very specific case where finding the Lipschitz constant can easily be achieved by finding the main direction of the first weight matrix. Therefore we are expecting any greedy algorithm to perform exceptionally well on these samples. Once again let us emphasize that using these networks for assessing performance and accuracy of an approach to compute the Lipschitz constant provides a biased view. However if a method fails this test it is a good indicator that the method, as it is implemented, has difficulties for approximating correctly a Lipschitz constant.

6 Experimental validation

6.1 Evaluation of Algorithm 1

We first evaluated the performance in enforcing a Lipschitz constant on a given network by building networks with different targets $\hat{L}$.

All the experiments were performed with a 6 layer linear neural network with 30 neurons on each layer. The input size is taken equal to the layer size: 30. We chose the ELU activation function as it is $\mathcal{C}^2$. We chose to prescribe the Lipschitz constant for the spectral norm thus we use power iteration for computing the operator norm. We noticed that enforcing a 1 spectral norm on all layers except the last one helps for convergence as it reduces the number of symmetries in the solution without loss of generality.

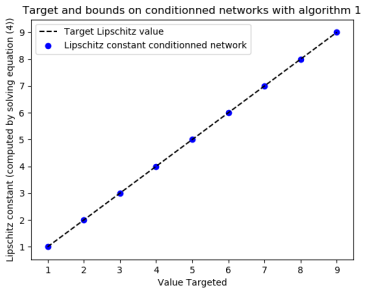

Fig. 1: Evolution of prescription of Lipschitz constant following algorithm 1 with objective (black) and Lipschitz constant of constrained network (blue)

Figure 1 presents the results using algorithm 1: the black curve is the objective that we target and the blue points are Lipschitz constant of the resulting networks. The values of the Lipschitz constant follows the tendency of the regularity we are enforcing even though the algorithm used provides no guaranties of achieving this much.

6.2 Comparing Lipschitz assessing algorithms

Having no guaranties on the convergence of any of the previous algorithms, we can hardly use them for testing Lipschitz assessing algorithm; this is why we will resort to the networks provided by algorithm 2 that ensures the exact value of the Lipschitz constant.

In this section, we present experiments in which we use this algorithm to compare Lipschitz assessing algorithms. As explained above, all the networks used in these experiments have a Lipschitz constant of 1.

Highlighting True/False bounds
Some algorithms are giving a True upper bound; this bound can be reasonably loose

depending on the approach used. Some other algorithms provide True lower bound that always return a value that is lower than the actual Lipschitz constant. Finally there exists also False upper/lower bounds that aims at approximating an upper/lower bound but with no guaranties to get a value that is above/below the Lipschitz constant. We test the following algorithms:

- The LipSDP-network algorithm [7] in Figure 2: the algorithm gives a ”true” upper bound
- The SeqLip algorithm [5] using a greedy approach in Figure 3, the algorithm gives an ”false” upper bound
- The SeqLip algorithm using a genetic optimizer in Figure 4, the algorithm gives a ”false” upper bound
- A ”true” lower bound computed by sampling uniformly the input space in Figure 5

The protocol of the experiment is the following: for a given layer size and depth, we build a network using Algorithm 2 and then assess the Lipschitz constant using all the methods enumerated above. We have chosen arbitrarily the dimension of the input to be equal to the size layer. We display the evolution of the estimation of the Lipschitz constant when making the layer size and depth vary for a given algorithm. We decided not to average the evaluation over multiple experiments because we are trying to display the flaws in given executions, not an average behavior. Advocating on general performance of the algorithms on neural networks would be biased considering the specific kind of networks that we are testing. Therefore we only want to highlight the dependencies between the methods and the characteristic dimensions of the neural network: dimension of the inputs and dimension of the parameters.

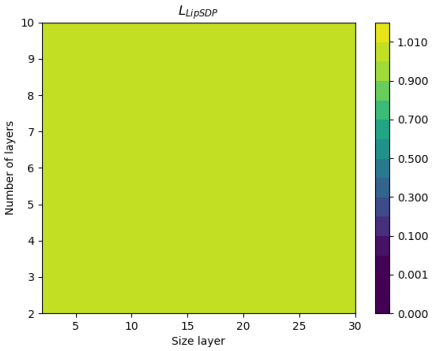

Fig. 2: Computation of the Lipschitz constant using the LipSDP algorithm for different depths (y axis) and different layer size (x axis).

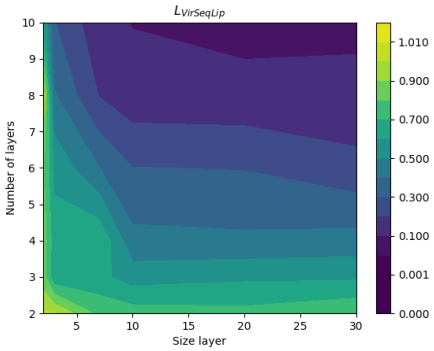

Fig. 3: Computation of the Lipschitz constant using the seqLip algorithm for different depths (y axis) and different layer size (x axis) with numpy optimizer.

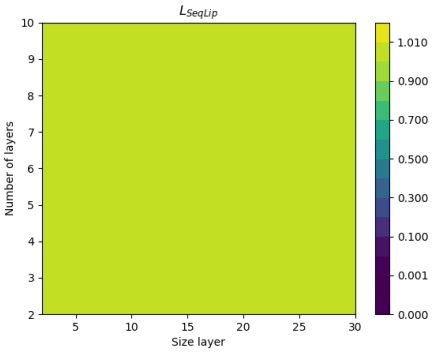

Fig. 4: Computation of the Lipschitz constant using the seqLip algorithm for different depths (y axis) and different layer size (x axis) with genetic optimizer.

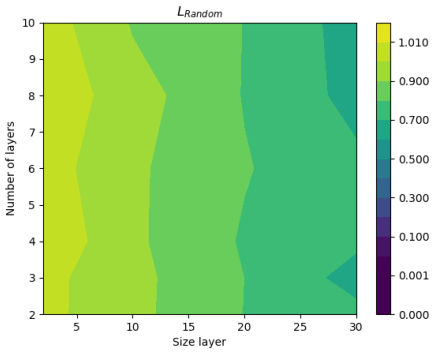

Fig. 5: Computation of the Lipschitz constant using the random exploration for different depths (y axis) and different layer size (x axis)

Figure 2 and 4 provide the best landscape of estimation of the Lipschitz constant: according to the color scale the methods appear to be finding accurately the value of 1 of the Lipschitz constant which makes them very suitable for estimating the Lipschitz smoothness. One very interesting comparison to make is between Figure 3 and 4: they are both implementations of SeqLip however they are resorting to different solvers. The former solver is greedy - and therefore sub-optimal for dealing with an NP hard problem - and the latter is non deterministic - and therefore finds the optimal solution provided it has iterated sufficiently. This shows, although the theory proves that SeqLip provides ultimately an upper bound, that the result is rarely the correct value if the problem is not solved suitably. On top of that it showcases that the value given by this algorithm should be treated carefully and can't be compared reliably to a lower bound. Figure 5 presents a methods for computing a lower bound for the Lipschitz constant: this is a random search that seeks the maximum of the differential quotient. The random search depends only on the dimension of the sampling space (that is taken equal to the layer size in our specific case) and thus why the quality of the approximation only depends on the abscissa.

Highlighting computational limits

One other major use of these test cases is the possibility to test the limits of these algorithms and their behaviors alongside validity boundaries: are their predictions still valid next to their validity limits or are the algorithms already giving a degraded value of the Lipschitz constant? To that extend we probe the four previous algorithms on a much larger range of values of depth and size layer that should trespass most of the algorithm range of validity. This time we present in hatches the zones that were not solved by the algorithms. This can be considered as zooming out from the previous section although the object of this study is more the validity domain rather than the accuracy of the predictions.

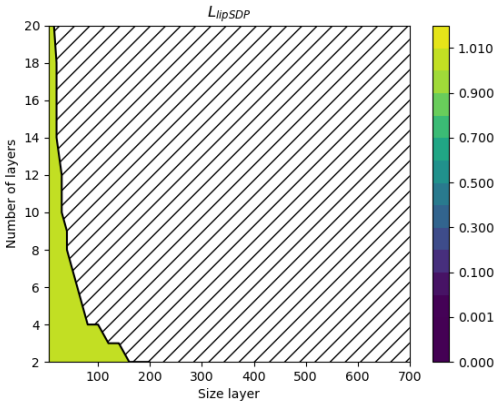

Fig. 6: Computation of the Lipschitz constant using the LipSDP algorithm for larger depths (y axis) and larger layer size (x axis).

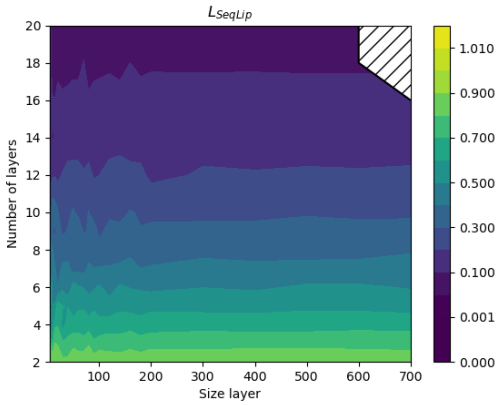

Fig. 7: Computation of the Lipschitz constant using the seqLip algorithm for larger depths (y axis) and larger layer size (x axis) with numpy optimizer.

On Figure 6 we see that the algorithm gives a very good approximation of the Lipschitz constant but on a small region: as depicted in the original work from [7] this "network" version of LipSDP suffers greatly from the dimension, we clearly see that this is the case here. Figure 7 and 8 present two different optimizers on the same maximization problem. Figure 8 is clearly dependent on the dimension of the parameters as it is displaying similar quality of estimation along the iso-dimension curves ($y \times x^2 = \text{constant}$) whereas Figure 6 displays similar performance along same depths. On the other hand the random approach (Figure 9) seems to suffer much less from the dimension than the two previous algorithms making it a more viable choice at high parameter dimension as it only depends on the dimension of the input.

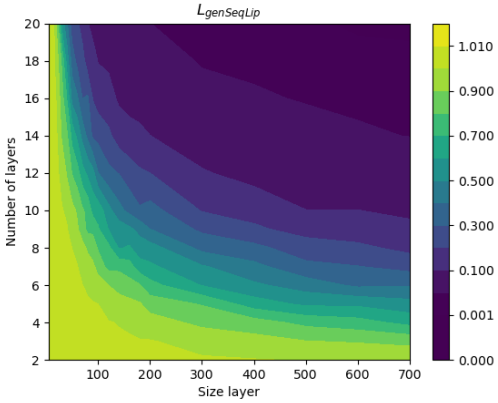

Fig. 8: Computation of the Lipschitz constant using the seqLip algorithm for larger depths (y axis) and larger layer size (x axis) with genetic optimizer.

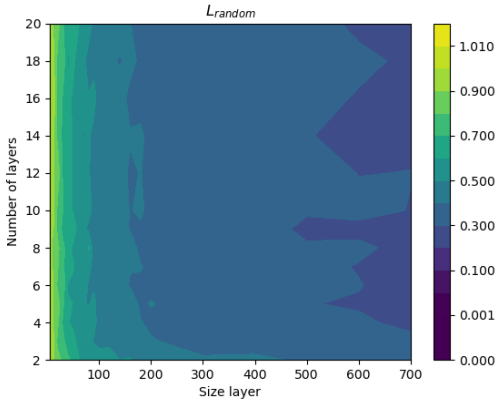

Fig. 9: Computation of the Lipschitz constant using the random exploration for larger depths (y axis) and larger layer size (x axis).

7 Conclusions and future work

This paper introduced a new paradigm for evaluating and understanding algorithms assessing the Lipschitz constant of neural networks. This work fills a gap in the evaluation of such algorithms, as none of the methods proposed to date have been evaluated on models for which the constant to be estimated is known. In addition, our experiments allow a better analysis of the estimates given by the algorithms. It helps to highlight the specificity of some approaches, such as the dependency to the input dimension for some of the algorithms or the choice of the optimizer for others. Overall it allows to improve and validate the parameters chosen for a given estimation algorithm: our results show that, for computing lower bounds, random approaches are quite easily plagued by the dimension if the sampling does not increase accordingly.

In future work, we plan to attempt at softening the constraints that we enforced on the neural networks, without sacrificing our knowledge of the Lipschitz constant. Extensions to convolution layers seems the logical path but it has its own challenges as the Lipschitz constant depends not only on the parametrization of the convolution layer but also on the input size, which can differ from one sample to another. One could also consider, in the same fashion that we did in this contribution, slightly constrained special cases where it would be easier to actually know the Lipschitz constant of the convolutions beforehand.

8 Declaration of funding

This work was partially supported by the ANRT and the ANR-19-CHIA-0017-01-DEEP-VISION project.

References

1. Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
2. Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.
3. Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. In David Helmbold and Bob Williamson, editors, *Computational Learning Theory*, volume 2111, pages 224–240. Springer Berlin Heidelberg. Series Title: Lecture Notes in Computer Science.
4. Louis Béthune, Alberto González-Sanz, Franck Mamalet, and Mathieu Serrurier. The Many Faces of 1-Lipschitz Neural Networks. *CoRR*, abs/2104.05097, 2021. arXiv: 2104.05097.
5. Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 3839–3848, 2018.
6. Todd Huster, Cho-Yu Jason Chiang, and Ritu Chadha. Limitations of the Lipschitz Constant as a Defense Against Adversarial Examples. In Carlos Alzate, Anna Monreale, Haytham Assem, Albert Bifet, Teodora Sandra Buda, Bora Caglayan, Brett Drury, Eva García-Martín, Ricard Gavaldà, Stefan Kramer, Niklas Lavesson, Michael Madden, Ian M. Molloy, Maria-Irina Nicolae, and Mathieu Sinn, editors, *ECML PKDD 2018 Workshops - Nemesis 2018, UrbReas 2018, SoGood 2018, IWAISe 2018, and Green Data Mining 2018, Dublin, Ireland, September 10-14, 2018, Proceedings*, volume 11329 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2018.
7. Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George J. Pappas. Efficient and Accurate Estimation of Lipschitz Constants for Deep Neural Networks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 11423–11434, 2019.
8. Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10900–10910, 2018.
9. Tong Chen, Jean-Bernard Lasserre, Victor Magron, and Edouard Pauwels. Semialgebraic Optimization for Lipschitz Constants of ReLU Networks. *arXiv:2002.03657 [cs, math]*, October 2020. arXiv: 2002.03657.
10. Mathieu Serrurier, Franck Mamalet, Alberto González-Sanz, Thibaut Boissin, Jean-Michel Loubes, and Eustasio del Barrio. Achieving Robustness in Classification Using Optimal Transport With Hinge Regularization. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 505–514. Computer Vision Foundation / IEEE, 2021.

11. Moustapha Cissé, Piotr Bojanowski, Edouard Grave, Yann N. Dauphin, and Nicolas Usunier. Parseval Networks: Improving Robustness to Adversarial Examples. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 854–863. PMLR, 2017.
12. Jérôme Bolte and Edouard Pauwels. Conservative set valued fields, automatic differentiation, stochastic gradient methods and deep learning. *Math. Program.*, 188(1):19–51, 2021.
13. William Piat, Jalal Fadili, Frédéric Jurie, and Sébastien da Veiga. Regularized Robust Optimization with Application to Robust Learning. working paper or preprint, June 2022.
14. Francesco Mezzadri. How to generate random matrices from the classical compact groups. *arXiv:math-ph/0609050*, February 2007. arXiv: math-ph/0609050.