A Quantitative Geometric Approach to Neural-Network Smoothness

Zi Wang
University of Wisconsin-Madison
zw@cs.wisc.edu

Gautam Prakriya
The Chinese University of Hong Kong
gautamprakriya@gmail.com

Somesh Jha
University of Wisconsin-Madison
jha@cs.wisc.edu

Abstract

Fast and precise Lipschitz constant estimation of neural networks is an important task for deep learning. Researchers have recently found an intrinsic trade-off between the accuracy and smoothness of neural networks, so training a network with a loose Lipschitz constant estimation imposes a strong regularization, and can hurt the model accuracy significantly. In this work, we provide a unified theoretical framework, a quantitative geometric approach, to address the Lipschitz constant estimation. By adopting this framework, we can immediately obtain several theoretical results, including the computational hardness of Lipschitz constant estimation and its approximability. We implement the algorithms induced from this quantitative geometric approach, which are based on semidefinite programming (SDP).[1] Our empirical evaluation demonstrates that they are more scalable and precise than existing tools on Lipschitz constant estimation for ℓ_∞-perturbations. Furthermore, we also show their intricate relations with other recent SDP-based techniques, both theoretically and empirically. We believe that this unified quantitative geometric perspective can bring new insights and theoretical tools to the investigation of neural-network smoothness and robustness.

1 Introduction

The past decade has witnessed the unprecedented success of deep learning in many machine learning tasks (Krizhevsky et al., 2017; Mikolov et al., 2013). Despite the growing popularity of deep learning, researchers have also found that neural networks are very vulnerable to adversarial attacks (Szegedy et al., 2014; Goodfellow et al., 2015; Papernot et al., 2016). As a result, it is important to train neural networks that are robust against those attacks (Madry et al., 2018). In recent years, the deep learning community starts to focus on certifiably robust neural networks (Albarghouthi, 2021; Hein and Andriushchenko, 2017; Katz et al., 2017; Cohen et al., 2019; Raghunathan et al., 2018; Wang et al., 2022; Leino et al., 2021). One way to achieve certified robustness is to estimate the smoothness of neural networks, where the smoothness is measured by the Lipschitz constant of the neural network. Recent works have found that to achieve both high accuracy and low Lipschitzness, the network has to significantly increase the model capacity (Bubeck and Sellke, 2021). This implies that there is an intrinsic tension between the accuracy and smoothness of neural networks.

Commonly considered adversarial attacks are the ℓ_∞ and ℓ_2-perturbations in the input space. Leino et al. (2021); Cohen et al. (2019) have successfully trained networks with low ℓ_2-Lipschitz constant,

[1] Our code is available at https://github.com/z1w/GeoLIP.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

and Huang et al. (2021) trained networks with low local Lipschitzness for ℓ_∞-perturbations. There are few well-established techniques to train neural networks with low global Lipschitzness for ℓ_∞-perturbations. The techniques for the ℓ_2-perturbation do not easily transfer to the ℓ_∞-case (Leino et al., 2021). One critical step is to measure the Lipschitz constant more precisely and efficiently. Jordan and Dimakis (2020) showed that for ReLU networks, it is NP-hard to approximate the Lipschitz constant for ℓ_∞-perturbations within a constant factor , and proposed an exponential-time algorithm to compute the exact Lipschitz constant. However, researchers are interested in more scalable approaches to certify and train networks. In this work, we consider the *Formal Global Lipschitz* constant (FGL) (See Equation (3)), which is roughly the maximum of the gradient operator norm, assuming all activation patterns on hidden layers are independent and possible. FGL is an upper bound of the exact Lipschitz constant and has been used in Raghunathan et al. (2018); Fazlyab et al. (2019); Latorre et al. (2020).

We address the Lipschitz constant estimation from the quantitative geometric perspective. Quantitative geometry aims to understand geometric structures via quantitative parameters, and has connections with many mathematical disciplines, such as functional analysis and combinatorics (Naor, 2013). In computer science, quantitative geometry plays a central role in understanding the approximability of discrete optimization problems. We approach those hard discrete optimization problems by considering the efficiently solvable continuous counterparts, and analyze the precision loss due to relaxation, which is often the SDP relaxation (Goemans and Williamson, 1995; Nesterov, 1998; Alon and Naor, 2004). The natural SDP relaxations for the intractable problems usually induce the optimal known polynomial time algorithms (Bhattiprolu et al., 2022). By adopting the quantitative geometric approach, we can immediately understand the computational hardness and approximability of FGL estimations. Our algorithms on two-layer networks are the natural SDP relaxations from the quantitative geometric perspective.

Latorre et al. (2020) employed polynomial optimization methods on the FGL estimation for ℓ_∞-perturbations. Polynomial optimization is a very general framework, and many problems can be cast in this framework (Motzkin and Straus, 1965; Goemans and Williamson, 1995). Therefore, we argue that this is not a precise characterization of the FGL-estimation problem. On the other hand, there are also several SDP-based techniques for FGL estimations. Raghunathan et al. (2018) proposed an SDP algorithm to estimate the FGL for ℓ_∞-perturbations of two layer networks, and Fazlyab et al. (2019) devised an SDP algorithm to estimate the ℓ_2-FGL. We will demonstrate the intricate relationships between our algorithms and these existing SDPs on two-layer networks.

Several empirical studies have found that techniques on one ℓ_p-perturbations often do not transfer to another ones, even though the authors claim that *in theory* these techniques should transfer (Fazlyab et al., 2019; Leino et al., 2021). This in-theory claim usually comes from a *qualitative* perspective. In finite-dimensional space, one can always bound one ℓ_p-norm from another one, so techniques for one ℓ_p-perturbations can also provide another bound for a different ℓ_p-perturbations. However, this bound is loose and in practice not useful (Latorre et al., 2020). Instead, we believe that when transferring techniques from one norm to another one, we should consider the quantitative geometric principle: we should separate the geometry-dependent component from the geometry-independent one in those techniques, and modify the geometry-dependent component accordingly for a different normed space. As we will demonstrate, our whole work is guided by this principle. We hope that our unified quantitative geometric framework can bring insights to the empirical hard-to-transfer observations, and new tools to address these issues.

Contributions. To summarize, we have made the following contributions:

1. We provide a unified theoretical framework to address FGL estimations, which immediately yields the computational hardness and SDP-based approximation algorithms on two-layer networks (Section 3).
2. We demonstrate the relations between our algorithms and other SDP-based tools, which in return inspires us to design the algorithms for multi-layer networks. This provides more insightful and compositional interpretations of existing works, and makes them easier-to-generalize (Sections 4 and 5).
3. We implement the algorithms and name the tool GeoLIP. We empirically validate our theoretical claims, and compare GeoLIP with existing methods to estimate FGL for ℓ_∞-perturbations. The result shows that GeoLIP provides a tighter bound (20%-60% improvements) than

existing tools on small networks, and much better results than the naive matrix-norm-product method for deep networks, which existing tools cannot handle (Section 6).

2 Preliminaries

Notation. Let $[n] = \{1, \ldots, n\}$. For two functions f and g, $f \circ g(x) = f(g(x))$. A 0-1 cube is $\{0,1\}^n$, and a norm-1 cube is $\{-1,1\}^n$ for some integer $n > 0$. $\mathbb{R}_+ = [0, \infty)$. For any vector $v \in \mathbb{R}^n$, $\text{diag}(v)$ is an $n \times n$ diagonal matrix, with diagonal values v. Let $e_n = (1, \ldots, 1) \in \mathbb{R}^n$ be an n-dimensional vector of all 1's; and $I_n = \text{diag}(e_n)$, the identity matrix. Let $||v||_p$ denote the ℓ_p norm of v. We use q to denote the Hölder conjugate of p as a convention, i.e., $\frac{1}{p} + \frac{1}{q} = 1$. If v is an operator in the ℓ_p-space, the operator norm of v is then $||v||_q$. Throughout the paper, we consider the ℓ_p-norm of the input's perturbation, and therefore, the ℓ_q-norm of the gradient, which acts as an operator on the perturbation. A square matrix $X \succeq 0$ means that X is positive semidefinite (PSD). Let $\text{tr}(X)$ be the trace of a square matrix X. If $a, b \in \mathbb{R}^n$, let $\langle a, b \rangle$ be the inner product of a and b.

Lipschitz function. Given two metric spaces (X, d_X) and (Y, d_Y), a function $f : X \to Y$ is *Lipschitz* continuous if there exists $K > 0$ such that for all $x_1, x_2 \in X$,

$$d_Y(f(x_2), f(x_1)) \leq K d_X(x_2, x_1). \tag{1}$$

The smallest such K satisfying Equation (1), denoted by K_f, is called the Lipschitz constant of f. For neural networks, X is in general $\mathbb{R}^m$ equipped with the ℓ_p-norm. We will only consider the case when $Y = \mathbb{R}$. In actual applications such as a classification task, a neural network has multiple outputs. The prediction is the class with the maximum score. One can then use the margin between each pair of class predictions and its Lipschitz constant to certify the robustness of a given prediction (Raghunathan et al., 2018; Leino et al., 2021). From Rademacher's theorem, if f is Lipschitz continuous, then f is is almost everywhere differentiable, and $K_f = \sup_x ||\nabla f(x)||_q$.

Neural network as function. A neural network $f : \mathbb{R}^m \to \mathbb{R}$ is characterized by:

$$f_1(x) = W^1 x + b_1; \; f_i(x) = W^i \sigma(f_{i-1}(x)) + b_i, i = 2, \ldots, d.$$

where $W^i \in \mathbb{R}^{n_{i+1} \times n_i}$ is the weight matrix between the layers, $n_1 = m$, d is the depth of the neural network, σ denotes an activation function, $b_i \in \mathbb{R}^{n_{i+1}}$ is the bias term, and $f = f_d \circ \cdots \circ f_1$. Because we only consider the $\mathbb{R}$ as the codomain of f, $W^d = u \in \mathbb{R}^{1 \times n_d}$ is a vector. From chain rule, the gradient of this function is

$$\nabla f(x) = (W^1)^T [\text{diag}(\sigma'(f_1(x)))(W^2)^T \cdots \text{diag}(\sigma'(f_{d-1}(x)))(W^d)^T]. \tag{2}$$

Common activation functions, including ReLU (Nair and Hinton, 2010), sigmoid functions, and ELU (Clevert et al., 2016) are almost everywhere differentiable. As a result, we are interested in the supremum operator norm of Equation (2).

However, checking all possible inputs x is infeasible, and common activation functions have bounded derivative values, say $[a, b]$. We are then interested in the following value instead:

$$\max_{v^i \in [a,b]^{n_i}} ||(W^1)^T \cdot \text{diag}(v^2) \cdot \cdots \cdot \text{diag}(v^d)(W^d)^T||_q, \tag{3}$$

where n_i is the dimension of each $\text{diag}(v^i)$. We call this value the *formal* global Lipschitz constant (FGL) because we treat all activation functions independent but in reality not all activation patterns are feasible. Therefore, this is an upper bound of the *true* global Lipschitz constant of the neural network. However, it is the value studied in most global Lipschitzness literature (Scaman and Virmaux, 2018; Fazlyab et al., 2019; Latorre et al., 2020), and also turns out useful in certifying the robustness of neural networks (Raghunathan et al., 2018; Leino et al., 2021; Pauli et al., 2022). We use ℓ_p-FGL to denote the FGL for ℓ_p-perturbations.

In this paper, we focus on the ℓ_p-perturbation on the input, where $p = \infty$ or $p = 2$. Because q is the Hölder conjugate of p, we are interested in the value of Equation (3), when $q = 1$ (for $p = \infty$), and $q = 2$ (for $p = 2$). Notice that in the ReLU-network case, $[a, b]$ is $[0, 1]$. We will use ReLU-networks as the illustration for the rest of the paper because of the popularity of ReLU in practice and the easy presentation of the 0-1-cube. However, the algorithms presented in this work can be adapted with minor adjustments to other common activation functions.

Remark 2.1. FGL considers all possible activation patterns on the hidden layers, while some of the activation patterns might be unachievable in reality. Therefore, FGL is an upper bound of the true Lipschitz constant. Notice that the activation pattern induced from an input is also decided by the bias term. Therefore, to find the true Lipschitz constant, one has to incorporate the information from the bias term.

3 Two-layer neural networks

In this section, we consider the two-layer neural network case. We reduce the FGL estimation to the matrix mixed-norm problem. This immediately yields the computational complexity and approximation algorithms for FGL estimations. In Appendix A.1, we show that we can consider $\{0, 1\}$ instead of $[0, 1]$ in Equation (3) for two-layer networks.

Problem description. For a two-layer network where $W^1 = W \in \mathbb{R}^{n\times m}$ and $W^d = u \in \mathbb{R}^{1\times n}$, its FGL (as in Equation (3)) is $\max_{y\in\{0,1\}^n} ||W^T \text{diag}(y)u^T||_q$, where we use y to denote v^1 in this case. If we expand the matrix multiplication, it is easy to check that this equals to $\max_{y\in\{0,1\}^n} ||W^T \text{diag}(u)y||_q$. Let $A = W^T \text{diag}(u)$, then the ℓ_p-FGL is

$$\max_{y\in\{0,1\}^n} ||Ay||_q. \tag{4}$$

3.1 ℓ_∞-FGL estimation

We consider a natural SDP relaxation to Equation (4) when $q = 1$, and analyze the result using the celebrated *Grothendieck Inequality*, which is a fundamental tool in functional analysis.

Mixed-norm problem. The $\infty \to 1$ mixed-norm of a matrix is defined as

$$||A||_{\infty\to 1} = \max_{||x||_\infty=1} ||Ax||_1.$$

The mixed-norm problem appears similar to Equation (4) when $q = 1$, except for that instead of a norm-1-cube, the cube in Equation (4) is a 0-1-cube. Alon and Naor (2004) showed that it is NP-hard, specifically MAXSNP-hard, to compute the $\infty \to 1$ mixed-norm of a matrix A, via a reduction to the graph Max-Cut problem. Moreover, Alon and Naor (2004) constructed a natural SDP relaxation for the mixed-norm problem:

$$\begin{aligned} &\max \text{tr}(BX) \\ &s.t.\ X \succeq 0, X_{ii} = 1, i \in [n+m], \end{aligned} \tag{5}$$

where A is a submatrix of B. We provide the detailed derivation of this relaxation in Appendix A.3. In fact, this relaxation admits a constant approximation factor. Grothendieck (1956) developed the local theory of Banach spaces, and showed that there exists an absolute value K_G such that

Theorem 3.1. *For any $m, n \geq 1$, $A \in \mathbb{R}^{n\times m}$, and any Hilbert space H, the following holds:*

$$\max_{u_i,v_j\in B(H)} \sum_{i,j} A_{ij}\langle u_i, v_j\rangle_H \leq K_G ||A||_{\infty\to 1},$$

where $B(H)$ denotes the unit ball of the Hilbert space.

The precise value of K_G is still an outstanding open problem, and it is known that $K_G < 1.783$ (Krivine, 1979; Braverman et al., 2011). The approximation factor of the SDP relaxation in Equation (5) is K_G. Similar to the mixed-norm problem, we show that the ℓ_∞-FGL estimation is MAXSNP-hard and provide an SDP relaxation, which also admits the K_G-approximation ratio. We provide a detailed explanation on why K_G is the approximation ratio and how we can view the SDP relaxation as a geometric transformation in Appendix A.4.

Theorem 3.2. *ℓ_∞-FGL estimation is* MAXSNP*-hard.*

From 0-1 cube to norm-1 cube. If we can transform the 0-1 cube in Equation (4) to a norm-1 cube, and formulate an equivalent optimization problem, then one can apply the SDP program in Equation (5) to compute an upper bound of the FGL. Indeed, we provide a cube rescaling technique,

and it allows us to construct the SDP for the ℓ_∞-FGL estimation. We provide the full detail of this technique in Appendix A.5, and the result SDP for the ℓ_∞-FGL estimation is

$$\begin{aligned} \max\ & \frac{1}{2}\text{tr}(BX) \\ s.t.\ & X \succeq 0, X_{ii} = 1, i \in [n+m+1], \end{aligned} \tag{6}$$

where B is a $(n+1+m) \times (n+1+m)$ matrix, and $B = \begin{pmatrix} 0 & 0 & 0 \\ A & Ae_n & 0 \end{pmatrix}$. As a result, we have:

Theorem 3.3. *There exists a polynomial-time approximation algorithm to estimate the ℓ_∞-FGL of two-layer neural networks, moreover, the approximation ratio is K_G.*

3.2 ℓ_2-FGL estimation

Scaman and Virmaux (2018) showed that the ℓ_2-FGL estimation is NP-hard. If $q = 2$ in Equation (4), the objective becomes similar to the $\infty \to 2$ mixed-norm problem. This is a quadratic optimization problem with a PSD weight matrix over a cube, and can be viewed as a generalization of the graph Max-Cut problem. In the quadratic optimization formulation of Max-Cut, the weight matrix is the Laplacian of the graph, a special PSD matrix (Goemans and Williamson, 1995). Nesterov (1998) generalized Goemans-Williamson's technique and analyzed the case when the weight matrix is PSD, showing that the natural SDP relaxation in this case has a $\frac{\pi}{2}$-approximation ratio. This provides a $\sqrt{\frac{\pi}{2}}$-approximation algorithm for the ℓ_2-FGL estimation. The approximation ratio comes from a similar inequality to the one in Theorem 3.1, known as the *Little Grothendieck Inequality*. The SDP for ℓ_2-FGL estimation is:

$$\begin{aligned} \max\ & \frac{1}{2}\sqrt{\text{tr}(\begin{pmatrix} A^TA & A^TAe_n \\ e_n^TA^TA & e_n^TA^TAe_n \end{pmatrix} X)} \\ s.t.\ & X \succeq 0, X_{ii} = 1, i \in [n+1]. \end{aligned} \tag{7}$$

The full derivation is provided in Appendix A.8, and we have the following theorem:

Theorem 3.4. *There exists a polynomial-time approximation algorithm to estimate the ℓ_2-FGL of two-layer neural networks with an approximation factor $\sqrt{\frac{\pi}{2}}$.*

Remark 3.5. As we have discussed, for two-layer networks, the ℓ_p-FGL estimation is essentially the $\infty \to q$ mixed-norm problem. Indeed the mixed-norm problem is an outstanding topic in theoretical computer science. As discussed in Bhattiprolu et al. (2018), the $\infty \to q$ mixed norm problem has constant approximation algorithms if $q \leq 2$, and is hard to approximate within almost polynomial factors when $q > 2$. Because when $q > 2$, its Hölder conjugate $p < 2$. This implies that for two-layer networks, the FGL estimation can be much harder for ℓ_p-perturbations when $p < 2$.

Briët et al. (2017) showed that it is NP-hard to approximate the $\infty \to 2$ mixed-norm problem better than $\sqrt{\frac{\pi}{2}}$. Raghavendra and Steurer (2009) proved that assuming the unique games conjecture (Khot, 2002), it is NP-hard to approximate the $\infty \to 1$ mixed-norm problem better than K_G. These optimal approximation ratios match our SDP relaxations for FGL estimations accordingly.

4 Relations to existing SDP works

Before introducing our approach for multi-layer networks, we first examine some existing SDP works on FGL estimations, and discuss their relationships with our natural SDP relaxations in Section 3.

ℓ_∞-FGL estimation. Raghunathan et al. (2018) formulated an SDP that only works for two-layer networks. Theirs is essentially the same as ours in Equation (6) (See the detailed comparison in Appendix A.10). However, we provide a rigorous derivation and simpler formulation, and also a sound theoretical analysis of the bound, which illustrate more insights to this problem. Raghunathan et al. (2018) treated the SDP relaxation as a heuristic to a hard quadratic programming problem. We prove that this relaxation is not only a heuristic, but in fact induces an approximation algorithm with a tight bound.

ℓ_2-FGL estimation. Fazlyab et al. (2019) proposed LipSDP, another SDP-based algorithm for the ℓ_2-FGL estimation problem. Fazlyab et al. (2019) provided several variants of LipSDP to balance the

precision and scalability. Pauli et al. (2022) demonstrated that the most precise version of LipSDP, *LipSDP-Network*, fails to produce an upper bound for ℓ_2-FGL. In this paper, all the references of LipSDP are to *LipSDP-Neuron*, the less precise version. Surprisingly, even though the approach in LipSDP appears quite different from Equation (7), we show that LipSDP is dual of Equation (7) to estimate the ℓ_2-FGL on two-layer networks. LipSDP for two-layer networks is:

$$\min_{\zeta,\lambda}\Big\{\sqrt{\zeta} : \begin{pmatrix} -2abW^TWT - \zeta I_m & (a+b)W^TT \\ (a+b)TW & -2T + u^Tu \end{pmatrix} \preceq 0, \lambda_i \geq 0\Big\},$$

where $T = \text{diag}(\lambda)$ for $\lambda \in \mathbb{R}^n_+$; a and b are the lower and upper bounds of the activation's derivative.

We will construct a new quadratic program, which we show is equivalent to Equation (4) when $q = 2$, and LipSDP is its dual SDP relaxation.

Let the input of the i-th activation node on $\text{diag}(y)$ be y_i, and w_i be the row vector of W. Hence, $y_i = w_i x$. Let $\Delta x \in \mathbb{R}^m$ be a perturbation on x, so $\Delta y_i = w_i \Delta x$. Let $\Delta\sigma(y) \in \mathbb{R}^n$ denote the induced perturbation on $\text{diag}(y)$. The ***constraint*** from the activation function is $\frac{\Delta\sigma(y)_i}{\Delta y_i} \in [a, b]$, in other words, $\frac{\Delta\sigma(y)_i}{w_i \Delta x} \in [a, b]$. One can write the range constraint as

$$(\Delta\sigma(y)_i - a \cdot w_i\Delta x)(\Delta\sigma(y)_i - b \cdot w_i\Delta x) \leq 0.$$

This can be written in the quadratic form:

$$\begin{pmatrix} w_i\Delta x \\ \Delta\sigma(y)_i \end{pmatrix}^T \begin{pmatrix} -2ab & a+b \\ a+b & -2 \end{pmatrix} \begin{pmatrix} w_i\Delta x \\ \Delta\sigma(y)_i \end{pmatrix} \geq 0, \quad \forall i \in [n]. \tag{8}$$

Since for the two layer network, $f(x) = u\sigma(y)$, then $\Delta f(x) = u\Delta\sigma(y)$. The ***objective*** for ℓ_2-FGL estimation is $\max_{\Delta x, \Delta\sigma(y)} \sqrt{\frac{(u\Delta\sigma(y))^2}{(\Delta x)^2}}$. The equivalence between this program and Equation (4) when $q = 2$ is presented in Appendix A.11.

Remark 4.1. Another interpretation for the quadratic program is that we want to quantify how the output changes given a data-independent input change, i.e., Δx. In other words, we want to analyze the effect of Δx propagating from the input to the output, with symbolic values rather than actual inputs. The idea is similar to symbolic execution from program analysis (Baldoni et al., 2018).

Duality to LipSDP. Now we will show that LipSDP is the dual SDP to the program formulated above. The dual SDP derivation is of similar form in Ben-Tal and Nemirovski (2001, Ch.4.3.1). Let us introduce a variable ζ such that $\zeta - \frac{(u\Delta\sigma(y))^2}{(\Delta x)^2} \geq 0$. In other words,

$$\zeta(\Delta x)^2 - (u\Delta\sigma(y))^2 \geq 0. \tag{9}$$

For each constraint in Equation (8), let us introduce a dual variable $\lambda_i \geq 0$. Multiply each constraint with λ_i, then $\begin{pmatrix} \Delta x \\ \Delta\sigma(y)_i \end{pmatrix}^T \begin{pmatrix} -2ab\lambda_i w_i^T w_i & (a+b)\lambda_i w_i^T \\ (a+b)\lambda_i w_i & -2\lambda_i \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta\sigma(y)_i \end{pmatrix} \geq 0, \quad \forall i \in [n]$.

Sum all of them, then we have $\begin{pmatrix} \Delta x \\ \Delta\sigma(y) \end{pmatrix}^T \begin{pmatrix} -2abW^TWT & (a+b)W^TT \\ (a+b)TW & -2T \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta\sigma(y) \end{pmatrix} \geq 0$, where $T = \text{diag}(\lambda)$ is the $n \times n$ diagonal matrix of dual variables $\lambda_1, \ldots, \lambda_n$.

Equation (9) can be rewritten as: $\begin{pmatrix} \Delta x \\ \Delta\sigma(y) \end{pmatrix}^T \begin{pmatrix} \zeta I_m & 0 \\ 0 & -u^Tu \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta\sigma(y) \end{pmatrix} \geq 0$. As a result, the dual program for the new optimization program is

$$\min_{\zeta,\lambda}\Big\{\sqrt{\zeta} : \begin{pmatrix} -2abW^TWT - \zeta I_m & (a+b)W^TT \\ (a+b)TW & -2T + u^Tu \end{pmatrix} \preceq 0, \lambda_i \geq 0\Big\}.$$

Remark 4.2. In Remark 3.5, we mention that $\sqrt{\frac{\pi}{2}}$ is the optimal approximation ratio for the $\infty \to 2$ mixed-norm problem, which matches the approximation ratio in Theorem 3.4. Hence, improving the natural SDP relaxation in Equation (7) can be very hard. The duality provides another evidence of LipSDP-Neuron's correctness, and hints that LipSDP-Network, the improved variant, may be wrong.

5 ℓ_∞-FGL estimation for multi-layer networks

For a multi-layer neural network, the formal gradient becomes a high-degree polynomial, and its ℓ_q-norm estimation becomes a high-degree polynomial optimization problem over a cube, which is in general a hard problem (Lasserre, 2015). We provide a discussion of the polynomial optimization approach of FGL estimation in Appendix B. Here we provide an SDP dual program of the ℓ_∞-FGL estimation inspired by the dual SDP approach in Section 4. The difference is that now we consider ℓ_∞-perturbations to the input space instead of ℓ_2. Hence, the objective becomes

$$\max_{\Delta x, \Delta\sigma(y)} \frac{|u\Delta\sigma(y)|}{||\Delta x||_\infty}.$$

If we add an extra constraint $||\Delta x||_\infty = 1$, the above objective becomes

$$\max_{\Delta x, \Delta\sigma(y)} \frac{1}{2}(u\Delta\sigma(y) + u\Delta\sigma(y)). \tag{10}$$

The constraints are

$$\begin{pmatrix} w_i\Delta x \\ \Delta\sigma(y)_i \end{pmatrix}^T \begin{pmatrix} 2ab & -(a+b) \\ -(a+b) & 2 \end{pmatrix} \begin{pmatrix} w_i\Delta x \\ \Delta\sigma(y)_i \end{pmatrix} \leq 0,\ \Delta(x)_j^2 \leq 1,\ \forall i \in [n], j \in [m].$$

We can write $\Delta(x)_j^2 \leq 1$ as $\begin{pmatrix} 1 \\ \Delta x_j \end{pmatrix}^T \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ \Delta x_j \end{pmatrix} \geq 0$.

Now let us introduce $n+m$ non-negative dual variables (τ, λ), where $\tau \in \mathbb{R}_+^m$ and $\lambda \in \mathbb{R}_+^n$. If we multiply each dual variable with the constraint and add all the constraints together, we will have

$$\begin{pmatrix} 1 \\ \Delta x \\ \Delta\sigma(y) \end{pmatrix}^T \begin{pmatrix} \sum_{j=1}^m \tau_j & 0 & 0 \\ 0 & -2abW^TWT_2 - T_1 & (a+b)W^TT_2 \\ 0 & (a+b)T_2W & -2T_2 \end{pmatrix} \begin{pmatrix} 1 \\ \Delta x \\ \Delta\sigma(y) \end{pmatrix} \geq 0,$$

where $T_1 = \text{diag}(\tau)$ and $T_2 = \text{diag}(\lambda)$. As a result, we can incorporate the objective Equation (10) and obtain the dual SDP for the ℓ_∞-FGL estimation:

$$\min_{\zeta,\lambda,\tau} \Big\{ \frac{\zeta}{2} : \begin{pmatrix} \sum_{j=1}^m \tau_j - \zeta & 0 & u \\ 0 & -2abW^TWT_2 - T_1 & (a+b)W^TT_2 \\ u^T & (a+b)T_2W & -2T_2 \end{pmatrix} \preceq 0, \lambda_i, \tau_j \geq 0 \Big\}. \tag{11}$$

Remark 5.1. The SDP programs in Section 3 are strictly feasible because the identity matrix is a positive definite solution. Hence, Slater's condition is satisfied and strong duality holds.

Multi-layer extension. We can simply extend the dual program to multiple-layer networks. We first vectorize all the units in the input layer and hidden layers, and then constrain them using layer-wise inequalities to formulate an optimization problem. Let us consider a general d-layer multi-layer network, where $W^i \in \mathbb{R}^{n_{i+1} \times n_i}$ for $i \in [d-1]$, and $W^d = u \in \mathbb{R}^{1 \times n_d}$. Let Δx denote the perturbation on the input layer, Δz^i be the perturbation on the i-th hidden layer, and w_j^i be the j-th row vector of W^i. The only difference between two layer networks and multi-layer networks is that we have the additional constraints:

$$\begin{pmatrix} \Delta z^i \\ \Delta z_j^{i+1} \end{pmatrix}^T \begin{pmatrix} -2ab(w_j^{i+1})^T w_j^{i+1} & (a+b)(w_j^{i+1})^T \\ (a+b)w_j^{i+1} & -2 \end{pmatrix} \begin{pmatrix} \Delta z^i \\ \Delta z_j^{i+1} \end{pmatrix} \geq 0.$$

Let $\Lambda_i \in \mathbb{R}_+^{n_i}$ and $T_i = \text{diag}(\Lambda_i)$ for $i \in [d]$. Following the similar SDP dual approach, we can add all the constraints together and formulate the following SDP program:

$$\min_{\zeta,\Lambda_i} \Big\{ \frac{\zeta}{2} : (L+N) \preceq 0, i \in [d] \Big\}, \tag{12}$$

where

$$L = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & -2ab(W^1)^TW^1T_2 & (a+b)(W^1)^TT_2 & \dots & 0 & 0 \\ 0 & (a+b)T_2W^1 & -2T_2 - 2ab(W^2)^TW^2T_3 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & (a+b)T_dW^{d-1} & -2T_d \end{pmatrix}, \tag{13}$$

Table 1: ℓ_∞-FGL estimation of various methods: DGeoLIP and NGeoLIP induce the same values on two layer networks. DGeoLIP always produces tighter estimations than LiPopt and MP do.

Network	DGeoLIP	NGeoLIP	LiPopt	MP	Sample	BruF
2-layer/16 units	185.18	185.18	259.44	578.54	175.24	175.24
2-layer/256 units	425.04	425.04	1011.65	2697.38	306.98	N/A
8-layer/64 units per layer	8327.2	——	N/A	$8.237 * 10^7$	1130.6	N/A

Table 2: Running time (in seconds) of various tools on ℓ_∞-FGL estimation: DGeoLIP and NGeoLIP are faster than LiPopt. Notice that the running time is implementation and solver-dependent.

Network	DGeoLIP	NGeoLIP	LiPopt	BruF
2-layer/16 units	28.1	22.3	1572	4.8
2-layer/256 units	976.0	70.9	2690	N/A
8-layer/64 units	329.5	——	N/A	N/A

$$N = \begin{pmatrix} \sum_{k=1}^{n_1} \Lambda_{1k} - \zeta & 0 & \dots & u \\ 0 & -T_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ u^T & 0 & \cdots & 0 \end{pmatrix}.$$

Remark 5.2. If we expand the matrix inequality derived from the compact neural-network representation in Fazlyab et al. (2019, Theorem 2), we will have exactly the same matrix for network constraints as L (Equation (13)) in the dual program formulation. In other words, we provide a compositional optimization interpretation to the compact neural-network representation in LipSDP. With this interpretation, one can extend the SDP to beyond feed-forward structures, such as skip connections (He et al., 2016). Notice that if we apply the similar reasoning to the multi-layer network ℓ_2-FGL estimation, we will obtain LipSDP-Neuron.

6 Evaluation and discussion

The primary goal of our work is to provide a theoretical framework, and also algorithms for ℓ_∞-FGL estimations on practically used networks. The ℓ_2-FGL can be computed using LipSDP. We have implemented the algorithms using MATLAB (MATLAB, 2021), the CVX toolbox (CVX Research, 2020) and MOSEK solver (ApS, 2019), and name the tool GeoLIP. To validate our theory and the applicability of our algorithms, we want to empirically answer the following research questions:

RQ1: Is GeoLIP better than existing methods in terms of precision and scalability?
RQ2: Are the dual SDP programs devised throughout the paper valid?

As we shall see, GeoLIP is indeed better than existing methods in terms of precision and scalability; and the dual SDP programs produce the same values as their natural-SDP-relaxation counterparts.

6.1 Experimental design

To answer **RQ1**, we will run GeoLIP and existing tools that measure the ℓ_∞-FGL on various feed-forward neural networks trained with the MNIST dataset (LeCun and Cortes, 2010). We will record the computed ℓ_∞-FGL to compare the precision, and the computation time to compare the scalability.

To answer **RQ2**, we will run the natural SDP relaxations for ℓ_p-FGL estimations proposed in Section 3, LipSDP for ℓ_2-FGL estimation, and the dual program Equation (11) for ℓ_∞-FGL on two-layer neural networks, and compare their computed FGLs.

Measurements. Our main baseline tool is ***LiPopt*** (Latorre et al., 2020), which is an ℓ_∞-FGL estimation tool.[2] Notice that LiPopt is based on the Python Gurobi solver (Gurobi Optimization, LLC, 2022), while we use the MATLAB CVX and MOSEK solver. LiPopt relies on a linear programming (LP) hierarchy for the polynomial optimization problem. We use LiPopt-k to denote the k-th degree of the LP hierarchy. ***BruF*** stands for an brute-force exhaustive enumeration of all possible activation patterns. This is the ground truth for FGL estimations. However, this is an exponential-time search, so we can only run it on networks with a few hidden units. ***Sample*** means that we randomly sample $200,000$ points in the input space and compute the gradient norm at those points. Notice that this is a lower bound of the true Lipschitz constant, and thus a lower bound of the FGL. ***MP*** stands for the weight-matrix-norm-product method. This is a naive upper bound of FGL. We use ***NGeoLIP*** to denote the natural SDP relaxations devised in Section 3, and ***DGeoLIP*** to denote the dual SDP Equation (12) for ℓ_∞-FGL estimation. Notice that NGeoLIP only applies to two-layer networks.

We use "—" in the result tables to denote that the experimental setting is not in the scope of the tool's application, and "N/A" to denote the computation takes too much time (> 20 hours).

Network setting. We run the experiments on fully-connected feed-forward neural networks, trained with the MNIST dataset for 10 epochs using the ADAM optimizer (Kingma and Ba, 2015). All the trained networks have accuracy greater than 92% on the test data. For two-layer networks, we use 8, 16, 64, 128, 256 hidden nodes. For multiple-layer networks, we consider 3, 7, 8-layer networks, and each hidden layer has 64 ReLU units. Because MNIST has 10 classes, we report the estimated FGL with respect to label 8 as in Latorre et al. (2020), and the average running time per class: we record the total computation time for all 10 classes from each tool, and report the average time per class.

6.2 Discussion

We present selected results in Tables 1 and 2, and related major discussions here. The full results, more experimental setup and additional discussions can be found in Appendix C.

RQ1. In the experiments of LiPopt, we only used LiPopt-2. In theory, if one can go higher in the LP hierarchy in LiPopt, the result becomes more precise. However, in the case of fully-connected neural networks, using degree-3 in LiPopt is already impractical. For example, on the simplest network that we used, i.e., the single-hidden-layer neural network with 8 hidden units, using LiPopt-3, one ℓ_∞-FGL computation needs at least 200 hours projected by LiPopt. As a result, for all the LiPopt-related experiments, we were only able to run LiPopt-2. As Latorre et al. (2020) pointed out, the degree has to be at least the depth of the network to compute a valid bound, so we have to use at least LiPopt-k for k-layer networks. LiPopt is unable to handle neural networks with more than two layers because this requires LiPopt with degrees beyond 2. Even if we only consider LiPopt-2 on two-layer networks, the running time is still much higher compared to GeoLIP. This demonstrates the great advantage of GeoLIP in terms of scalability compared with LiPopt. If we compare LiPopt-2 with GeoLIP on two-layer networks from Table 1, it is clear that GeoLIP produces more precise results. For networks with depth greater than 2, we can only compare GeoLIP with the matrix-norm-product method. As we can see from all experiments, GeoLIP's estimation on the FGL is always much lower than MP.

We have also shown that the two-layer network ℓ_∞-FGL estimation from GeoLIP has a theoretical guarantee with the approximation factor $K_G < 1.783$ (Theorem 3.3). If we compare the two-layer network results from GeoLIP and Sampling in Table 1, which is a lower bound of true Lipschitz constant, the ratio is within 1.783. This validates our theoretical claim.

RQ2. In Section 4, we have demonstrated the duality between NGeoLIP and LipSDP for the ℓ_2-FGL estimation on two-layer networks, even though the approaches appear drastically different. The experiments show that on two-layer networks, LipSDP and NGeoLIP for ℓ_2-FGL estimations (Table 7 in the appendix), and DGeoLIP and NGeoLIP for ℓ_∞-FGL estimations produce the same values. These results empirically validate the duality arguments, and also all the related SDP programs.

SDP relaxation. Applying SDP on intractable combinatorial optimization problem was pioneered by the seminal Goemans-Williamson algorithm for the Max-Cut problem (Goemans and Williamson, 1995). For two-layer networks, we have reduced the FGL estimation to the mixed-norm problem, and

[2] Another method was proposed by Chen et al. (2020), however, the code is not available and we are not able to compare it with GeoLIP.

provide approximation algorithms with ratios compatible with the known optimal constants in the corresponding mixed-norm problems. Improving them can be a very hard task. We also provide a compositional SDP interpretation of LipSDP-Neuron. Although Pauli et al. (2022) demonstrated the flaw in LipSDP-Network, our compositional SDP interpretation shows that LipSDP-Neuron is correct. In fact, from the compositional SDP interpretation, the program is only constrained by the underlying perturbation geometry and the layer-wise restriction from each hidden unit, so the constraints and objective exactly encode the FGL-estimation problem without additional assumptions. Because often the SDP relaxation for intractable problems gives the optimal known algorithms, we conjecture that GeoLIP and LipSDP are also hard to improve on FGL estimations.

Latorre et al. (2020) used polynomial optimization to address the ℓ_∞-FGL estimation. We argue that approaching FGL-estimations from the perspective of polynomial optimization loses the accurate characterization of this problem. For example, for two-layer networks, we have provided constant approximation algorithms to estimate FGLs in both ℓ_∞ and ℓ_2 cases. However, for a general polynomial optimization problem on a cube, we cannot achieve constant approximation. For example, the maximum independent set of a graph can be encoded as a polynomial optimization problem over a cube (Motzkin and Straus, 1965), but the maximum independent set problem cannot be approximated within a constant factor in polynomial time unless $\mathsf{P} = \mathsf{NP}$ (Trevisan, 2004).

7 Related work

Chen et al. (2020) employed polynomial optimization to compute the true Lipschitz constant of ReLU-networks for ℓ_∞-perturbations, and used Lasserre's hierarchy of SDPs (Lasserre, 2001) to solve the polynomial optimization problem. However, their approach is highly tailored to ReLU networks, while ours, like LipSDP, can handle common activations, such as sigmoid and ELU.

Latorre et al. (2020) also proposed to use LiPopt to estimate the local Lipschitz constant. However, estimating this quantity is not the problem studied in our work, and there are tools specifically designed for local perturbations and the Lipschitz constant (Laurel et al., 2022; Zhang et al., 2019).

Lipschitz regularization of neural networks is an important task, and recent works (Aziznejad et al., 2020; Bungert et al., 2021; Gouk et al., 2021; Krishnan et al., 2020; Terjék, 2020) have investigated this problem. However, here we study a related but different problem, i.e., Lipschitzness measurement of neural networks. Our work can motivate new Lipschitz regularization techniques.

8 Conclusion

In this work, we have provided a quantitative geometric framework for FGL estimations, and also algorithms for the ℓ_∞-FGL estimation. One important lesson is that when transferring techniques from one perturbations to another ones, we should also transfer the underlying geometry. One future work is to train smooth neural networks using the SDPs proposed in this paper.

Acknowledgments and Disclosure of Funding

The authors thank Vijay Bhattiprolu for introducing recent progress on the mixed-norm problems. The work is partially supported by Air Force Grant FA9550-18-1-0166, the National Science Foundation (NSF) Grants CCF-FMitF-1836978, IIS-2008559, SaTC-Frontiers-1804648, CCF-2046710 and CCF-1652140, and ARO grant number W911NF-17-1-0405. Zi Wang and Somesh Jha are partially supported by the DARPA-GARD problem under agreement number 885000.

References

Aws Albarghouthi. 2021. Introduction to Neural Network Verification. arXiv:2109.10317 [cs.LG]

Noga Alon and Assaf Naor. 2004. Approximating the Cut-Norm via Grothendieck's Inequality. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing* (Chicago, IL, USA) *(STOC '04)*. Association for Computing Machinery, New York, NY, USA, 72–80. `https://doi.org/10.1145/1007352.1007371`

MOSEK ApS. 2019. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.* http://docs.mosek.com/9.0/toolbox/index.html

Shayan Aziznejad, Harshit Gupta, Joaquim Campos, and Michael Unser. 2020. Deep Neural Networks With Trainable Activations and Controlled Lipschitz Constant. *IEEE TRANSACTIONS ON SIGNAL PROCESSING* 68 (2020), 4688–4699. https://doi.org/10.1109/TSP.2020.3014611

Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, and Irene Finocchi. 2018. A Survey of Symbolic Execution Techniques. *ACM Comput. Surv.* 51, 3, Article 50 (2018).

Aharon Ben-Tal and Arkadi Nemirovski. 2001. *Lectures on Modern Convex Optimization*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898718829 arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9780898718829

Vijay Bhattiprolu, Mrinalkanti Ghosh, Venkatesan Guruswami, Euiwoong Lee, and Madhur Tulsiani. 2018. Inapproximability of Matrix p→q Norms. *Electron. Colloquium Comput. Complex.* 25 (2018), 37.

Vijay Bhattiprolu, Euiwoong Lee, and Madhur Tulsiani. 2022. Separating the NP-Hardness of the Grothendieck Problem from the Little-Grothendieck Problem. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 215)*, Mark Braverman (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 22:1–22:17. https://doi.org/10.4230/LIPIcs.ITCS.2022.22

Mark Braverman, Konstantin Makarychev, Yury Makarychev, and Assaf Naor. 2011. The Grothendieck Constant is Strictly Smaller than Krivine's Bound. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. 453–462. https://doi.org/10.1109/FOCS.2011.77

Jop Briët, Oded Regev, and Rishi Saket. 2017. Tight Hardness of the Non-Commutative Grothendieck Problem. *Theory of Computing* 13, 15 (2017), 1–24. https://doi.org/10.4086/toc.2017.v013a015

Sebastien Bubeck and Mark Sellke. 2021. A Universal Law of Robustness via Isoperimetry. In *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (Eds.). https://openreview.net/forum?id=z71OSKqTFh7

Leon Bungert, René Raab, Tim Roith, Leo Schwinn, and Daniel Tenbrinck. 2021. CLIP: Cheap Lipschitz Training of Neural Networks. In *Scale Space and Variational Methods in Computer Vision*, Abderrahim Elmoataz, Jalal Fadili, Yvain Quéau, Julien Rabin, and Loïc Simon (Eds.). Springer International Publishing, Cham, 307–319.

Tong Chen, Jean B Lasserre, Victor Magron, and Edouard Pauwels. 2020. Semialgebraic Optimization for Lipschitz Constants of ReLU Networks. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 19189–19200. https://proceedings.neurips.cc/paper/2020/file/dea9ddb25cbf2352cf4dec30222a02a5-Paper.pdf

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1511.07289

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. 2019. Certified Adversarial Robustness via Randomized Smoothing. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 1310–1320. https://proceedings.mlr.press/v97/cohen19c.html

Inc. CVX Research. 2020. CVX: Software for Disciplined Convex Programming, Version 2.2, Build 1148. http://cvxr.com/cvx.

Steven Diamond and Stephen Boyd. 2016. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research* 17, 83 (2016), 1–5.

Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George Pappas. 2019. Efficient and Accurate Estimation of Lipschitz Constants for Deep Neural Networks. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/95e1533eb1b20a97777749fb94fdb944-Paper.pdf

Michel X. Goemans and David P. Williamson. 1995. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. ACM* 42, 6 (nov 1995), 1115–1145. https://doi.org/10.1145/227683.227684

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6572

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J. Cree. 2021. Regularisation of Neural Networks by Enforcing Lipschitz Continuity. *Mach. Learn.* 110, 2 (feb 2021), 393–416. https://doi.org/10.1007/s10994-020-05929-w

A. Grothendieck. 1956. Résumé de la théorie métrique des produits tensoriels topologiques. Bol. Soc. Mat. São Paulo 8, 1-79 (1956)..

Gurobi Optimization, LLC. 2022. Gurobi Optimizer Reference Manual. https://www.gurobi.com

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

Matthias Hein and Maksym Andriushchenko. 2017. Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 2263–2273.

Yujia Huang, Huan Zhang, Yuanyuan Shi, J Zico Kolter, and Anima Anandkumar. 2021. Training Certifiably Robust Neural Networks with Efficient Local Lipschitz Bounds. In *Thirty-Fifth Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=FTt28RYj5Pc

Matt Jordan and Alexandros G Dimakis. 2020. Exactly Computing the Local Lipschitz Constant of ReLU Networks. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 7344–7353. https://proceedings.neurips.cc/paper/2020/file/5227fa9a19dce7ba113f50a405dcaf09-Paper.pdf

Ravindran Kannan. 2010. Spectral Methods for Matrices and Tensors. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing* (Cambridge, Massachusetts, USA) *(STOC '10)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/1806689.1806691

Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Computer Aided Verification*, Rupak Majumdar and Viktor Kunčak (Eds.). Springer International Publishing, Cham, 97–117.

Subhash Khot. 2002. On the Power of Unique 2-Prover 1-Round Games. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing* (Montreal, Quebec, Canada) *(STOC '02)*. Association for Computing Machinery, New York, NY, USA, 767–775. https://doi.org/10.1145/509907.510017

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

Vishaal Krishnan, Abed AlRahman Al Makdah, and Fabio Pasqualetti. 2020. Lipschitz Bounds and Provably Robust Training by Laplacian Smoothing. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS'20)*. Curran Associates Inc., Red Hook, NY, USA, Article 917, 12 pages.

J.L Krivine. 1979. Constantes de Grothendieck et fonctions de type positif sur les sphères. *Advances in Mathematics* 31, 1 (1979), 16–30. https://doi.org/10.1016/0001-8708(79)90017-3

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* 60, 6 (may 2017), 84–90. https://doi.org/10.1145/3065386

Jean B. Lasserre. 2001. Global Optimization with Polynomials and the Problem of Moments. *SIAM Journal on Optimization* 11, 3 (2001), 796–817. https://doi.org/10.1137/S1052623400366802 arXiv:https://doi.org/10.1137/S1052623400366802

Jean Bernard Lasserre. 2015. *An Introduction to Polynomial and Semi-Algebraic Optimization*. Cambridge University Press. https://doi.org/10.1017/CBO9781107447226

Fabian Latorre, Paul Rolland, and Volkan Cevher. 2020. Lipschitz constant estimation of Neural Networks via sparse polynomial optimization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJe4_xSFDB

Jacob Laurel, Rem Yang, Gagandeep Singh, and Sasa Misailovic. 2022. A Dual Number Abstraction for Static Analysis of Clarke Jacobians. *Proc. ACM Program. Lang.* 6, POPL, Article 56 (jan 2022), 30 pages. https://doi.org/10.1145/3498718

Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010). http://yann.lecun.com/exdb/mnist/

Klas Leino, Zifan Wang, and Matt Fredrikson. 2021. Globally-Robust Neural Networks. In *International Conference on Machine Learning (ICML)*.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJzIBfZAb

MATLAB. 2021. *9.11.0.1837725 (R2021b) Update 2*. The MathWorks Inc., Natick, Massachusetts.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (Lake Tahoe, Nevada) *(NIPS'13)*. Curran Associates Inc., Red Hook, NY, USA, 3111–3119.

T. S. Motzkin and E. G. Straus. 1965. Maxima for Graphs and a New Proof of a Theorem of Turán. *Canadian Journal of Mathematics* 17 (1965), 533–540. https://doi.org/10.4153/CJM-1965-053-6

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning* (Haifa, Israel) *(ICML'10)*. Omnipress, Madison, WI, USA, 807–814.

Assaf Naor. 2013. Quantitative geometry. *Proceedings of the National Academy of Sciences* 110, 48 (2013), 19202–19205. https://doi.org/10.1073/pnas.1320388110 arXiv:https://www.pnas.org/content/110/48/19202.full.pdf

Yu Nesterov. 1998. Semidefinite relaxation and nonconvex quadratic optimization. *Optimization Methods and Software* 9, 1-3 (1998), 141–160. https://doi.org/10.1080/10556789808805690 arXiv:https://doi.org/10.1080/10556789808805690

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. 372–387. https://doi.org/10.1109/EuroSP.2016.36

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`

Patricia Pauli, Anne Koch, Julian Berberich, Paul Kohler, and Frank Allgöwer. 2022. Training Robust Neural Networks Using Lipschitz Bounds. *IEEE Control Systems Letters* 6 (2022), 121–126. `https://doi.org/10.1109/LCSYS.2021.3050444`

Prasad Raghavendra and David Steurer. 2009. Towards Computing the Grothendieck Constant. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms* (New York, New York) *(SODA '09)*. Society for Industrial and Applied Mathematics, USA, 525–534.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. 2018. Certified Defenses against Adversarial Examples. In *International Conference on Learning Representations*. `https://openreview.net/forum?id=Bys4ob-Rb`

R. Rietz. 1974. A proof of the Grothendieck inequality. *Israel Journal of Mathematics* 19 (1974), 271–276. `https://doi.org/10.1007/BF02757725`

Kevin Scaman and Aladin Virmaux. 2018. Lipschitz Regularity of Deep Neural Networks: Analysis and Efficient Estimation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 3839–3848.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). `http://arxiv.org/abs/1312.6199`

Dávid Terjék. 2020. Adversarial Lipschitz Regularization. In *International Conference on Learning Representations*. `https://openreview.net/forum?id=Bke_DertPB`

Luca Trevisan. 2004. Inapproximability of Combinatorial Optimization Problems. arXiv:cs/0409043 [cs.CC]

Zi Wang, Aws Albarghouthi, Gautam Prakriya, and Somesh Jha. 2022. Interval Universal Approximation for Neural Networks. *Proc. ACM Program. Lang.* 6, POPL, Article 14 (jan 2022), 29 pages. `https://doi.org/10.1145/3498675`

Huan Zhang, Pengchuan Zhang, and Cho-Jui Hsieh. 2019. RecurJac: An Efficient Recursive Algorithm for Bounding Jacobian Matrix of Neural Networks and Its Applications. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, 01 (Jul. 2019), 5757–5764. `https://doi.org/10.1609/aaai.v33i01.33015757`

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Our main claims in the abstract and introduction accurately reflect the paper's contributions and scope. In particular, we listed our contributions in Section 1, and forward referenced each contribution to the corresponding section in the paper.
   (b) Did you describe the limitations of your work? [Yes] We clearly defined what is the quantity to measure and what the network is in Section 2, and what the assumptions are for each theorem in Section 3.

(c) Did you discuss any potential negative societal impacts of your work? [Yes] We discussed them in Appendix D.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We have read the ethics review guidelines. Because our work is to measure the smoothness of neural networks (see Sections 1 and 2), and we only used the standard MNIST dataset (see Section 6.1), the paper conforms to guidelines.

2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes] We clearly defined the quantity to measure and the network structures in Sections 2 to 5.

(b) Did you include complete proofs of all theoretical results? [Yes] We provided important intuition and ideas in the main paper (see Sections 3 to 5), and included complete proofs in Appendix A.

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We included the code, data, and instructions as a URL.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We included major experimental setting in Section 6.1, and detailed specification in Appendix C.1.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A] Our experiments are deterministic. Given a neural network, our algorithm always returns the same result.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] This was provided in Appendix C.1.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] In terms of dataset, we only used the standard MNIST, and cited the creators. All the tools used in the paper were properly cited. See Section 6.1 and Appendix C.1.

(b) Did you mention the license of the assets? [Yes] See Appendix C.1.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We included our code as a URL.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Elided background, derivations and proofs

A.1 Additional analysis background

Gradient as operator. If a function $g : \mathbb{R}^m \to \mathbb{R}$ is a differentiable function at $a \in \mathbb{R}^m$, then the total derivative of g at a is

$$Dg(a) = [\frac{\partial g}{\partial x_1}(a), \ldots, \frac{\partial g}{\partial x_m}(a)],$$

and the gradient of g at a is $\nabla g(a)$ is the transpose of $Dg(a)$. The linear approximation of g at a is $\langle Dg(a), dx \rangle$. Equivalently, we can view the change of a function with respect to an infinitesimal

perturbation as the inner product of $\nabla g(a)$ and dx. In this sense, the gradient acts as an operator on the perturbation.

Differentiable activation. Because we want to upper bound the true Lipschitz constant, we only need to show that the quantity considered in the paper indeed upper bounds the true Lipschitz constant considered in the paper. If the activation function is differentiable, then the neural network f is also differentiable, so Equation (3) is trivially true, as proved and applied in Latorre et al. (2020, Theorem 1 and Equation 4).

ReLU activation. For ReLU networks, it is true if we have $[a, b] = [0, 1]$. One can consider the (Clarke) generalized Jacobian as in Jordan and Dimakis (2020). At each input point, the Clarke Jacobian is contained in $\{(W^1)^T \cdot \text{diag}(v^2) \cdot \cdots \cdot \text{diag}(v^d)(W^d)^T \mid v^i \in [a, b]^{n_i}\}$. Alternatively, we can also use the perturbation propagation argument in Section 4 to see this upper bound. Note that Raghunathan et al. (2018) used this interval representation for ReLU's derivative.

Maximum over hypercube. Now we want to show that the optimization problems over hypercubes considered in this work attain the maximum at the vertices. Without of loss of generality, let us assume the hypercube is $[-1, 1]^n$. Otherwise, we can transform the hypercube to $[-1, 1]^n$. Let $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, and $z \in \mathbb{R}^n$.

We will use the following facts

1. $||x||_1 = \max_{z\in\{-1,1\}^n} \langle x, z\rangle$;
2. ℓ_∞ is the dual of ℓ_1;
3. Let $U \subseteq \mathbb{R}^n$. When $\max_{x,z\in U}\langle Ax, Az\rangle$ is well-defined, we have $\max_{x\in U}\langle Ax, Ax\rangle = \max_{x,z\in U}\langle Ax, Az\rangle$.

The first fact is from $||x||_1 = |x_1| + \ldots + |x_n| = \max_{z\in\{-1,1\}^n} \langle x, z\rangle$. The second fact is from Hölder's inequality for finite-dimensional vector space. For the third one, $\langle Ax, Az\rangle$ is maximized only when $Ax = Az$. Now we can show that the maximization problems considered in this paper attain the maximum at the hypercube vertices.

$$\begin{aligned}
\max_{||x||_\infty=1} ||Ax||_1 &= \max_{||x||_\infty=1, y\in\{-1,1\}^m} \langle Ax, y\rangle && \text{(From fact (1))} \\
&= \max_{||x||_\infty=1, y\in\{-1,1\}^m} \langle x, A^T y\rangle \\
&= \max_{y\in\{-1,1\}^m} ||A^T y||_1 && \text{(From fact (2))} \\
&= \max_{x\in\{-1,1\}^n, y\in\{-1,1\}^m} \langle x, A^T y\rangle \\
&= \max_{x\in\{-1,1\}^n, y\in\{-1,1\}^m} \langle Ax, y\rangle.
\end{aligned} \tag{14}$$

$$\begin{aligned}
\max_{||x||_\infty=1} ||Ax||_2^2 &= \max_{||x||_\infty=1} \langle Ax, Ax\rangle \\
&= \max_{||x||_\infty=1, ||z||_\infty=1} \langle Ax, Az\rangle && \text{(From fact (3))} \\
&= \max_{||x||_\infty=1, ||z||_\infty=1} \langle A^T Ax, z\rangle.
\end{aligned}$$

Using the similar idea in Equation (14), we have

$$\max_{||x||_\infty=1} ||Ax||_2^2 = \max_{x\in\{-1,1\}^n, z\in\{-1,1\}^n} \langle Ax, Az\rangle = \max_{x\in\{-1,1\}^n} \langle Ax, Ax\rangle.$$

More generally, in the bilinear forms considered above, if $x = x_1 \otimes \cdots \otimes x_d$ is generated by the tensor product of variables over cubes, we can fix one variable and write x as a matrix product, and then move the fixed variable to the hypercube vertices. We can repeat this process to move all variables to the vertices.

A.2 Additional definitions

For any neural network f, let $OPT(f)$ be the optimal value of Equation (3). We say an algorithm $\mathcal{A}$ is an approximation algorithm for Equation (3) with approximation ratio $\alpha > 1$, if $OPT(f) \leq \mathcal{A}(f) \leq \alpha OPT(f)$.

A.3 SDP for the $\infty \to 1$ mixed-norm problem

Recall that for $v \in \mathbb{R}^m$, $||v||_1 = \max_{||u||_\infty=1} \langle u, v\rangle$. We can reformulate the mixed-norm problem as follows:

$$\max_{x\in\{-1,1\}^n} ||Ax||_1 = \max_{(x,y)\in\{-1,1\}^{n+m}} \langle Ax, y\rangle.$$

If we let $z = \begin{pmatrix} x^T & y^T \end{pmatrix}$, we can have

$$\max_{(x,y)\in\{-1,1\}^{n+m}} \langle Ax, y\rangle = \max_{z\in\{-1,1\}^{n+m}} z \cdot B \cdot z^T,$$

where B is a $(m+n) \times (m+n)$ matrix. The last m rows and first n columns of B is A, and the rest are 0: $B = \begin{pmatrix} 0 & 0 \\ A & 0 \end{pmatrix}$.

The natural SDP relaxation of the $\infty \to 1$ mixed-norm problem is:

$$\max \text{tr}(BX)$$
$$s.t.\ X \succeq 0, X_{ii} = 1, i \in [n+m].$$

In other words, we treat X as the SDP matrix relaxed from the rank-1 matrix $z^T \cdot z$.

A.4 SDP relaxation and Grothendieck inequalities

In this work, we used the Grothendieck inequality as in Theorem 3.1:

$$\max_{u_i,v_j\in B(H)} \sum_{i,j} A_{ij}\langle u_i, v_j\rangle_H \leq K_G ||A||_{\infty\to 1}, \tag{15}$$

for any $A \in \mathbb{R}^{n\times m}$; and the little Grothendieck inequality:

$$\max_{u_i,v_j\in B(H)} \sum_{i,j} (A^T A)_{ij}\langle u_i, v_j\rangle_H \leq \frac{\pi}{2} ||A||^2_{\infty\to 2}. \tag{16}$$

Notice that $A^T A$ is a PSD matrix.

As discussed in Appendix A.3,

$$||A||_{\infty\to 1} = \max_{z\in\{-1,1\}^{n+m}} z \cdot B \cdot z^T.$$

The natural SDP relaxation is

$$\max \text{tr}(BX)$$
$$s.t.\ X \succeq 0, X_{ii} = 1, i \in [n+m].$$

Because $X \succeq 0$, $X = MM^T$ for some $M \in \mathbb{R}^{(m+n)\times d}$, where $d \geq 1$. Let M_i be the i-th row vector of M. $X_{ij} = \langle M_i, M_j\rangle$, and $X_{ii} = 1$ means $\langle M_i, M_i\rangle = 1$. As a result, $\text{tr}(BX) = \sum_{i,j} A_{ij}X_{ij} = \sum_{i,j} A_{ij}\langle M_i, M_j\rangle_H$, where H is the Hilbert space of $\mathbb{R}^d$ equipped with the canonical inner product. Thus, Equation (15) implies that K_G is the approximation ratio in the SDP relaxation for the mixed-norm problem.

In contrast, in the mixed-norm problem, the variable to B_{ij} is $z_i z_j$, the product of two scalars. If $d = 1$ in the SDP relaxation, M is a column vector, and X is a rank-1 matrix. In this case, the SDP coincides with the combinatorial problem, because the inner product degenerates to the multiplication of two scalars. Hence, the SDP relaxation can be viewed as a continuous relaxation of a discrete problem, and Equation (15) quantifies this geometric transformation. Another interpretation for the SDP relaxation is that SDP drops the rank-1 constraint in the quadratic formulation of the mixed-norm problem.

A.5 Rescaling from 0-1 cube to norm-1 cube

Now let us show how we transform the 0-1 cube in Equation (4) to a norm-1 cube, and formulate an equivalent optimization problem. As a result, we can apply the SDP program in Equation (5) to compute an upper bound of the ℓ_∞-FGL.

Let $x_i = (t_i + 1)/2$, where $t_i \in \{-1, 1\}$. We have

$$\begin{aligned} &\max_{x\in\{0,1\}^n} ||Ax||_1 \\ =&\max_{x\in\{0,1\}^n, y\in\{-1,1\}^m} y^T Ax \\ =&\max_{t\in\{-1,1\}^n, y\in\{-1,1\}^m} \frac{1}{2} y^T A(t + e_n). \end{aligned} \tag{17}$$

Let OPT_1 be the optimal value of

$$\max_{(t,y)\in\{-1,1\}^{n+m}} y^T A(t + e_n).$$

Introduce another variable $\tau \in \{-1, 1\}$, and let OPT_2 be the optimal value of

$$\max_{(t,y,\tau)\in\{-1,1\}^{n+m+1}} y^T A(t + \tau e_n). \tag{18}$$

Lemma A.1. $OPT_1 = OPT_2$.

Proof. Clearly $OPT_2 \geq OPT_1$.

Now if $(\hat{t}, \hat{y}, \tau = -1)$ is an optimal solution to Equation (18), then $(-\hat{t}, -\hat{y}, \tau = 1)$ is also an optimal solution, so $OPT_2 \leq OPT_1$. □

Now let $z = (t, \tau)$, and we can verify that $y^T A(t + \tau e_n) = y^T Bz$, where $B = (A \quad Ae_n)$.

As a result, the semidefinite program to the ℓ_∞-FGL constant is

$$\max \frac{1}{2}\text{tr}(BX)$$
$$s.t.\ X \succeq 0, X_{ii} = 1, i \in [n + m + 1],$$

where B is a $(n + 1 + m) \times (n + 1 + m)$ matrix, and $B = \begin{pmatrix} 0 & 0 & 0 \\ A & Ae_n & 0 \end{pmatrix}$.

A.6 Proof of Theorem 3.2

Proof. We will use the cube rescaling techniques introduced in Appendix A.5. Alon and Naor (2004) showed that matrix cut-norm is MAXSNP-hard. We will show that if one can solve the FGL estimation problem, then one can find the cut norm of a matrix.

Given a matrix A, the cut norm of a matrix $A \in \mathbb{R}^{m\times n}$ is defined as

$$CN(A) = \max_{x\in\{0,1\}^n, y\in\{0,1\}^m} \langle Ax, y\rangle.$$

We need to transform y from 0-1 cube to norm-1 cube, so similarly let $y_i = (t_i + 1)/2$, where $t_i \in \{-1, 1\}$. The we will have

$$CN(A) = \max_{x\in\{0,1\}^n, y\in\{0,1\}^m} \langle Ax, y\rangle = \frac{1}{2} \max_{x\in\{0,1\}^n, t\in\{-1,1\}^m} \langle Ax, (t + e_m)\rangle.$$

Let $B = \begin{pmatrix} A \\ e_m^T A \end{pmatrix}$. From above we know that

$$\max_{x\in\{0,1\}^n, t\in\{-1,1\}^m} \langle Ax, (t + e_m)\rangle = \max_{x\in\{0,1\}^n, (t,\tau)\in\{-1,1\}^{m+1}} \langle Bx, (t, \tau)\rangle.$$

One can then construct a two layer neural network, where the first weight matrix is B^T, and the second weight matrix is $(1, \ldots, 1) \in \mathbb{R}^n$. Because the network we consider has only one output, the second weight matrix is only a vector. The FGL of this network is exactly twice of the cut norm of A. □

A.7 Proof of Theorem 3.3

Proof. Let $B = \begin{pmatrix} 0 & 0 & 0 \\ A & Ae_n & 0 \end{pmatrix}$. Combing Equations (5), (17) and (18), the approximation algorithm for Equation (4) where $q = 1$ is induced by the following SDP program:

$$\max \frac{1}{2}\text{tr}(BX)$$
$$s.t.\ X \succeq 0, X_{ii} = 1, i \in [n+m+1].$$

□

A.8 Natural SDP relaxation of ℓ_2-FGL estimation

Now let $q = 2$ in Equation (4), we will have:

$$\max_{y \in \{0,1\}^n} ||Ay||_2.$$

In other words, we only need to solve the following program:

$$\max_{z \in \{0,1\}^n} z^T (A^T A) z. \tag{19}$$

Let $M = A^T A$, then M is a PSD matrix. We have demonstrated the scaling techniques in Appendix A.5. Let $x \in \{-1, 1\}^{n+1}$, one can verify that

$$\max_{z \in \{0,1\}^n} z^T M z = \frac{1}{4} \max_{x \in \{-1,1\}^{n+1}} x^T \hat{M} x, \tag{20}$$

where $\hat{M} = \begin{pmatrix} M & Me_n \\ e_n^T M & e_n^T M e_n \end{pmatrix}$.

It is easy to verify that if M is PSD, $\hat{M}$ is also PSD. Because $M = A^T A$, $\hat{M} = (A, Ae_n)^T \cdot (A, Ae_n)$. Now we can consider the following natural SDP relaxation to $\max_{x \in \{-1,1\}^{n+1}} x^T \hat{M} x$:

$$\begin{aligned} &\max \text{tr}(\hat{M}X) \\ &s.t.\ X \succeq 0, X_{ii} = 1, i \in [n+1]. \end{aligned} \tag{21}$$

This SDP relaxation admits a $\frac{\pi}{2}$-approximation factor from Equation (16) (Rietz, 1974; Nesterov, 1998).

A.9 Proof of Theorem 3.4

Proof. Let $\hat{M} = \begin{pmatrix} M & Me_n \\ e_n^T M & e_n^T M e_n \end{pmatrix}$, where $M = A^T A$. Combining Equations (19) to (21), the approximation algorithm for Equation (4) where $q = 2$ is induced by the following SDP program:

$$\begin{aligned} &\max \frac{1}{2}\sqrt{\text{tr}(\hat{M}X)} \\ &s.t.\ X \succeq 0, X_{ii} = 1, i \in [n+1]. \end{aligned} \tag{22}$$

□

A.10 Comparison with Raghunathan et al. (2018)

Raghunathan et al. (2018) formulated the following SDP to upper bound the ℓ_∞-FGL on two-layer neural networks:

$$\begin{aligned} &\max \frac{1}{4}\text{tr}(CX) \\ &s.t.\ X \succeq 0, X_{ii} = 1, i \in [n+m+1], \end{aligned} \tag{23}$$

where C is a $(m+n+1)\times(m+n+1)$ matrix, and $C = \begin{pmatrix} 0 & 0 & A^T \\ 0 & 0 & e_n^T A^T \\ A & Ae_n & 0 \end{pmatrix}$.

If we compare Equations (6) and (23), $C = B + B^T$. Because X is symmetric, $\text{tr}(CX) = 2\text{tr}(BX)$. Therefore, Equations (6) and (23) produce the same result.

A.11 Equivalence between the new optimization program and Equation (19)

Notice that because $u \in \mathbb{R}^{1\times n}$, $u\Delta\sigma(x)$ is a scalar. We can view each z_i in Equation (19) as $\frac{\Delta\sigma(x)_i}{\Delta y_i}$, the derivative of $\sigma(x)_i$ without the limit. Therefore, $\Delta\sigma(x)_i = z_i\Delta y_i$. Recall that from Section 4, $\Delta y_i = w_i\Delta x$, so $\Delta\sigma(x)_i = w_i z_i \Delta x$, then $u\Delta\sigma(x) = \Delta x \sum_i^n u_i z_i w_i = \Delta x(Az)$, where $A = W^T\text{diag}(u)$ as defined in Equation (4).

As a result, from Cauchy–Schwarz inequality, the above objective is

$$\max_{\Delta x, \Delta\sigma(x)} \frac{(u\Delta\sigma(x))^2}{(\Delta x)^2} = \max_z (Az)^2,$$
$$s.t.\ z \in [a,b]^n.$$

This demonstrates the equivalence between the new optimization program and Equation (19) when $[a,b] = [0,1]$ for $\sigma = \text{ReLU}$.

B Polynomial optimization approach to the FGL estimation

We briefly discuss the gradient approach to estimate the FGL. Let us use a three layer network as an example:

$$f(x) = u\sigma(V\sigma(Wx + b_1) + b_2),$$

where $x \in \mathbb{R}^{l\times 1}$, $W \in \mathbb{R}^{n\times l}$, $b_1 \in \mathbb{R}^n$, $V \in \mathbb{R}^{m\times n}$, $b_2 \in \mathbb{R}^m$ and $u \in \mathbb{R}^{1\times m}$.

The formal gradient vector of this network is

$$W^T\text{diag}(y)V^T\text{diag}(z)u^T,$$

where $\text{diag}(y) \in \mathbb{R}^{n\times n}$ and $\text{diag}(z) \in \mathbb{R}^{m\times m}$. The i-th component of this vector is then

$$\sum_{k=1}^{m}\sum_{j=1}^{n}(u_k V_{kj} W_{ji}) \cdot (y_j z_k).$$

Therefore, the ℓ_p-norm estimation of the formal gradient ends up being a polynomial optimization problem over a cube. For example, the ℓ_1-norm (corresponding to ℓ_∞-perturbations) of the gradient is

$$\max_{x_i\in\{-1,1\}, y_j\in\{0,1\}, z_k\in\{0,1\}} \sum_{i,j,k=1}^{l,n,m} T_{ijk} \cdot x_i y_j z_k, \tag{24}$$

where $T_{ijk} = W_{ji}V_{kj}u_k$.

This is essentially a tensor cut-norm problem, and it is an open problem whether there exists an approximation algorithm within a constant factor to the general tensor cut-norm problem (Kannan, 2010). Notice that Equation (24) is not a general tensor-cut-norm problem, because the tensor is generated from the weight matrices. For example, if we fix j, the projected matrices of T are of rank-1. Each vector in $T_{:,j,:}$ is the product of $V_{kj}u_k$ with the vector $W_{j:}$:

$$\forall k : T_{:,j,k} = W_{j:}V_{kj}u_k.$$

However, we do not have the theoretical technique to exploit the low-rank structure of these special polynomial optimization problems. The perturbation analysis in Sections 4 and 5 can be viewed as exploiting this structure in practice.

Table 3: ℓ_∞-FGL estimations of different methods for two-layer networks: DGeoLIP and NGeoLIP induce the same estimations, and they are also close to the sampled lower bounds. In the meantime, the result from GeoLIP is tighter than LiPopt's result.

#UNITS	DGEOLIP	NGEOLIP	LIPOPT-2	MP	SAMPLE	BRUF
8	142.19	142.19	180.38	411.90	134.76	134.76
16	185.18	185.18	259.44	578.54	175.24	175.24
64	287.60	287.60	510.00	1207.70	253.89	N/A
128	346.27	346.27	780.46	2004.34	266.22	N/A
256	425.04	425.04	1011.65	2697.38	306.98	N/A

Table 4: Average running time (in seconds) of different methods for two-layer-network ℓ_∞-FGL estimations: GeoLIPs are faster than LiPopt.

# HIDDEN UNITS	DGEOLIP	NGEOLIP	LIPOPT-2	BRUF
8	23.1	21.5	1533	< 0.1
16	28.1	22.3	1572	4.8
64	93.4	31.7	1831	N/A
128	292.5	42.2	2055	N/A
256	976.0	70.9	2690	N/A

C Complete experimental specifications and results

GeoLIP is available at `https://github.com/z1w/GeoLIP`. To accommodate users who do not have access to MATLAB, we also implement a version based on CVXPY (Diamond and Boyd, 2016). However, the MATLAB implementation works more efficiently in terms of memory and speed, and we encourage users to work with the MATLAB version when possible. We conducted all the GeoLIP-related experiments with the MATLAB version.

C.1 Experimental specifications

Tools. We obtain the LiPopt implementation from `https://github.com/latorrefabian/lipopt`, under the MIT License.

Server specification. All the experiments are run on a workstation with forty-eight Intel® Xeon® Silver 4214 CPUs running at 2.20GHz, and 258 GB of memory, and eight Nvidia GeForce RTX 2080 Ti GPUs. Each GPU has 4352 CUDA cores and 11 GB of GDDR6 memory.

Dataset and split. We used the standard MNIST dataset from the PyTorch package (Paszke et al., 2019). We used the "train" parameter in the MNIST function to split training and testing data.

C.2 Experimental results

Single hidden layer. We consider the ℓ_∞-FGL estimation on two layer neural networks with different numbers of hidden units. The results are summarized in Tables 3 and 4.

Multiple hidden layers. We consider the ℓ_∞-FGL estimation on 3, 7, 8-layer neural networks. Each hidden layer in the network has 64 ReLU units. The results are summarized in Tables 5 and 6.

ℓ_2-FGL estimation. We measure the ℓ_2-FGL on two-layer networks mainly to compare whether Equation (7) and LipSDP produce the same result. Additionally, we also want to empirically examine the approximation guarantee from Theorem 3.4. Still, we consider networks with 8, 16, 64, 128, 256 hidden nodes. The results are summarized in Tables 7 and 8.

Table 5: ℓ_∞-FGL estimations of different methods for multi-layer networks: GeoLIP's result is much tighter than the matrix-product method, and LiPopt is unable to handle these networks.

# LAYERS	GEOLIP	MATRIX PRODUCT	SAMPLE	LIPTOPT
3	529.42	9023.65	311.88	N/A
7	5156.5	$1.423 * 10^7$	1168.8	N/A
8	8327.2	$8.237 * 10^7$	1130.6	N/A

Table 6: Average running time (in seconds) of GeoLIP for multi-layer-network ℓ_∞-FGL estimations.

3-LAYER NET	7-LAYER NET	8-LAYER NET
120.9	284.3	329.5

C.3 Additional discussions

Duality. The results in Table 7 show that the results of LipSDP and GeoLIP on two-layer-network ℓ_2-FGL estimation are exactly the same, which empirically demonstrates the duality between LipSDP and GeoLIP, as discussed in Section 4. Though Pauli et al. (2022) showed that the most precise version of LipSDP is invalid for estimating an upper bound of ℓ_2-FGL, our dual-program argument shows that the less precise version of LipSDP is correct.

Precision. We showed that GeoLIP's approximation factor for the ℓ_2-FGL estimation on two layer networks is $\sqrt{\frac{\pi}{2}} \approx 1.253$ in Theorem 3.4. The ℓ_2-FGL from GeoLIP is very close to the sampled lower bound of true Lipschitz constant in Table 7. On the other hand, because the result from GeoLIP is an upper bound of FGL, and this result is not much greater than the sampled lower bound of true Lipschitz constant, this empirically demonstrates that the true Lipschitz constant is not very different from the FGL on two-layer networks.

Running time. If we compare the running time in Tables 4 and 8, the dual program takes more time to solve than the natural relaxation. This is particularly true when the number of hidden neurons increases. From the reported numbers of variables and equality constraints by CVX, the dual program and natural relaxation have similar numbers. It is also observed that the CPU usage is higher when the natural relaxation is being solved. We want to point out that the running time and optimization algorithm are solver-dependent, and efficiently solving SDP is beyond the scope of this work. It is an interesting future direction to exploit the block structure of the dual programs, and develop algorithms that are compatible with those programs, because training smooth networks is a critical task, and it is promising to incorporate the SDP programs.

ℓ_2 versus ℓ_∞ FGLs. If we compare results from Tables 3 and 7, we can also find that the discrepancy between matrix product method and sampled lower bound is much smaller in the ℓ_2 case. This could also explain why Gloro works for ℓ_2-perturbations but not the ℓ_∞ case in practice, where Leino et al. (2021) used matrix-norm product to upper bound the Lipschitz constant of the network in Gloro.

Sampling. Sampling can only give a lower bound of the true Lipschitz constant, while we are trying to estimate an upper bound. We use sampling as a sanity check to ensure that the SDP method is at least sound and indeed provides an upper bound of the FGL. It is interesting to see that in networks where we can brute-force enumerate all the activation patterns, sampling provides very close results to the ground-truth ones. Notice that for those networks, there are only a few hidden units (8 or 16), while we sample many (200,000) inputs, which might activate all or most of the patterns. However, for networks with many activation nodes, it is infeasible to have a brute-force enumeration of all the activation patterns, so we do not have the ground-truth information. Sampling has no guarantee whether it can activate all patterns unless we have sampled all possible inputs, which is also impractical.

Multi-layer network guarantees. The discrepancy between the results from sampling and GeoLIP is relatively large for multi-layer networks. The approximation guarantee of GeoLIP is in terms of

Table 7: ℓ_2-FGL estimations of different methods for two-layer networks: LipSDP and NGeoLIP induce the same estimations, and these results are also close to the sampled lower bounds.

#UNITS	NGEOLIP	LIPSDP	MP	SAMPLE	BRUF
8	6.531	6.531	11.035	6.527	6.527
16	8.801	8.801	13.936	8.795	8.799
64	12.573	12.573	22.501	11.901	N/A
128	15.205	15.205	30.972	13.030	N/A
256	18.590	18.590	35.716	14.610	N/A

Table 8: Average running time (in seconds) of LipSDP and NGeoLIP for two-layer-network ℓ_2-FGL estimations.

# HIDDEN UNITS	LIPSDP	NGEOLIP	BRUF
8	11.5	1.2	< 0.1
16	15.7	1.2	5.1
64	64.2	1.3	N/A
128	216.1	1.7	N/A
256	758.1	4.1	N/A

the FGL, rather than true Lipschitz constant. It is unclear how large the gap between true Lipschitz constant and the FGL is for multi-layer networks. Narrowing this gap is an interesting research direction and beyond the scope of this work. We do not know whether for multi-layer networks, GeoLIP has an approximation guarantee that is independent of the network. We leave this as an open problem.

D Negative societal impacts

Our work is mainly theoretical and to measure an intrinsic mathematical property of neural networks, and can benefit the verification of deep-learning systems. A misuse of our work can give a false sense of safety, so the practical use of our work should be careful.