Estimating Neural Network Robustness via Lipschitz Constant and Architecture Sensitivity

Abulikemu Abuduweili
Carnegie Mellon University
abulikea@andrew.cmu.edu

Changliu Liu
Carnegie Mellon University
cliu6@andrew.cmu.edu

Abstract: Ensuring neural network robustness is essential for the safe and reliable operation of robotic learning systems, especially in perception and decision-making tasks within real-world environments. This paper investigates the robustness of neural networks in perception systems, specifically examining their sensitivity to targeted, small-scale perturbations. We identify the Lipschitz constant as a key metric for quantifying and enhancing network robustness. We derive an analytical expression to compute the Lipschitz constant based on neural network architecture, providing a theoretical basis for estimating and improving robustness. Several experiments reveal the relationship between network design, the Lipschitz constant, and robustness, offering practical insights for developing safer, more robust robot learning systems.

Keywords: Robustness, Lipschitz Continuity, Robot Learning, Neural Networks

1 Introduction

Deep neural networks have been successfully applied to many tasks in robotics [1], including perception [2], prediction [3], planning, and control [4]. However, their sensitivity to input perturbations poses significant challenges [5], particularly in safety-critical applications like autonomous driving and human-robot collaboration [6]. For instance, a small but well-designed modification to an input image can cause a neural network used in perception systems to misclassify the image [7], undermining its reliability in applications like autonomous driving or robotic navigation. Such issues raise concerns about the deployment of these models in real-world robotic systems, where safety and robustness are critical. In autonomous driving, adversarial examples could trigger unintended actions, such as incorrect path planning or object misdetection, potentially compromising the safety of autonomous systems.

In general, there are two different approaches one can evaluate the robustness of a neural network [8]: adversarial attack-based methods that demonstrate an upper bound, or verification-based methods that prove a lower bound. Adversarial attack approaches are easy to conduct, but the upper bound may not be useful [9]. Verification-based approaches, while sound, are substantially more difficult to implement in practice, and all attempts have required approximations [10]. Besides the adversarial attack and verification-based methods for evaluating the robustness, some works study the neural network's intrinsic robustness based on the Lipschitz continuity of neural network [11]. In deep neural networks, tight bounds on its Lipschitz constant can be extremely useful in a variety of applications: (1) The adversarial robustness of a neural network is closely related to its Lipschitz continuity [12]. Constraining local Lipschitz constants in neural networks is helpful in avoiding adversarial attacks [11]. (2) Generalization bounds critically rely on the Lipschitz constant of the neural networks in deep learning theory[13]. In these applications and many others, it is essential to estimate the Lipschitz constant both accurately and efficiently.

Various approaches have been proposed to measure and control the Lipschitz constant of neural networks [14]. Early work by Szegedy *et al.* [5] introduced an upper bound based on spectral norms

Workshop on Safe and Robust Robot Learning for Operation in the Real World (SAFE-ROL) at CoRL 2024.

of linear layers. Fazlyab *et al.* developed a convex programming framework for tighter bounds, and Shi *et al.* computed relatively precise Lipschitz constants using bound propagation techniques[15]. While these methods provide numerical estimates, they do not fully explore the relationship between neural network architecture and the corresponding Lipschitz constants.

In this work, we propose an analytical expression for the Lipschitz constant, tailored to different neural network architectures. This analytical approach provides a deeper understanding of how network design influences robustness, particularly in robotic perception systems where maintaining reliability under diverse environmental conditions is crucial. Through experiments, we validate the accuracy of our analytical expression and investigate how network depth, width, and other architectural choices impact robustness. Our goal is to identify architectures that achieve minimal Lipschitz constants, thereby maximizing robustness under comparable accuracy levels — a key consideration for safe and robust robot learning systems deployed in the real world. Our contributions can be summarized as:

- We present an expression for the Lipschitz constant based on the architecture of neural networks, providing valuable insights into designing robust neural networks for robot learning systems.
- The experimental results validate the proposed mathematical expression and demonstrate the relationship between neural network architecture and its robustness in perception tasks.
- In conclusion, our findings indicate that wider networks tend to be more robust than narrower ones, shallower networks exhibit greater robustness compared to deeper ones, and neural networks with weights of lower variance are generally more robust than those with higher variance.

2 Related Works

Robust Learning in Robotics. Robust robot learning focuses on the development of autonomous systems that can operate reliably and safely within dynamic and uncertain environments [16, 17]. Various approaches have been implemented to enhance robustness, including safe reinforcement learning, which aims to optimize policies while ensuring safety constraints [18, 19], and techniques like domain adaptation [20] and online adaptation [21, 22], which facilitate system adaptability across diverse environments. Furthermore, methods such as adversarial training and imposing constraints on the Lipschitz constant have shown promise in enhancing the robustness of robot learning systems by mitigating vulnerability to environmental variations and adversarial inputs [23, 11].

Adversarial Training and Verification. Adversarial training is a commonly used technique for improving the robustness of a neural network [7]. However, while such an adversarially trained network is made robust to some attacks in training, it can still be vulnerable to unseen attacks. Then neural Network Verification can be used to prove a lower bound on the robustness of neural networks [10]. Certified robust training under verification, focusing on training neural networks with certified and provable robustness – the network is considered robust on an example if and only if the prediction is provably correct for any perturbation in a predefined set [24, 25].

Lipschitz constant and Robustness. Different from certified robust training, some researchers bound the sensitivity of the function to input perturbations by bounding the Lipschitz constant to certify or improve the robustness of neural networks [11]. Weng *et al.* [26] convert the robustness analysis problem into a local Lipschitz constant estimation problem. Designing and training neural networks with bounded Lipschitz constant is a promising way to obtain certifiably robust classifiers [27]. Many works handle the Lipschitz bound by leveraging specific mathematical properties such as the spectral norm [28, 29]. Unlike previous studies, this work focuses on deriving an analytical expression for the Lipschitz constant based on the architecture of a neural network. We investigate the relationship between neural network architecture and robustness. While prior research typically calculates the Lipschitz constant using exact network parameters and numerical methods, our approach emphasizes the role of architecture rather than specific parameters in estimating the Lipschitz constant.

3 Lipschitz Continuity of Neural Networks

Notation. We use boldface letters to denote vectors (e.g., $\boldsymbol{x}$) or vector functions (e.g., $\boldsymbol{f}$), and use x_i or f_i to denote its i-th element. We use capital letters to denote matrices (e.g., $\mathbf{W}$), and use $W_{i,j}$ to denote the element of i-th row and j-th column. For a unary function σ, $\sigma(\boldsymbol{x})$ applies $\sigma(\cdot)$ element-wise on vector $\boldsymbol{x}$. The l_p-norm ($p \geq 1$) and l_∞-norm of a vector $\boldsymbol{x}$ are defined as $\|\boldsymbol{x}\|_p = (\sum_i |x_i|^p)^{\frac{1}{p}}$ and $\|\boldsymbol{x}\|_\infty = \max_i |x_i|$, respectively. In the following, we consider a real-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$. We mainly consider perturbation in l_p norm, e.g. $||\delta||_p$.

3.1 Neural Networks

In this work, we consider standard neural networks composed of affine layers (such as multilayer perceptrons or convolutional layers) combined with element-wise activation functions.

$$\boldsymbol{x}^{(l)} = \sigma^{(l)}(\tilde{\boldsymbol{x}}^{(l)}), \quad \tilde{\boldsymbol{x}}^{(l)} = \mathbf{W}^{(l)}\boldsymbol{x}^{(l)-1} + \boldsymbol{b}^{(l)} \tag{1}$$

Here M is the number of layers and usually $\sigma^{(M)}(\boldsymbol{x}) = \boldsymbol{x}$ is the identity function. The network takes $\boldsymbol{x}^{(0)} := \boldsymbol{x}$ as the input and outputs $\boldsymbol{x}^{(M)} := \boldsymbol{y}$. We use $\boldsymbol{f}$ to denote the whole neural network functions:

$$\boldsymbol{f}(\boldsymbol{x}) = \sigma^{(l)}(\mathbf{W}^{(l)}(\cdots \sigma^{(1)}(\mathbf{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})) + \boldsymbol{b}^{(l)}) \tag{2}$$

In this work, we mainly consider the classification tasks. Let $\boldsymbol{x} \in \mathbb{R}^n$ be an input vector of a m-class classification function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$. The predicted logits (with softmax activation) is $\boldsymbol{y} = [y_1, y_2, \cdots, \hat{y}_m] = \boldsymbol{f}(\boldsymbol{x})$, and the predicted class is given as $c(\boldsymbol{x}) = \arg\max_{1 \leq i \leq m} y_i$.

3.2 Lipschitz Continuous

Definition 1 (Lipschitz continuous.) A function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ is called L-Lipschitz continuous w.r.t. norm $\|\cdot\|$ if there exists a constant L for any pair of inputs $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^m$, such that

$$\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|. \tag{3}$$

The smallest L for which the previous inequality is true is called the Lipschitz constant of $\boldsymbol{f}$. For local Lipschitz functions (i.e. functions whose restriction to some neighborhood around any point is Lipschitz), the Lipschitz constant may be computed using its differential operator.

Theorem 1 (Rademacher [30], Theorem 3.1.6) If $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ is is a Lipschitz continuous function, and $\boldsymbol{f}$ is differentiable almost everywhere, then

$$L = \sup_{\boldsymbol{x}} \|\nabla \boldsymbol{f}(\boldsymbol{x})\|, \tag{4}$$

where $\mathbf{J} := \nabla \boldsymbol{f}(\boldsymbol{x}) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ is the Jacobian matrix, and $\|\mathbf{J}\| = \sup_{\boldsymbol{u}, \|\boldsymbol{u}\|=1} \|\mathbf{J}\boldsymbol{u}\|$ is the operator norm of the Jacobian matrix. According to the Min-max principle for singular values [31], the largest singular value $s_{max}(\mathbf{M})$ of a Matrix $\mathbf{M}$ is equal to the operator norm: $s_{max}(\mathbf{M}) = \|\mathbf{M}\|$. Then we have the following proposition.

Proposition 1. (Estimating the Lipschitz constant.) Lipschitz constant L of a function $\boldsymbol{f}$ can be estimated by the largest singular value of its Jacobian $\mathbf{J} = \nabla \boldsymbol{f}(\boldsymbol{x})$:

$$L = \|\mathbf{J}\| = s_{max}(\mathbf{J}). \tag{5}$$

3.3 Lipschitz Constant and Robustness

If the neural network $\boldsymbol{f}$ has a small Lipschitz constant L, then L-Lipschitz continuity implies that the change of network output can be strictly controlled under input perturbations.

Definition 2 (Perturbed example and adversarial example.) Let $\boldsymbol{x} \in \mathbb{R}^n$ be an input vector of a classification function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$, and the predicted class is given as $c(\boldsymbol{x}) = \arg\max_{1 \leq i \leq m} y_i, y_i =$

$f_i(\boldsymbol{x})$. Given $\boldsymbol{x}$, we say $\boldsymbol{x_a} = \boldsymbol{x} + \boldsymbol{\delta}$ is a perturbed example of $\boldsymbol{x}$ with noise $\boldsymbol{\delta} \in \mathbb{R}^n$ under l_p perturbation $\|\boldsymbol{\delta}\|_p = \epsilon$. An adversarial example is a perturbed example $\boldsymbol{x_a}$ that changes the predicted class $c(\boldsymbol{x})$.

Definition 3 (Margin of a prediction.) The margin of a prediction denotes the difference between the largest and second-largest output logits:

$$\text{margin}(\boldsymbol{f}(\boldsymbol{x})) = \max(0, y_t - \max_{i \neq t} y_i) \tag{6}$$

where $\boldsymbol{y} = [y_1, y_2, \cdots] = \boldsymbol{f}(\boldsymbol{x})$ is the predicted logits from the model $\boldsymbol{f}$ on data point $\boldsymbol{x}$. y_t is the correct logit ($\boldsymbol{x}$ belongs to t-th class). We assume that the original prediction is correct: i.e. $y_t = \arg\max_i y_i$. According to the results from Li *et al.* [32], we have the sufficient condition for a data point to be provably robust to perturbation-based adversarial examples:

Theorem 2 (Li [32]) If $2^{\frac{p-1}{p}} \cdot L \cdot \epsilon < \text{margin}(\boldsymbol{f}(\boldsymbol{x}))$, where $\boldsymbol{f}$ is a L-Lipschitz continuous function under l_p norm, then $\boldsymbol{x}$ is robust to any input perturbation $\boldsymbol{\delta}$ with $\|\boldsymbol{\delta}\|_p \leq \epsilon$.

Specifically, we have a proposition for l_2 or l_∞ perturbation by simply letting $p = 2$ or $p = \infty$ in Theorem 2.

Proposition 2. (Certified Robustness of Lipschitz networks.) For a neural network classifier $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ with Lipschitz constant L. Then the neural network classifier is provably robust under l_2 perturbation $\|\boldsymbol{\delta}\|_2 \leq \frac{\sqrt{2}}{2L}\text{margin}(\boldsymbol{f}(\boldsymbol{x}))$ or provably robust under l_∞ perturbation $\|\boldsymbol{\delta}\|_\infty \leq \frac{1}{2L}\text{margin}(\boldsymbol{f}(\boldsymbol{x}))$.

3.4 Maximum Singular Value of Random Matrices

As demonstrated in proposition 2, the Lipschitz constant determines the certified robustness of a neural network classifier unde l_p norm perturbation. Our goal is to derive an analytical expression for the Lipschitz constant based on a given neural network architecture and explore the relationship between the architecture and robustness. According to 3.2, the Lipschitz constant corresponds to the largest singular value of the neural network's Jacobian matrix. Thus, the challenge is to estimate this largest singular value. In this study, we propose an approximation method using Random Matrix Theory (RMT) [33] to estimate the Lipschitz constant. Although this method does not yield the exact value of the Lipschitz constant, the approximation remains valuable for exploring its analytical relationship with neural network architectures.

Theorem 3 (Rudelson [34]). Let $\mathbf{A}$ be an $N \times n$ random matrix whose entries are independent copies of some random variable with zero mean, unit variance, and finite fourth moment. Suppose that the dimensions N and n grow to infinity while the aspect ratio n/N converges to some number $y \in (0, 1]$, $n/N \to y$. Then the maximum singular value $s_{max}(\mathbf{A})$ converges to:

$$\frac{1}{\sqrt{N}} s_{max}(\mathbf{A}) \to 1 + \sqrt{\frac{n}{N}} \quad \text{almost surely.} \tag{7}$$

Thus, asymptotically, the expectation of the maximum singular value is $\mathbb{E}s_{max}(\mathbf{A}) \approx \sqrt{N} + \sqrt{n}$. For the maximum singular value of the product of matrix $\mathbf{A}$ and a scaler α, we have:

$$s_{max}(\alpha\mathbf{A}) = \|\alpha\mathbf{A}\| = \alpha\|\mathbf{A}\| = \alpha s_{max}(\mathbf{A}). \tag{8}$$

Equation (7) provides the maximum singular value for matrices with unit variance. If the variance is α^2, it can be approximated by scaling the maximum singular value of the unit variance matrix by α. Thus, we propose the following expression for the maximum singular value of random matrices with variance α^2.

Proposition 3. (Maximum singular value of a random matrix.) Let $\mathbf{A}$ be an $N \times n$ random matrix whose entries are independent copies of some random variable with zero means, α^2 variance, and finite fourth moment. The expectation of the maximum singular value can be approximated by $\mathbb{E}s_{max}(\mathbf{A}) \approx \alpha(\sqrt{N} + \sqrt{n})$.

3.5 Variance of the Jacobian

Proposition 3 provides an approximation method for estimating the singular value of a random matrix. The Jacobian matrix of a neural network is random at initialization due to the randomly initialized network parameters. Therefore, at least during initialization, we can estimate the Lipschitz constant of a neural network using Propositions 3 and 1. Furthermore, even during training, the network parameters remain nearly random for wide neural networks, based on neural tangent kernel analysis [35]. Thus, in this subsection, we estimate certain statistical properties (e.g., mean and variance) of the Jacobian matrix, which can be used to approximate the Lipschitz constant.

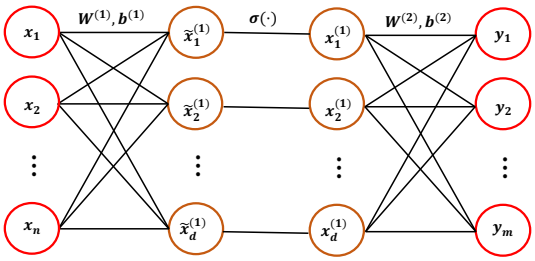


Figure 1: Neural Network Architectures.

As an illustration, we first consider a multilayer perceptron (MLP) with one hidden layer, as shown in fig. 1. The analysis can be extended to MLPs with additional hidden layers due to the application of the chain rule for derivatives. In fig. 1, n represents the input dimension, m is the output dimension, and d is the hidden dimension. The network parameters include weights $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}$, and biases $\boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$. The activation function is denoted as $\sigma(\cdot)$. We initialize these parameters using Xavier initialization [36]:

$$W_{i,j}^{(1)} \sim \mathbb{N}(0, \frac{2\alpha^2}{d+n}), \quad b_i^{(1)} = 0, \; W_{i,j}^{(2)} \sim \mathbb{N}(0, \frac{2\alpha^2}{m+d}), \quad b_i^{(2)} = 0, \tag{9}$$

We can compute the expectation and variance of the Jacobian as follows. Please refer to appendix A for further details.

$$\mathbb{E}[J_{i,j}] = \sum_{k=1}^{d} \mathbb{E}[W_{i,k}^{(2)}]\mathbb{E}[\sigma'(\tilde{x}_k^{(1)})]\mathbb{E}[W_{k,j}^{(1)}] = 0 \tag{10}$$

$$\mathbb{VAR}[J_{i,j}] = \sum_{k=1}^{d} \mathbb{VAR}[W_{i,k}^{(2)}]\mathbb{VAR}[\sigma'(\tilde{x}_k^{(1)})]\mathbb{VAR}[W_{k,j}^{(1)}] = \frac{4d}{(d+n)(d+m)}\alpha^4 q^2 \tag{11}$$

where $q^2 = \mathbb{VAR}[\sigma'(x)]$ denotes the variance of random variables $\sigma'(x)$, when $x \sim \mathbb{N}(0,1)$.For example, with the ReLU activation function, $q^2 = \mathbb{VAR}[\sigma'(x)] = \frac{1}{4}$ when x follows a standard normal distribution. By applying the chain rule of derivatives, this approach can be extended to compute the variance of the Jacobian matrix for any M-layer network, leading to the following proposition.

Proposition 4. (Expectation and Variance of a Jacobian matrix.) Let $\boldsymbol{f}$ denote the M-layer neural network with input dimension n, output dimension m, and hidden dimension d. The Jacobian matrix is denoted as $\mathbf{J} = \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\boldsymbol{x}}$.Assume the neural network is initialized using Xavier initialization. Then, the expectation and variance of the Jacobian can be estimated as:

$$\mathbb{E}[J_{i,j}] = 0, \quad \mathbb{VAR}[J_{i,j}] = \frac{4d}{(d+n)(d+m)}\alpha^{2M} q^{2M-2}, \tag{12}$$

where $q^2 = \mathbb{VAR}[\sigma'(x)]$ denotes the variance of $\sigma'(x)$ for $x \sim \mathbb{N}(0,1)$. For ReLU, $q = \frac{1}{2}$.

3.6 Lipschitz Constant and Robustness of Neural Networks

By combining Proposition 4, Proposition 3, and Proposition 1, we derive an analytical expression for the Lipschitz constant of neural networks

Proposition 5. (Expectation of Lipschitz Constant for neural networks.) Let $\boldsymbol{f} : \mathbb{R}^n \times \mathbb{R}^m$ denote the M-layer feedforward neural network with input dimension n, output dimension m, and hidden dimension d. Let $\sigma(\cdot)$ denote the activation function. Assume the neural network was initialized with Xavier initialization, such that the weight matrix $W^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l-1)}}$ satisfies the statistical properties $\mathbb{E}[W^{(l)}] = 0, \mathbb{VAR}[W^{(l)}] = \frac{2\alpha^2}{n^{(l)}+n^{(l-1)}}$. Then, the expectation of the Lipschitz constant can be estimated as:

$$\mathbb{E}[L] = \frac{2\sqrt{d}}{\sqrt{(d+n)(d+m)}} \alpha^M q^{M-1} (\sqrt{n} + \sqrt{m}), \tag{13}$$

where $q^2 = \mathbb{VAR}[\sigma'(x)]$ denotes the variance of $\sigma'(x) = \frac{\partial \sigma(x)}{\partial x}$ for $x \sim \mathbb{N}(0, 1)$. For ReLU, $q = \frac{1}{2}$.

Proposition 2 illustrates the relationship between robustness and the Lipschitz constant. Using the Lipschitz constant as a bridge between robustness and neural network architecture, and combining Proposition 2 with Proposition 5, we derive the following properties:

- If $\alpha q > 1$, increasing the number of network layers (depth M) results in an increase in the Lipschitz constant, leading to decreased robustness.
- Increasing the number of hidden dimensions (width d) decreases the Lipschitz constant, thereby increasing robustness.
- Increasing the weight variance α results in an increase in the Lipschitz constant, which in turn decreases the robustness of the neural network.

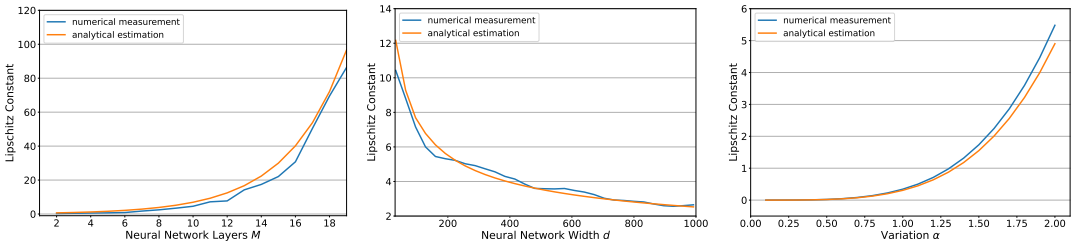
The exact values of the neural network weights change during training, meaning the analytical estimation in eq. (13) may not be accurate after training. The critical point is whether the distribution of the Jacobian matrix, as described in eq. (19), satisfies the conditions of Theorem 3 (singular value of random matrices). Notably, Theorem 3 does not require the matrix's entries to follow a Gaussian distribution; it generally requires that the distribution has a zero third moment and a finite fourth moment. With some modifications, such as normalization $(J_{i,j} - \mu)/\sigma$, the distribution of the Jacobian's elements may still meet these conditions. At the very least, this condition holds true for sufficiently wide neural networks. The Neural Tangent Kernel (NTK) Theory suggests that, in infinitely wide networks, the weights remain nearly constant during training [35].

In this work, we assume that the Jacobian after training continues to satisfy the zero third-moment and finite fourth-moment conditions. We defer a detailed analysis and evaluation of this assumption to future work. Under this assumption, the Lipschitz constant of a neural network after training can still be estimated using eq. (13). The primary difference from the initialization phase is the use of the variance after training, $\tilde{\alpha}$, to replace the initial variance α of weights. Specifically, for a weight matrix $W^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l-1)}}$, we have $\mathbb{VAR}[W^{(l)}_{i,k}(t = 0)] = \frac{2\alpha^2}{n^{(l)}+n^{(l-1)}}$ at initialization. After training for t^* steps, the variance becomes $\mathbb{VAR}[W^{(l)}_{i,k}(t = t^*)] = \frac{2\tilde{\alpha}^2}{n^{(l)}+n^{(l-1)}}$. By measuring $\tilde{\alpha}$,we can then estimate the Lipschitz constant using eq. (13).

4 Experiments

4.1 Evaluating the correctness of Proposition 5

One of the contributions of this work is the proposal of an analytical expression for the Lipschitz constant of neural networks, as given in eq. (13).In this section, we evaluate the accuracy of this analytical expression through toy experiments. We design various neural network architectures with different numbers of layers M, hidden dimensions d, and weight variances α^2. At initialization, we estimate the Lipschitz constant and compare the analytical estimation from eq. (13) with numerical measurements obtained using bound propagation [15]. The numerical method provides an accurate approximation of the Lipschitz constant. To minimize the impact of randomness, we conduct 10 trials for each neural network configuration and report the average results. Additionally, we apply moving average smoothing to filter the curves for better clarity.



(a) L v.s. depth M (b) L v.s. hidden dimension d (c) L v.s. weight variance α

Figure 2: Comparison of estimated Lipschitz constant between analytical estimation eq. (13) and numerical measurement.

Figure 2 presents the experimental results comparing the Lipschitz constant estimates from the proposed analytical expression in eq. (13) and the numerical measurements from previous work [15]. Figure 2a illustrates the Lipschitz constant behavior across different numbers of layers (depth) in neural networks. In this experiment, the width of all networks is set to $d = 256$ and $\alpha = 2$. As shown, the proposed analytical estimation closely matches the numerical measurement. In summary, as the network depth (M) increases, the Lipschitz constant increases exponentially, following the trend L_0^M. Figure 2b shows the Lipschitz constant as a function of network width. For this experiment, we use 4-layer networks with $\alpha = 2$. Again, the analytical estimation aligns closely with the numerical measurement. The results indicate that as the number of hidden dimensions (width d) increases, the Lipschitz constant decreases, following the trend $L_0/\sqrt{d}$. Finally, fig. 2c demonstrates the Lipschitz constant behavior under different weight variances α. The proposed analytical estimation continues to match the numerical measurement. These results validate the correctness of the proposed analytical estimation of the Lipschitz constant for multilayer perceptrons.

4.2 Simple perception experiments

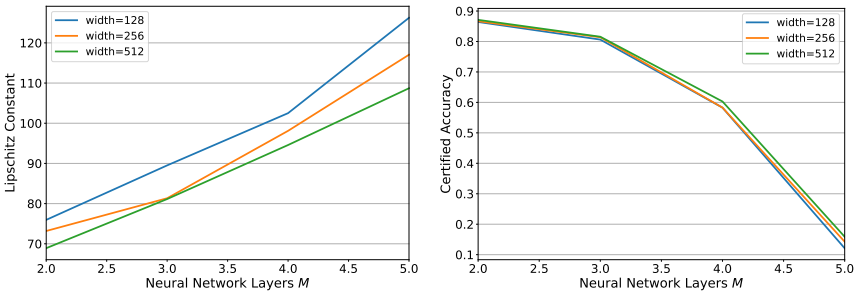
The analytical estimation of the Lipschitz constant from eq. (13) may not be entirely accurate after training. However, it still offers valuable insights into trained networks, particularly the relationship between neural network architecture and robustness. In this section, we conduct a visual perception experiment using the MNIST image classification dataset [37]. In the experiment, we design various neural architectures with different numbers of layers M, hidden dimensions d, and weight variances α^2. We train the model using Stochastic Gradient Descent until converges. We train the models using Stochastic Gradient Descent (SGD) until convergence. After training, we evaluate the test set accuracy and compute the certified accuracy using a verification method. Specifically, for each image, we assess the robustness of the neural network under an $\|\delta\|_2 \leq 1$ perturbation using Interval Bound Propagation (IBP) [38]. We then calculate the average certified accuracy under these perturbations.

Table 1: Standard accuracy, certified accuracy, Lipschitz constant of different network architectures.

Layer	Width	Accuracy	Certified Accuracy	Lipschitz Constant
2	128	0.942	0.864	76.0
2	256	0.943	0.866	73.2
2	512	0.944	0.871	68.9
3	128	0.956	0.806	89.5
3	256	0.957	0.814	81.3
3	512	0.961	0.816	81.1
4	128	0.965	0.582	102.5
4	256	0.968	0.582	98.1
4	512	0.971	0.593	94.5

The experimental results are presented in table 1. While all network architectures achieve similar standard accuracy, their certified accuracy and Lipschitz constant vary significantly. These results

align with the analytical conclusions discussed in section 3.6. 1) As the number of network layers (depth M) increases, the Lipschitz constant increases, resulting in decreased robustness. 2) As the number of hidden dimensions (width d) increases, the Lipschitz constant decreases, leading to increased robustness. Figure 3 visually illustrates these conclusions for clarity.



(a) Lipschitz constant v.s. depth M (b) Certified accuracy v.s. depth M

Figure 3: Lipschitz constant and Certified accuracy of different neural network architectures.

We also conduct experiments to examine the impact of different variances α. Table 2 shows the standard accuracy, certified accuracy, and Lipschitz constant for a 4-layer neural network with a width of $d = 128$ under different initial weight variances α. In the table, accuracy, certified accuracy, and the Lipschitz constant are measured after training. α represents the standard deviation of the weights at initialization. While α may change during training, the initial value provides insight into the Lipschitz constant: a larger initial α results in a larger Lipschitz constant and, consequently, reduced robustness. The evolution of the Lipschitz constant and certified accuracy during training is detailed in appendix B.1.

Table 2: Standard accuracy, certified accuracy, Lipschitz constant under different weight variance.

α	Accuracy	Certified Accuracy	Lipschitz Constant
0.3	0.928	0.720	99.7
1.0	0.965	0.583	102.5
3.0	0.974	0	819.6

5 Conclusions

Ensuring the robustness of neural networks is essential for the safe and reliable operation of robot learning systems in real-world environments. In this work, we investigated the relationship between neural network architecture and robustness. We proposed an analytical expression for estimating the Lipschitz constant of neural networks. This expression allows us to evaluate the Lipschitz constant and investigate its connection to network architecture and robustness. Our analytical expression and experimental results suggest that shallower, wider networks tend to exhibit greater robustness.

Limitations and Future work. 1) This study focuses solely on multilayer perceptrons (MLPs). In future work, we plan to extend our analysis to other network architectures, such as convolutional neural networks (CNNs) and networks with residual connections. 2) The experiments were conducted on the MNIST image classification task. In future research, we aim to test our methods on more complex perception tasks and other robot-learning applications 3) Our current analysis primarily centers on the Lipschitz constant as a key metric for evaluating robustness. However, as noted in Proposition 2, robustness is also affected by the margin of predictions (i.e., the confidence level of the model's outputs). In future studies, we intend to incorporate the margin of predictions into our robustness analysis and expand our experiments to more complex datasets.

References

[1] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, et al. The limits and potentials of deep learning for robotics. *The International journal of robotics research*, 37(4-5):405–420, 2018.

[2] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, et al. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[3] A. Abuduweili, S. Li, and C. Liu. Adaptable human intention and trajectory prediction for human-robot collaboration. *arXiv preprint arXiv:1909.05089*, 2019.

[4] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of field robotics*, 37(3):362–386, 2020.

[5] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[6] R. Liu, R. Chen, A. Abuduweili, and C. Liu. Proactive human-robot co-assembly: Leveraging human intention prediction and robust safe control. In *2023 IEEE Conference on Control Technology and Applications (CCTA)*, pages 339–345. IEEE, 2023.

[7] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[8] M. Balunović and M. Vechev. Adversarial training and provable defenses: Bridging the gap. In *8th International Conference on Learning Representations (ICLR 2020)(virtual)*. International Conference on Learning Representations, 2020.

[9] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017.

[10] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, M. J. Kochenderfer, et al. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404, 2021.

[11] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.

[12] F. Latorre, P. Rolland, and V. Cevher. Lipschitz constant estimation of neural networks via sparse polynomial optimization. *arXiv preprint arXiv:2004.08688*, 2020.

[13] P. L. Bartlett, D. J. Foster, and M. J. Telgarsky. Spectrally-normalized margin bounds for neural networks. *Advances in neural information processing systems*, 30, 2017.

[14] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[15] Z. Shi, Y. Wang, H. Zhang, J. Z. Kolter, and C.-J. Hsieh. Efficiently computing local lipschitz constants of neural networks via bound propagation. *Advances in Neural Information Processing Systems*, 35:2350–2364, 2022.

[16] C. Mueller, J. Venicx, and B. Hayes. Robust robot learning from demonstration and skill repair using conceptual constraints. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6029–6036. IEEE, 2018.

[17] A. Abuduweili and C. Liu. Robust nonlinear adaptation algorithms for multitask prediction networks. *International Journal of Adaptive Control and Signal Processing*, 35(3):314–341, 2021.

[18] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5(1):411–444, 2022.

[19] W. Zhao, Y. Sun, F. Li, R. Chen, R. Liu, T. Wei, and C. Liu. GUARD: A safe reinforcement learning benchmark. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

[20] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4243–4250. IEEE, 2018.

[21] A. Abuduweili and C. Liu. Robust online model adaptation by extended kalman filter with exponential moving average and dynamic multi-epoch strategy. In *Learning for Dynamics and Control*, pages 65–74. PMLR, 2020.

[22] A. Abuduweili and C. Liu. Online model adaptation with feedforward compensation. In *Conference on Robot Learning*, pages 3687–3709. PMLR, 2023.

[23] X. Chen, A. Ghadirzadeh, M. Björkman, and P. Jensfelt. Adversarial feature training for generalizable robotic visuomotor control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1142–1148. IEEE, 2020.

[24] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems*, 31, 2018.

[25] T. Wei, Z. Wang, P. Niu, A. Abuduweili, W. Zhao, C. Hutchison, E. Sample, and C. Liu. Improve certified training with signal-to-noise ratio loss to decrease neuron variance and increase neuron stability. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

[26] L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*, pages 5276–5285. PMLR, 2018.

[27] B. Zhang, D. Jiang, D. He, and L. Wang. Rethinking lipschitz neural networks and certified robustness: A boolean function perspective. In *Advances in Neural Information Processing Systems*, 2022.

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

[29] Q. Nguyen, M. Mondelli, and G. F. Montufar. Tight bounds on the smallest eigenvalue of the neural tangent kernel for deep relu networks. In *International Conference on Machine Learning*, pages 8119–8129. PMLR, 2021.

[30] H. Federer. *Geometric measure theory*. Springer, 2014.

[31] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge university press, 2012.

[32] Q. Li, S. Haque, C. Anil, J. Lucas, R. B. Grosse, and J.-H. Jacobsen. Preventing gradient attenuation in lipschitz constrained convolutional networks. *Advances in neural information processing systems*, 32, 2019.

[33] G. Akemann, J. Baik, and P. Di Francesco. *The Oxford handbook of random matrix theory*. Oxford University Press, 2011.

[34] M. Rudelson and R. Vershynin. Non-asymptotic theory of random matrices: extreme singular values. In *Proceedings of the International Congress of Mathematicians 2010 (ICM 2010) (In 4 Volumes) Vol. I: Plenary Lectures and Ceremonies Vols. II–IV: Invited Lectures*, pages 1576–1602. World Scientific, 2010.

[35] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[36] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[37] Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[38] S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, and P. Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

A More details about the Variance of the Jacobian

As an illustration, we first consider a multilayer perceptron (MLP) with one hidden layer, as shown in fig. 1. The analysis can be extended to MLPs with additional hidden layers due to the application of the chain rule for derivatives. In fig. 1, n represents the input dimension, m is the output dimension, and d is the hidden dimension. The network parameters include weights $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}$, and biases $\boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$. The activation function is denoted as $\sigma(\cdot)$. We initialize these parameters using Xavier initialization [36]:

$$W_{i,j}^{(1)} \sim \mathbb{N}(0, \frac{2\alpha^2}{d+n}), \quad b_i^{(1)} = 0, \tag{14}$$

$$W_{i,j}^{(2)} \sim \mathbb{N}(0, \frac{2\alpha^2}{m+d}), \quad b_i^{(2)} = 0, \tag{15}$$

where α controls the variance of initialization. The scaling of each weight is determined by its corresponding dimensions. The i-th coordinate of the final output is:

$$y_i = \sum_{k=1}^{d} W_{i,k}^{(2)} x_k^{(1)} + b_i^{(2)} \tag{16}$$

$$x_k^{(1)} = \sigma(\tilde{x}_k^{(1)}), \quad \tilde{x}_k^{(1)} = \sum_{j=1}^{n} W_{k,j}^{(1)} x_j + b_k^{(1)} \tag{17}$$

Then the (i,j)-th elements of the Jacobian is $\mathbf{J} = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$:

$$J_{i,j} = \frac{\partial y_i}{\partial x_j} = \sum_{k=1}^{d} \frac{\partial y_i}{\partial x_k^{(1)}} \frac{\partial x_k^{(1)}}{\partial \tilde{x}_k^{(1)}} \frac{\partial \tilde{x}_k^{(1)}}{\partial x_j} \tag{18}$$

$$= \sum_{k=1}^{d} W_{i,k}^{(2)} \sigma'(\tilde{x}_k^{(1)}) W_{k,j}^{(1)} \tag{19}$$

where $\sigma'(\tilde{x}_k^{(1)}) := \frac{\partial \sigma(\tilde{x}_k^{(1)})}{\partial \tilde{x}_k^{(1)}}$. Note that, $W_{i,k}^{(2)}$, $\sigma'(\tilde{x}_k^{(1)})$, and $W_{k,j}^{(1)}$ are independent. At initialization, we have:

$$\mathbb{E}[W_{i,k}^{(2)}] = \mathbb{E}[W_{i,k}^{(1)}] = 0 \tag{20}$$

$$\mathbb{VAR}[W_{i,k}^{(2)}] = \frac{2\alpha^2}{d+n}, \quad \mathbb{VAR}[W_{k,j}^{(1)}] = \frac{2\alpha^2}{d+m} \tag{21}$$

Next, we compute the expectation and variance of the Jacobian:

$$\mathbb{E}[J_{i,j}] = \sum_{k=1}^{d} \mathbb{E}[W_{i,k}^{(2)}] \mathbb{E}[\sigma'(\tilde{x}_k^{(1)})] \mathbb{E}[W_{k,j}^{(1)}] = 0 \tag{22}$$

$$\mathbb{VAR}[J_{i,j}] = \sum_{k=1}^{d} \mathbb{VAR}[W_{i,k}^{(2)}] \mathbb{VAR}[\sigma'(\tilde{x}_k^{(1)})] \mathbb{VAR}[W_{k,j}^{(1)}] \tag{23}$$

$$= \frac{2\alpha^2}{d+n} \cdot \frac{2\alpha^2}{d+m} \cdot \mathbb{VAR}[\sigma'(\tilde{x}_k^{(1)})] \tag{24}$$

$$= \frac{4d}{(d+n)(d+m)} \alpha^4 q^2 \tag{25}$$

where $q^2 = \mathbb{VAR}[\sigma'(x)]$ denotes the variance of random variables $\sigma'(x)$, when $x \sim \mathbb{N}(0,1)$. For example, with the ReLU activation function:

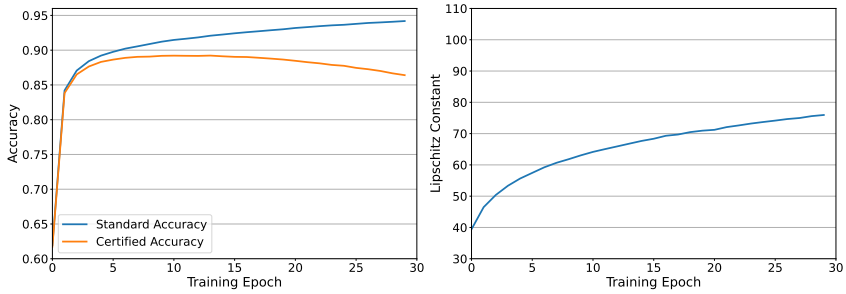
$$\sigma(x) = \max(0, x), \sigma'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{26}$$

Thus, $q^2 = \mathbb{VAR}[\sigma'(x)] = \frac{1}{4}$ when x follows a standard normal distribution.

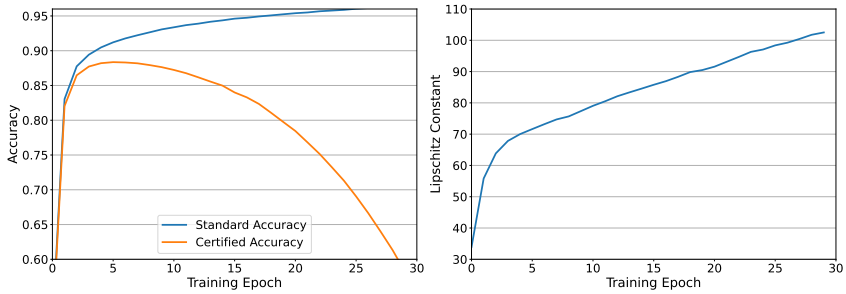
B Additional Experiments

B.1 Lipschitz Constant during Training

Figure 4 shows the standard accuracy, certified accuracy, and Lipschitz constant during training for a 2-layer neural network and a 4-layer neural network. As observed, the Lipschitz constant increases over the course of training. The certified accuracy initially increases in the early stages of training but decreases in the later stages. This decrease in certified accuracy may be attributed to the increase in the Lipschitz constant. Furthermore, fig. 4 indicates that the 2-layer network maintains a smaller Lipschitz constant and higher certified accuracy compared to the 4-layer network.



(a) 2-layer neural network



(b) 4-layer neural network

Figure 4: Lipschitz constant and accuracy during training .

B.2 Weight Decay Increases the Robustness of Training

Weight decay is a common regularization technique used in neural network training. Let $\boldsymbol{f}(\mathbf{W}; \boldsymbol{x})$ represent the output of the neural network for an input $\boldsymbol{x}$ given the learnable parameter $\mathbf{W}$. Assume the task-dependent objective function (e.g. cross-entropy) is $\mathcal{L}(f(\mathbf{W}; \boldsymbol{x}), \boldsymbol{y}^\star)$, where $\boldsymbol{y}^\star$ is a ground-truth labels. Then the training objective with weight decay can be formulated as:

$$\min_{\mathbf{W}} \mathcal{L}(f(\mathbf{W}; \boldsymbol{x}), \boldsymbol{y}^\star) + \lambda \|\mathbf{W}\|, \tag{27}$$

where λ is a penalty term that controls the weight decay.

We demonstrate that weight decay can help bound the Lipschitz constant during training. For simplicity, we consider a single-layer neural network with either ReLU or Sigmoid activation functions. The network output can be expressed as: $\boldsymbol{f(W; x)} = \sigma(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$, we have:

$$\begin{aligned} \|\boldsymbol{f(W, x_1)} - \boldsymbol{f(W, x_1)}\| &= \|\sigma(\mathbf{W}\boldsymbol{x_1} + \boldsymbol{b}) - \sigma(\mathbf{W}\boldsymbol{x_2} + \boldsymbol{b})\| && (28) \\ &\leq \|\mathbf{W}\boldsymbol{x_1} + \boldsymbol{b} - \mathbf{W}\boldsymbol{x_2} + \boldsymbol{b}\| = \|\mathbf{W}\boldsymbol{x_1} - \mathbf{W}\boldsymbol{x_2}\| && (29) \\ &\leq \|\mathbf{W}\|\|\boldsymbol{x_1} - \boldsymbol{x_2}\| && (30) \end{aligned}$$

The Lipschitz constant L denotes the smallest value for: $\|\boldsymbol{f(x_1)} - \boldsymbol{f(x_2)}\| \leq L\|\boldsymbol{x_1} - \boldsymbol{x_2}\|$. We have $L \leq \|\mathbf{W}\|$. For M-layer's network, we have a similar property:

$$L \leq \prod_{l=1}^{M} \|\mathbf{W}^{(l)}\| \tag{31}$$

Thus, weight decay serves to minimize the upper bound of the Lipschitz constant. Table 3 presents the experimental results for certified accuracy and the Lipschitz constant under different weight decay values (varying λ in eq. (27)). As shown, a larger penalty term λ results in a smaller Lipschitz constant and, consequently, higher robustness.

Table 3: Standard accuracy, certified accuracy, Lipschitz constant with weight decay.

λ	Accuracy	Certified Accuracy	Lipschitz Constant
0	0.965	0.583	102.6
0.0001	0.964	0.594	100.9
0.001	0.962	0.675	87.0
0.01	0.940	0.841	40.1