Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра математики

Магістерська робота

освітній ступінь – магістр

на тему:

“IMPACT OF ADVERSARIAL SPARSITY AS AN AUXILIARY METRIC IN ADVERSARIAL ROBUSTNESS”

Виконав: студент 2-го року навчання

Освітньо-наукової програми

“Прикладна математика”, 113

Кузьменко Дмитро Олександрович

Керівник Швай Н. О.

кандидат фіз.-мат. наук

Рецензент_____

Магістерська робота захищена

з оцінкою _____

Секретар ЕК _____

«____» _____ 2023 р.

Київ – 2023

TABLE OF CONTENTS

TABLE OF CONTENTS **2**
1 INTRODUCTION **4**
2 PRELIMINARIES **7**
2.1 Robustness and astuteness 7
2.2 Local Lipschitzness 7
2.3 r-separability 8
2.4 L-ball and Epsilon Parameter 8
3 ADVERSARIAL ROBUSTNESS **9**
3.1 Adversarial Attacks 9
3.1.1 White-Box & Black-Box Attacks 9
3.1.2 Targeted & Untargeted Attacks 10
3.1.5 Logit-Based & Hard-Label-Based Attacks 10
3.2 Adversarial Defenses 10
3.2.1 Adversarial Training (AT) 10
3.2.2 Locally-Linear Regularization (LLR) 11
3.2.3 Robust self-training (RST) 12
3.2.4 TRADES 13
3.2.5 Gradient Regularization 13
3.2.6 Input Transformation and Augmentation 14
3.2.7 Adversarial Weight Perturbation 14
3.3 Model Architectures and Their Impact on Robustness 14
3.3.1 Convolutional Neural Networks (CNNs) 14
3.3.2 Vision Transformers 15
3.3.3 Diffusion Models 16
3.4 Adversarial Sparsity 17
3.4.1 Prior definitions 17
3.4.2 Adversarial set and adversarial accuracy 17
3.4.3 Sparsity 18
3.4.4 Sparsity and number of perturbations 18
3.4.5 Constrained PGD 19
3.4.6 Computing sparsity 20

4 EXPERIMENTS **21**
4.1 Dataset and threat-model selection 21
Data subsampling 21
4.2 Experiments flow 21
4.3 Sparsity algorithms 23
4.3.1 L2 sparsity 23
4.3.2 L∞ Sparsity re-implementation. 24
4.3.3 Proposed approach: L∞ sparsity with hybrid search with n-Ary and binary counterparts 25
4.4 Initial contribution 27
5 RESULTS **28**
5.1 The key takeaways of the experiments 28
5.2 Impact of number of directions 31
5.3 Sparsity estimation time and directions count analysis 32
6 CONCLUSIONS **34**
REFERENCES **36**
APPENDIX 1 43

1 INTRODUCTION

The challenge of securing machine learning models against adversarial attacks is a critical area of research, particularly in the realm of computer vision. With the increasing usage of these models in various real-world scenarios, ensuring robustness to adversarial perturbations is paramount. Previous works have researched different the paradigms and mechanisms of adversarial robustness and resistance to adversarial attacks [1-3]. Researchers studied multiple variants and enhancements of adversarial attacks to create more elaborate and imperceivable perturbations [4-8] as well durable and robust adversarial defenses [9-11] to combat and successfully mitigate perturbed samples. A separate attention of research community was focused on establishing robustness-accuracy trade-off [12-13] and tending to overestimated robustness of numerous defenses due to the gradient obfuscating phenomenon [14].

Olivier and Raj in their recent work [15] propose to constrain the attack to only look in a subset of the set of admissible perturbations Δ. They try to answer the question of how large such subset must be to find an adversarial perturbation in it. They quantify the expected size of the subset: the larger the size, the fewer perturbations there are. They denote this metric adversarial sparsity - a devoted focus of our work.

The purpose of this research is to investigate adversarial sparsity in computer vision models and introduce a more efficient method for adversarial sparsity estimation. To fulfil this objective, the following tasks have been undertaken:

1. To implement and evaluate an n-Ary search algorithm as an improvement over the conventional binary search method used in adversarial sparsity estimation.
2. To benchmark and compare the performance of the proposed n-Ary search algorithm against the traditional binary search algorithm.
3. To explore the implications of adversarial sparsity on the robustness of machine learning models.

The object of our study is adversarial sparsity, specifically L_∞ sparsity, in machine learning models designed for computer vision tasks. The research primarily

employs the methodology of adversarial attack simulations and sparsity estimation, combined with a comparative analysis of different search algorithms. We utilize robust computer vision models, tested against a range of adversarial attacks, and evaluate their performance using the L_∞ sparsity measure. We used RobustBench [16] repository of trained models for our experiments.

The novelty of this research is in the introduction and evaluation of an improved n-Ary search algorithm for adversarial sparsity estimation. The algorithm shows improved computational efficiency while maintaining the accuracy of the sparsity estimates. Furthermore, we present novel findings on the effect of adversarial sparsity on the robustness of machine learning models.

The practical significance of this research is twofold. First, it provides a more efficient method for adversarial sparsity estimation, which can benefit other researchers and practitioners in the field. Second, the insights on the relationship between adversarial sparsity and model robustness can potentially guide the development of more robust machine learning models, thus enhancing their performance in real-world scenarios subject to adversarial threats.

This research work is divided into comprehensive sections, each detailing a significant aspect of adversarial robustness and sparsity in machine learning models used in computer vision.

Section 2 is devoted to a thorough exploration of adversarial robustness in the context of machine learning. It delves into the concepts of Lipschitz continuity, r-separability of datasets [3], and the robustness-accuracy trade-off [12-13]. Additionally, this section covers several mathematical preliminaries crucial to understanding the nuances of adversarial attacks and their implications for model robustness.

Section 3 of this work focuses on adversarial attacks and defenses [4-8]. It elaborates on the mechanisms underlying these attacks, their objectives, and the defenses employed against them. Moreover, it introduces the metrics used to evaluate

the efficacy of these defenses, specifically natural accuracy, robust accuracy, and adversarial sparsity – the principal metric of our study.

Section 4 describes the experiments conducted to explore adversarial sparsity. This chapter outlines the reimplementation of an existing adversarial sparsity code solution [34], leveraging RobustBench and timm models to validate the algorithm. It further details the choice of employing these techniques in Computer Vision tasks, specifically on the CIFAR-10 dataset with L_∞ sparsity and an epsilon value of 8/255.

Section 5 describes the results of this work. The conclusions are outlined in section 6.

Through this structured approach, our research aims to provide a comprehensive exploration of adversarial sparsity and its implications for the robustness of machine learning models. We anticipate that our findings will contribute to the ongoing effort to enhance the security and performance of these models in adversarial environments.

2 PRELIMINARIES

Understanding adversarial robustness, the ability of a machine learning model to resist adversarial attacks, is a crucial aspect of model security in the field of machine learning.

2.1 Robustness and astuteness

Let $X \subseteq R^d$ be an instance space equipped with a robustness-measuring distance: $X \times X \to R^+$. Let $[C] = \{1, 2, \ldots, C\}$ be the set of available labels with $C \geq 2$. For a function $f : X \to \mathrm{R}^C$, let $f(x)_i$ denote the coordinate i.

Let $B(x, \varepsilon)$ denote a ball of radius $\varepsilon > 0$ around x in a metric space. B_∞ denotes the L_∞ ball. A classifier g is ***robust*** at x with radius $\varepsilon > 0$ if for all $x' \in \mathrm{B}(x, \varepsilon)$, holds $g(x') = g(x)$. Additionally, g is ***astute*** at (x, y) if $g(x') = y$ for all $x' \in \mathrm{B}(x, \varepsilon)$. The astuteness of g at radius $\varepsilon > 0$ under a distribution μ is

$$\Pr_{(x,y)\sim\mu}[g(x') = y \text{ for all } x' \in \mathrm{B}(x, \varepsilon)].$$

Finding the most astute g defines the task of robust classification [36]. Natural accuracy describes a standard (without perturbations) accuracy.

2.2 Local Lipschitzness

Definition 1. Let $(X, dist)$ be a metric space. A function $f : X \to R^C$ is *L-locally Lipschitz* at radius r if for each $i \in [C]$, we have $|f(x)_i - f(x')_i| \leq L \cdot dist(x, x')$ for all x' with $dist(x, x') \leq r$.

Definition 2. Separated data distributions are formally defined as follows. Let X contain C disjoint classes $X^{(1)}, \ldots, X^{(C)}$, where all points in $\mathrm{X}^{(i)}$ have label i for $i \in [C]$.

2.3 r-separability

The *r*-separability of a dataset is a key concept in understanding adversarial robustness. It is a measure of how easily the data points from different classes can be separated by a margin of radius *r* in the input space. High r-separability means that data points from different classes can be distinctly identified, making it difficult for an adversary to cause misclassification through small perturbations.

Definition 3. We say that a data distribution over $\cup_{i\in[C]}X^{(i)}$ is r-separated if $dist(X^{(i)}, X^{(j)}) \geq 2r$ for all $i \neq j$, where $dist(X^{(i)}, X^{(j)}) = \min_{x\in X(i),x'\in X(j)} dist(x, x')$.

The separation between any two instances belonging to distinct classes is never less than 2*r*. In many real-life classification activities, classes are clearly divided. For instance, when the L_∞ norm is used as the distance measure, images of differing classes (such as a boat, a house, a person, and so on) will exhibit *r*-separation. The value of *r*, greater than 0, depends on the specific characteristics of the image space.

2.4 L-ball and Epsilon Parameter

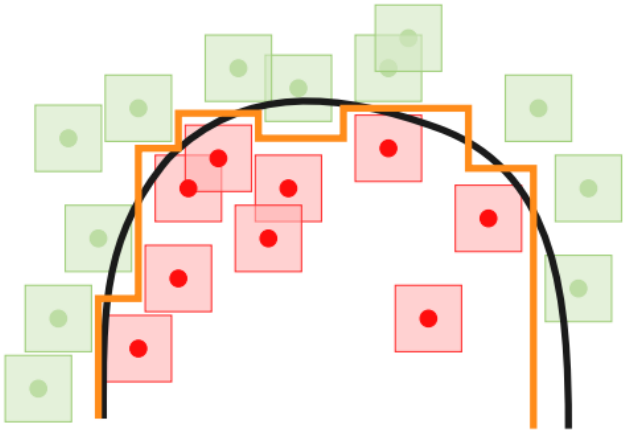

Figure 1. The classifier represented by the orange boundary exhibits low local Lipschitzness as it remains constant within the L∞ spheres surrounding the data points.

On the other hand, the classifier associated with the black curve, while possessing high accuracy on unaltered data, is susceptible to adversarial examples [3]

3 ADVERSARIAL ROBUSTNESS

Understanding adversarial attacks and defenses, as well as the effect of different model architectures on robustness, is essential in the realm of adversarial robustness. This section aims to provide an exhaustive background on the existing adversarial attacks, defenses, and the impact of various model structures on robustness.

3.1 Adversarial Attacks

Since the ground-breaking discovery in [37], a range of adversarial attacks have emerged. These can be broadly classified into two categories: white-box and black-box attacks. White-box attacks, as detailed in studies by Gu et al. [38], Carlini and Wagner [4], and Gowal et al. [39], are granted access to the model, allowing them to iteratively perturb the input to maximize the loss value.

On the other hand, black-box attacks, where the attackers do not have access to gradient information, typically launch attacks by transferring adversarial examples as demonstrated by Tramèr et al. [40], or by extensive queries as shown in studies by Andriushchenko et al. [7], and Chen et al. [8].

3.1.1 White-Box & Black-Box Attacks

White-box attacks assume complete access to the target model, including its architecture, parameters, and training data. These attacks often employ gradient-based methods to compute adversarial perturbations. Notable examples include the Fast Gradient Sign Method (FGSM) [1], the Basic Iterative Method (BIM) [17], and the Projected Gradient Descent (PGD) [2].

In contrast, black-box attacks [18] do not have direct access to the model's internals. They might only have access to the model's input-output pairs, and they

generate adversarial examples based on this limited information. Techniques such as substitute model training and query-based attacks are typically used in black-box settings.

3.1.2 Targeted & Untargeted Attacks

In a targeted adversarial attack, the attacker aims to manipulate the model into misclassifying an input as a specific incorrect class. Conversely, untargeted attacks aim to cause any misclassification, irrespective of the output class.

3.1.5 Logit-Based & Hard-Label-Based Attacks

Logit-based attacks, such as the Carlini & Wagner (C&W) attack [4], modify the inputs based on the model's logit outputs (the outputs before the final activation function). C&W finds the smallest perturbation that can change the model's prediction to a specific target class.

Hard-label-based attacks, on the other hand, assume access only to the model's final predictions. The Boundary attack [19] is a notable example that starts with an adversarial example and gradually reduces the perturbation while keeping the example adversarial.

3.2 Adversarial Defenses

Adversarial defenses aim to improve the model's robustness against adversarial attacks. Various strategies exist, including adversarial training, regularization techniques, input transformation, weight perturbations, etc.

3.2.1 Adversarial Training (AT)

Adversarial Training is a widely used defense strategy that involves augmenting the training set with adversarial examples generated using a specific attack algorithm. During training, the model is exposed to both clean examples and adversarial examples

and is trained to minimize the loss on both types of examples. By incorporating adversarial examples into the training process, Adversarial Training helps to improve the model's robustness and its ability to generalize to unseen adversarial samples. It encourages the model to learn more robust and discriminative features that are resilient to adversarial perturbations. Adversarial Training has shown to be effective in enhancing the robustness of models against a wide range of adversarial attacks.

3.2.2 Locally-Linear Regularization (LLR)

Locally-Linear Regularization [20] is a defense technique used to improve the robustness of neural networks against adversarial attacks. It introduces a regularization term to the loss function during training, which encourages the model to have locally linear behavior around each training example. The motivation behind LLR is that adversarial examples often lie in regions of the input space where the model's behavior is nonlinear. By promoting locally linear behavior, LLR aims to make the model more robust to adversarial perturbations.

Proposition. Consider a loss function $l(x)$ that is once-differentiable, and a local neighbourhood defined by $B(\varepsilon)$. Then $\forall \delta \in B(\varepsilon)$, $|l(x + \delta) - l(x)| \leq |\delta^T \nabla_x l(x)| + \gamma(\varepsilon, x)$. Adversarial loss tends to $l(x)$, i.e., $l(x + \delta) \rightarrow l(x)$.

Following the analysis above, authors propose the following objective for adversarially robust training:

$$L(D) = E_D[l(x) + \underline{\boldsymbol{\lambda\gamma(\varepsilon, x)} + \mu|\delta^T_{LLR}\nabla_x l(x)|}],$$

where the underlined part is the LLR, λ and μ are hyper-parameters to be optimized, and $\delta_{LLR} = \text{argmax}_{\delta \in B(\varepsilon)} g(\delta; x)$ (recall the definition of $g(\delta; x)$ from Eq (5), $\delta \in B(\varepsilon)$ is an adversarial perturbation, and $g(\delta; x) = |l(x + \delta) - l(x) - \delta^T \nabla_x l(x)|$ - an indicator of how linear the surface is.

3.2.3 Robust self-training (RST)

Robust Self-Training [11] is a defense strategy that leverages the concept of self-training to improve the robustness of a model. In self-training, the model is initially trained on the original training data and then used to generate pseudo-labeled data from unlabeled examples.

RST extends this approach by incorporating adversarial examples into the self-training process. It generates adversarial examples based on the current model's predictions on unlabeled data and uses these adversarial examples, along with their pseudo-labels, to further train the model. By iteratively refining the model using both clean and adversarial examples, RST aims to enhance the model's robustness.

Robust Self-Training (RST) is a semi-supervised learning technique that aims to improve the performance of models on adversarial examples by leveraging unlabeled data.

The procedure generally involves two main steps:

I. Training a robust teacher model - This step involves training a model, referred to as the teacher, on labeled data (X, Y) using robust optimization techniques to resist adversarial examples. If F represents the function associated with the teacher model, this can be expressed mathematically as:

$$\min_F \max_{\{||\delta|| \leq \epsilon\}} L(F(X+\delta),\ Y)$$

L represents the loss function, δ denotes the adversarial perturbations, and ε is the allowable perturbation magnitude.

II. Generating pseudolabels and training a student model - The robust teacher model is then used to predict labels (*pseudolabels*) for the unlabeled data U. A new model, referred to as the student, is trained on the union of the original labeled data and the newly created (U, *pseudolabels*) data:

$$\min_G \max_{\{||\delta|| \leq \epsilon\}} L(G(X+\delta),\ Y) \cup L(G(U+\delta),\ pseudolabels)$$

Here, G represents the function associated with the student model.

By iterating these two steps, RST can significantly improve the model's robustness against adversarial attacks while making effective use of unlabeled data. It is noteworthy that the selection of pseudolabels requires careful consideration to avoid reinforcing incorrect predictions.

3.2.4 TRADES

TRADES [13] is a defense method designed to strike a trade-off between natural accuracy and robust accuracy. It achieves this trade-off by minimizing a surrogate loss function that combines the cross-entropy loss on clean examples and the robust loss on adversarial examples.

The robust loss is typically measured using the KL-divergence between the model's predictions on clean and adversarial examples. By jointly optimizing the surrogate loss, TRADES aims to improve the model's robust accuracy while maintaining a reasonable level of natural accuracy. It is known for its effectiveness in improving robustness against various adversarial attacks.

3.2.5 Gradient Regularization

Gradient Regularization (GR) [21] is a defense technique that introduces an additional regularization term to the loss function during training to enhance the model's robustness against adversarial attacks. It aims to minimize the sensitivity of the model's output to small perturbations in the input space by penalizing large gradients. The regularization term is usually formulated as the L2 norm or L1 norm of the gradients of the model's output with respect to the input. By discouraging large gradients, Gradient Regularization helps to mitigate the impact of adversarial perturbations and improve the model's robustness.

3.2.6 Input Transformation and Augmentation

Input transformation and augmentation defenses [23, 24] preprocess the inputs or augment the training data to reduce the effect of adversarial perturbations. Techniques include Gaussian noise injection, image quilting, JPEG compression [26], and adversarial patch augmentation.

3.2.7 Adversarial Weight Perturbation

The Adversarial Weight Perturbation (AWP) [25] addresses the issue of non-flat weight loss landscapes in deep neural networks. The main idea behind AWP is to introduce worst-case weight perturbations into the model during training to improve both training robustness and the robust generalization gap.

By introducing worst-case weight perturbations, AWP encourages the model to explore and learn more robust features that are less sensitive to small input perturbations. This results in improved training robustness, allowing the model to better withstand adversarial attacks.

3.3 Model Architectures and Their Impact on Robustness

Different model architectures have varying levels of inherent robustness against adversarial attacks. This section will discuss two influential types of models: Convolutional Neural Networks (CNNs) and Transformer models, and a novel class of generative models, diffusion models, and their effects on robustness.

3.3.1 Convolutional Neural Networks (CNNs)

CNNs [27] are primarily used in image recognition tasks and have been a popular target for adversarial attacks due to their widespread use. They exhibit varying degrees of robustness depending on their depth, width, and specific architectural choices. In general, models with larger capacity, such as deeper or wider CNNs, can achieve higher robustness when trained with proper regularization.

Network	Valid Acc. Against (%)	
	Natural	PGD-20
WRN32-10 w/o dilation	87.25	45.84
ResNet-18 + NADAR	81.35	50.92
ResNet-34 + NADAR	83.57	52.64
ResNet-50 + NADAR	83.23	52.89
ResNet-101 + NADAR	84.39	**53.89**
WRN32-10 + NADAR	**86.23**	53.43

Table 1. Neural architecture dilation for adversarial robustness (NADAR) [41], with various backbones.

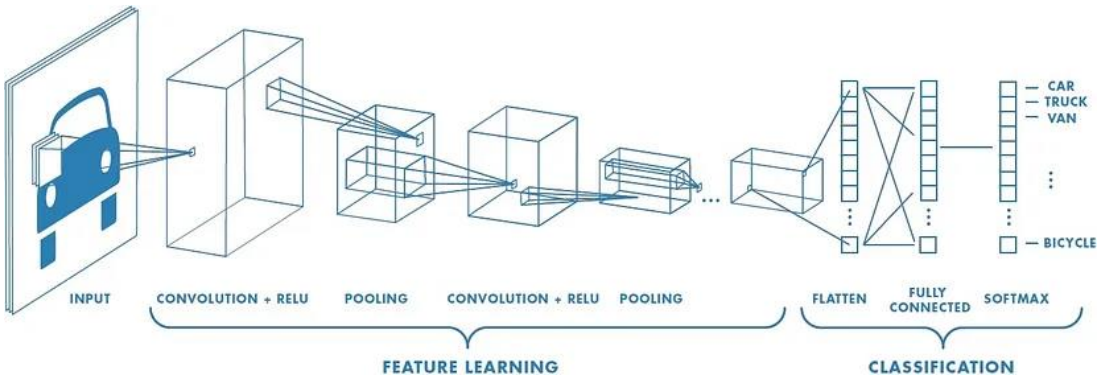

Figure 2. Basic architecture of CNN [42].

3.3.2 Vision Transformers

Vision Transformers (ViT) [28], depicted in figure 3, are a class of models in machine learning that apply the transformer architecture, originally designed for natural language processing tasks, to computer vision tasks.

The key innovation of Vision Transformers is their ability to model global dependencies within the image, which is something that traditional convolutional neural networks (CNNs) struggle with due to their local receptive fields. The performance of ViTs has been shown to be competitive with state-of-the-art CNNs on several benchmark datasets, which has generated significant interest in the research community. ViTs are actively researched in the community of adversarial robustness.

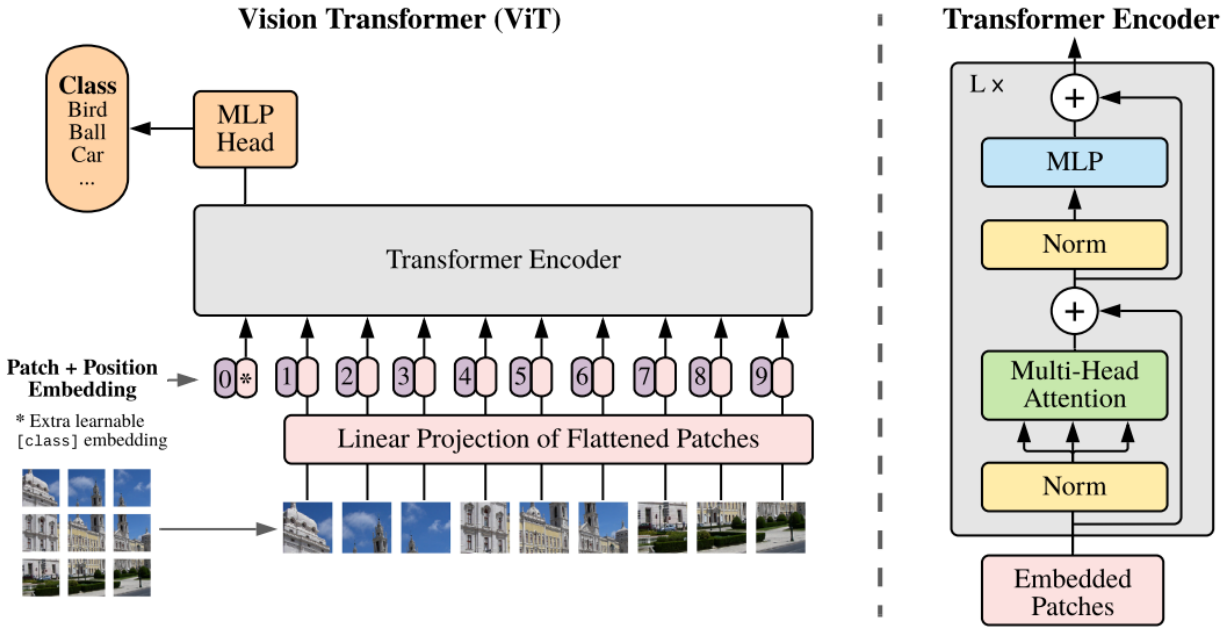

Figure 3. The high-level overview of Vision Transformer: Patch and Positional embeddings combined with linear projection (left); encoder part that incorporates multi-head attention (right) [28].

3.3.3 Diffusion Models

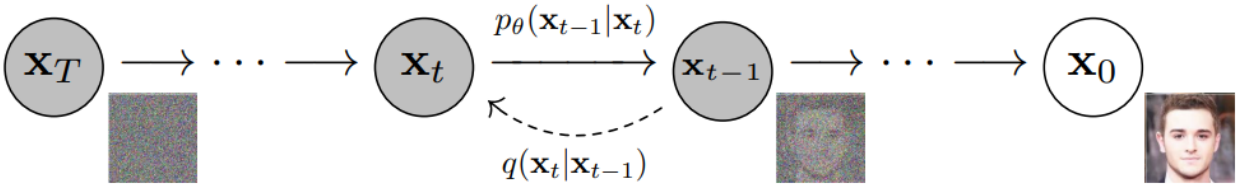

Figure 4: The directed graphical model of denoising diffusion probabilistic model [29].

Diffusion models [29] (figure 4) are a new class of generative models that can simulate the data distribution by a process of noisy diffusion. These models have shown promise in enhancing model robustness as they can generate diverse and realistic samples, potentially improving the model's understanding of the data distribution [32].

3.4 Adversarial Sparsity

3.4.1 Prior definitions

Throughout the following sections *f* designates a machine learning model, and *L* its loss function. ε is the radius of the attack, and Δ is the set of admissible perturbations. Usually $\Delta = B_k^n$ with B_k^n the n-dimensional hyperball in norm L_k:

$$B_\infty^n = [0,1]^n$$

$$B_k^n = \{x \in R^n | x_1^2 + \ldots + x_n^2 \leq 1\},$$

where x_k is the k^{th} coordinate of *x*.

3.4.2 Adversarial set and adversarial accuracy

Authors compute sparsity is a simplified manner by only considering adversarial examples that use all of their perturbation limit. For $k < \infty$ $\Delta = S_k^n$, using the hypersphere ($\|x\| = 1$) rather than the hyperball ($\|x\| \leq 1$). For $k = \infty$, $\Delta = \{\pm 1\}^n$, i.e. only the finite set of vertices of the hypercube is considered. This is achieved without significantly limiting generality because potent attacks like PGD utilize their full perturbation limit (and the simpler FGSM attack is limited to do the same). Given a point *x* and a radius ε, we define the adversarial set as the set of successful admissible perturbations:

$$Adv(f, x, \varepsilon) := \{\delta \in \Delta \,/\, (x + \varepsilon * \delta) \neq f(x)\}$$

Adversarial perturbations are always unit vectors to be scaled by factor ε when applied. Adversarial accuracy over a point x is then defined as follows:

$$AA(f, x, \varepsilon) := 1[Adv(f, x, \varepsilon) = \emptyset]$$

The points are sampled from probability distributions. *U(X)* designates the uniform distribution over measurable set *X*.

3.4.3 Sparsity

Adversarial sparsity is a metric that gauges the usual size required for a subset of Δ to include an adversarial perturbation. More formally, we consider a series of expanding subsets of Δ: with $m_1 < m_2$, $\emptyset \subseteq \Delta^{m1} \subseteq \Delta^{m2} \subseteq \Delta$. We can define adversarial sparsity relative to Δ^m as:

$$AS(f, x, \varepsilon, \Delta^m) := inf\{m, \Delta^m \cap Adv(f, x, \varepsilon) \neq \emptyset\}$$

If a distribution *D* of such sequences is provided, we can define adversarial sparsity as the expected value of AS:

$$AS_{full}(f, x, \varepsilon) := E_{(\Delta^m)\sim D}[AS(f, x, \varepsilon, \Delta^m)].$$

In a broader sense, greater adversarial sparsity implies more robust models, under the condition that certain plausible constraints are applied to the distribution *D*. Specifically, the distribution should be isotropic, meaning it does not favor any specific direction.

We'll provide more specific definitions related to the threat model used in this study below. For L_∞ perturbations, we sample both a vertex $u \in U(\{\pm 1\}^n)$ and a permutation of dimensions $\sigma \in S_n$. For $m \in \{1, ..., n\}$ we define Δ^m as the set of perturbations differing of *u* only over dimensions $\sigma(1), ..., \sigma(m)$:

$$\Delta^m := \{\delta \in S_\infty^n \ / \forall k > m \ \delta_{\sigma(k)} = u_{\sigma(k)}\}$$

In other words Δ^m is the set of perturbations differing from *u* by at most *m* specific pixels $\sigma(1), ..., \sigma(m)$. The permutation σ is required to enforce that all dimensions are equally probable under the distribution.

3.4.4 Sparsity and number of perturbations

If *Adv(f, x, ε)* represents "one half" of Δ (e.g., all perturbations where the perturbation of the bottom left pixel is positive), then its sparsity will be zero for half the directions, but significantly larger for many others. This leads to an expected sparsity that doesn't accurately reflect the vastness of this adversarial set.

Consider a simplified scenario where the set of perturbations is finite: *Adv(f, x, ε)* equals $\{\delta_1, ..., \delta_k\}$. If δ_k is presumed to be uniformly sampled over the permissible set, then there are no favored directions with high adversarial concentration, unlike the previous example. In such a case, a mathematical relationship between sparsity and *k* can be established. For L_∞ sparsity, the expected value of *AS(f, x, ε)* when sampling δ_j is:

$$E[AS(f,x,\varepsilon)] = \sum_{m=0}^{n} (1-2^{-m})^k$$

$$\frac{n-log_2^k}{4} \leq E[AS(f,x,\varepsilon)] \leq n - log_2^k + \frac{e}{e-1},$$

Sparsity tends to vary like $n - log_2^k$. In the general case *Adv(f, x, ε)* is infinite (even though AS(f, x, ε) is finite in L_∞) and we are interested in its volume relative to Δ, rather than its cardinal. Both situations can be linked if considering

$$AS(f,x,\varepsilon) = \cup_{j=1}^{k} Sc(\delta_j, \beta)$$

In other words, *Adv(f, x, ε)* constitutes the union of spherical caps with the same radius δ. The term "local adversarial radius" could be used to describe β, which is the extent to which a typical adversarial perturbation must be modified to return to the original input. Under this scenario, the volume of *Adv(f, x, ε)* equates to *k * V(Sc(u, β))* (assuming a disjoint union), and the divergence of *AS(f, x, ε)* from the finite case would be at most β. This implies that sparsity is influenced by both *k* and β. Thus, comparing the sparsity of two models is similar to comparing the size of their adversarial sets, provided these sets have comparable local radii.

3.4.5 Constrained PGD

For L_∞ attacks this is straightforward: given a vertex *u*, a perturbation σ and a number of dimensions *m*, we append after projection an additional step enforcing the constraints:

$$\forall k > m \; \delta_{\sigma(k)} \leftarrow u_{\sigma(k)}$$

Such a constrained attack shall be denoted PGD_m in this work.

3.4.6 Computing sparsity

In order to estimate sparsity for L_∞ attacks, we employ binary search to examine potential values of *m*, given a direction *u* and direction permutation σ, and execute the constrained attack at each step. We calculate the average over multiple sampled directions to estimate the full adversarial sparsity (AS_{full}), with 100 samples as suggested by the authors.

Adversarial sparsity is only applicable when adversarial perturbations exist near an input *x* and is designed to quantify the degree of susceptibility in models lacking robustness. When assessing a model that exhibits robustness around *x* (where $Adv(f, x, \varepsilon)$ is an empty set), we default it to a maximum value of dimensions *n* for L_∞ attacks.

4 EXPERIMENTS

4.1 Dataset and threat-model selection

The dataset choice for the experiments was **CIFAR-10** with $\mathbf{L}_\infty$ distance and **eps of 8/255**. While it is the most relevant and consistent benchmark in adversarial robustness, it has also got various pretrained adversarial models for inference. Following up the previous work on the topic, we use **RobustBench** as a main framework for conducting experiments. All the experiments were conducted on a single RTX 3060 GPU.

Data subsampling

Adversarial sparsity takes a considerable amount of time to be computer, and in order to maximize the number of experiments in this work, the subset of 50 class-balanced samples of CIFAR-10 dataset was selected.

4.2 Experiments flow

Initially, we focused more on investigating the newest state-of-the-art transformers, their ability to wrangle perturbed samples, and, of course, how adversarial sparsity changes in the models of this architecture. It was mentioned by Debenedetti et al [30] that the perturbations found for their XCiT transformer-based variation are more semantic than those of ResNet, suggesting that the robust XCiT's perturbations are more aligned with human perception. The usage of XCiT in their light recipe is thus much stronger for a capable assailant. We planned to train multiple Vision Transformers from scratch, following the training procedure outlined in [30], but having measured the compute time for a single sufficient experiments with a full train dataset on our GPU, we decided that it would be aimless to concentrate our efforts on such approach. The transformer models' size varies from 26M to 104M parameters, yielding 19-30h full training length depending on the model. We decided to go the

more traditional way and focused on inferring 3 different models – a completely adversarially untrained MobileViTv2 (1.12M) [31], XCiT-M12 (46M) [30], and a state-of-the-art CIFAR-10 benchmark Wang2023Better_WRN_70-16 (267M) [32], diffusion-based WideResNet. MobileViTv2 is a lightweight untrained vision transformer, XCiT is a thoroughly trained cross-variance image transformer trained with strong and basic augmentation in an adversarial fashion, and WRN_70-16 is a diffusion probabilistic model with wide ResNet backbone trained with TRADES [13].

4.3 Sparsity algorithms

4.3.1 L_2 sparsity

Algorithm 1: L_2 sparsity computation algorithm

Require: model f, point (x, y), $K \in \mathbb{N}, N \in \mathbb{N}$
  $k \leftarrow 0$
  $i \leftarrow 0$
  while $i \leq N$ **do**
    $\alpha_0^i \leftarrow 0, \alpha_1^i \leftarrow \pi, u_i \sim \mathcal{U}(S^n)$
    $i \leftarrow i + 1$
  end while
  while $k \leq K$ **do**
    $i \leftarrow 0$
    while $i \leq N$ **do**
      $\alpha^i \leftarrow \frac{\alpha_0^i + \alpha_1^i}{2}$
      $x_{\text{adv}} \leftarrow PGD_{\alpha^i}(f, x, y, u_i)$
      if $f(x_{\text{adv}}) \neq y$ **then**
        $\alpha_1^i \leftarrow \alpha^i$
      else
        $\alpha_0^i \leftarrow \alpha^i$
      end if
      $i \leftarrow i + 1$
    end while
    $k \leftarrow k + 1$
  end while
  return $\frac{1}{n} \sum_0^n \alpha_i$

Algorithm 1. Computation of L_2 sparsity specified in [32].

The pseudocode for L_2 sparsity involves K - a number of directions (defaulting to 100), namely the vectors in the input space along which the adversarial perturbations are created; N - a number search steps (defaulting to 10); α_0, α_1 - the min and max values of the range of arc angle to consider for creating the adversarial attack (exclusive to L2); u_i - uniform directions on the L_2 sphere; PGD - the constrained Projected Gradient Descend attack.

To estimate sparsity for L_2 (resp. L_∞) attacks, given a direction u (resp. u, σ) we explore values of α (resp. m) with binary search, and at each step run the constrained attack. Adversarial sparsity is the expected value over uniformly sampled directions.

We adjust the search range (m_0, m_1) dynamically based on the success of the attack. If the attack is successful, we reduce the upper bound of the search range; if it is not successful, we increase the lower bound.

4.3.2 L_∞ Sparsity re-implementation.

Algorithm 2 L_∞ sparsity, binary search

Require: model f, point $(x, y), K \in \mathbb{N}, N \in \mathbb{N}$

```
k ← 0
i ← 0
while i ≤ N do
    m_0^i ← n_pixels * 0.5ε, m_1^i ← n_pixels * 6ε, (u_i, σ_i), ∼ U(S^n)
    i ← i + 1
end while
while k ≤ K do
    i ← 0
    while i ≤ N do
        m^i ← (m_0^i + m_1^i)/2                          ▷ binary search
        x_adv ← PGD_{m^i}(f, x, y, (u_i, σ_i))
        if f(x_adv) ≠ y then
            m_1^i ← m^i
        else
            m_0^i ← m^i
        end if
        i ← i + 1
    end while
    k ← k + 1
end while
return (1/n) Σ_0^n m_i
```

Algorithm 2 L_∞ sparsity, binary search

Require: model f, point $(x, y), K \in \mathbb{N}, N \in \mathbb{N}$

```
k ← 0
i ← 0
while i ≤ N do
    m_0^i ← 48, m_1^i ← 289, (u_i, σ_i) ~ U(S^n)
    i ← i + 1
end while
while k ≤ K do
    i ← 0
    while i ≤ N do
        m^i ← (m_0^i + m_1^i)/2                    ▷ binary search
        x_adv ← PGD_{m^i}(f, x, y, (u_i, σ_i))
        if f(x_adv) ≠ y then
            m_1^i ← m^i
        else
            m_0^i ← m^i
        end if
        i ← i + 1
    end while
    k ← k + 1
end while
return (1/n) Σ_0^n m_i
```

Algorithm 2. Our implementation of L_∞ sparsity: the main differences being m_0, m_1 parameters used instead of α_0, α_1; their boundary values are the minimum resp. maximum number of pixels to be perturbed in each direction, i.e. 0.5ε * (32x32x3) for m_0, and 6ε * (32x32x3) for m_1; the attack is now constrained under L_∞ PGD, and the directions u are sampled with a permutation setting σ.

4.3.3 Proposed approach: L_∞ sparsity with hybrid search with n-Ary and binary counterparts

According to the authors of the work [32], while attacks over multiple directions can be computed in batches, binary search constitutes the major bottleneck of this algorithm. Some heuristics could help speeding up sparsity computation. They suggest using batched n-Ary search with fewer directions to find a first approximation of sparsity, then confirming/refining it with more directions. We tested the speed and approximation accuracy of traditional L_∞ sparsity computation, and decided to modify the bottle-neck part of the algorithm to speed up the process while retaining the good approximation of sparsity.

Algorithm 3 (Proposed) L_∞ sparsity, n-Ary hybrid search

Require: model f, point $(x, y), K \in \mathbb{N}, N \in \mathbb{N}, arity \in \mathbb{N}, 1 \leq steps_{arity} < 10$

$i, j, k \leftarrow 0$
$N_{ary} = \frac{N}{arity}$
while $i \leq N, j \leq N_{ary}$ **do**
  $m_0^i, m_0^j \leftarrow n_{pixels} * 0.5\varepsilon, m_1^i, m_1^j \leftarrow n_{pixels} * 6\varepsilon, (u_i, \sigma_i), (u_j, \sigma_j) \sim U(S^n)$
  $i \leftarrow i + 1$
  $j \leftarrow j + 1$
end while
while $k \leq steps_{arity}$ **do**
  $j \leftarrow 0$
  while $j \leq N_{ary}$ **do**
    while $1 \leq arity_i \leq arity$ **do**
      $m^i_{arity_i} \leftarrow m_0^j + \frac{arity_i(m_1^j - m_0^j)}{arity}$ ▷ n-Ary search phase
    end while
    $x_{adv} \leftarrow PGD_{m^j}(f, x, y, (u_j, \sigma_j))$
    if $f(x_{adv}) \neq y$ **then**
      $m_1^j \leftarrow m^j$
    else
      $m_0^j \leftarrow m^j$
    end if
    $j \leftarrow j + 1$
  end while
  $k \leftarrow k + 1$
end while
while $k + steps_{arity} \leq K$ **do**
  $i \leftarrow 0$
  while $i \leq N$ **do**
    $m^i \leftarrow \frac{m_0^i + m_1^i}{2}$ ▷ binary search phase
    $x_{adv} \leftarrow PGD_{m^i}(f, x, y, (u_i, \sigma_i))$
    if $f(x_{adv}) \neq y$ **then**
      $m_1^i \leftarrow m^i$
    else
      $m_0^i \leftarrow m^i$
    end if
    $i \leftarrow i + 1$
  end while
  $k \leftarrow k + 1$
end while
return $\frac{1}{n} \sum_0^n m_i$

Algorithm 3. The approach now involves two phases - n-Ary search phase that uses higher arity to quickly narrow down the range of parameters *m*; and the binary search phase which uses more directions for refined sparsity approximation.

We consider applying a hybrid approach, starting with a larger n-Ary search to quickly narrow down the potential range, and then switching to a binary search for fine-tuning.

This new version of the function processes each data sample separately and then computes the average sparsity over all samples. The n-ary search phase uses fewer directions, which should speed up the computation, while the binary search phase uses more directions for more accurate results. This hybrid approach should offer a good balance between computational efficiency and result accuracy.

4.4 Initial contribution

The authors provide an implementation of adversarial sparsity computation (both L_2 and L_∞) in their GitHub repository [36]. However, the initial code design and lack of intuitiveness of reproducing the results or at least measuring sparsity in our custom pipeline urged us to restructure and re-implement the modules and the algorithm in a more concise and convenient-to-use way.

This is the first, in our opinion, major contribution of our work – the redesigned pipeline that consists only of a singular *'linf_sparsity'* method fully reflects the mechanisms described in [32] and implemented in [36], while also introducing a convenient wrapper for adversarially trained models' inference, and an incorporation of a main evaluation metric for adversarial robustness, i.e. robust accuracy. The snippets of code are displayed in in the Appendix 1.

The results of the aforementioned experiments and notable observations are depicted in section 5.

5 RESULTS

Two approaches for n-Ary configuration were tested: 5-ary with 5 steps (all-round balanced), and 3-ary with 7 steps (quicker estimation and faster convergence at the cost of preciseness in sparsity approximation). The values for number of steps for n-Ary phase were chosen ad hoc, aiming to highlight the edges of possible configurations for this search phase, namely a comparison of robust binary search to an all-round consistent, but fast 5-ary search (half the number of steps), and the opposite extreme - 3-ary search, less arity but with more directions and more steps (predicted time efficiency).

The more comprehensive and detailed investigation of different sets of values for both the arity and the number of steps is planned for future work. Table 2 depicts all 3 models tested under different search configurations. It is important to outline that the model with the best sparsity approximation is always going to be a binary search-based one.

The first notable observation of our work is that application of hybrid n-Ary search is a classical accuracy-speed trade-off problem. Raising the arity and the number steps for the n-Ary stage yields a much faster approximation of sparsity, though it overestimates the sparsity, making the model seem more robust than it is in reality. The concept is similar to the one described in [33].

5.1 The key takeaways of the experiments

- Employing 3-ary search with 7 steps yields 30-34% computational time saving, while it also increases sparsity overestimation by 4-15%.
- The 5-ary search with 5 steps saves 7-12% time, but also lacks such a steep sparsity overestimation increase – 0.3-4%. This is explained by the fact that such a configuration is a “golden middle”, it balances the trade-off quite well, and was therefore chosen in our experiments.

- It is interesting to note that the sparsity overestimation is more impactful for less trained models. MobileViTv2 suffers 15% overestimation, WRN-70-16 though - only 6%, and XCiT, the most robust transformer, only is impacted by 4% of such overestimation.
- The sparsity ranges for each model correlate very well with the respective robust accuracy values - the higher the sparsity, the higher the set of minimum perturbed pixels is needed to fool the model.
- WRN, being the largest model, does not suffer too much from its lighter transformers counterparts (MobileViTv2 & XCiT) as the computational intrinsics of self-attention mechanism in transformers require more resource and time to backpropagate through all the nodes and more time for the model to converge.

Model	Model size, mil params	Search configuration	Robust Acc., %	Sparsity	Time, it/s	Time saved, %	Sparsity deviation, %
MobileViTv2	1.12	binary	0.12	62.4	13.1	-	-
MobileViTv2	1.12	3-ary, 7 steps	0.12	71.5	9.8	**34**	15
MobileViTv2	1.12	5-ary, 5 steps	0.10	64.9	11.6	12	**4**
WRN-70-16	267	binary	0.84	142.4	24.5		-
WRN-70-16	267	3-ary, 7 steps	0.84	150.4	18.8	**30**	6
WRN-70-16	267	5-ary, 5 steps	0.84	142.8	22.0	11	**0.3**
XCiT-M	46	binary	0.56	115.2	21.1	-	-
XCiT-M	46	3-ary, 7 steps	0.56	119.7	16.1	**31**	4
XCiT-M	46	5-ary, 5 steps	0.56	116.4	19.7	7	**1**

Table 2. The comparison of binary- and n-Ary-based sparsity computation configurations between MobileViTv2, WRN-70-16, and XCiT-M on 50 CIFAR-10 samples. Time consumption decrease is best with 3-ary, 7 steps and the overestimation of sparsity gained - with 5-ary, 5 steps (bold); the baseline, most accurate

approximation of sparsity is considered to be achieved with binary configuration (underlined).

5.2 Impact of number of directions

The further study of time saving impact has been incorporated. The analysis compared binary and n-Ary search setups on single sample under different *n_cones_sparsity* configurations (from 10 to 100 with step 10).

The average time gain for this experiment comprised **6.84%** (figure 5). Interestingly, the highest efficiency increase is reached at either small or very large numbers of directions. This can potentially be explained that arity-based time decrease is the most impactful during early stages of the directions approximation, or much farther into the numbers.

```
6 print(relative_timegain)
7 print('{:.4f}'.format(np.mean(relative_timegain)))
```

```
[0.1303 0.0894 0.0417 0.0445 0.0449 0.0457 0.045  0.0591 0.0817 0.1017]
0.0684
```

Figure 5. The demonstration of relative time gain for each setup with 10, 20, …, 100 dimensions and a the mean relative time gain across all the experiments.

As number of directions *n* increases, the change in approximation of sparsity over time, i.e. increased number of directions, decreases. Further increasing the direction count will not yield more exact approximation of sparsity while only increasing the computational cost of the algorithm.

5.3 Sparsity estimation time and directions count analysis

In this work, we also conducted analyses of the dependence of sparsity and computational time against the number of directions.

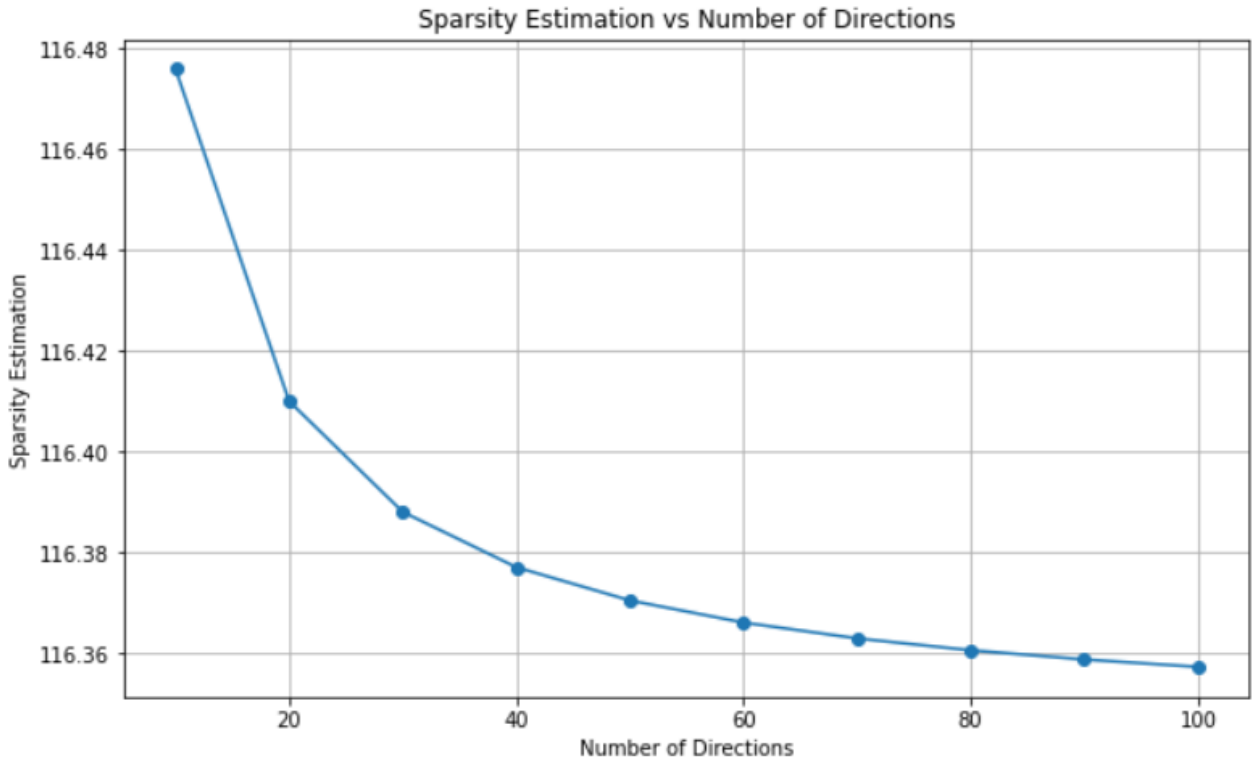

Figure 6. The plot of sparsity estimation against the number of directions. We can observe a sparsity overestimation pattern similar to hyperbolic decay. Choosing the right number of directions, perhaps similar to "elbow" method, may yield an optimally selected trade-off between speed and estimation accuracy.

Finding the optimal number of directions for sparsity approximation, i.e. with best speed and approximation consistency, is without a doubt an important and separate point for investigation. We assume there is an optimal stopping point where the cost of another iteration does not supersede the gain from the target loss decrease (in our case, sparsity overestimation), similar to the know mechanism of early stopping [35]. For instance, experiments on finding the optimal value of absolute or relative decrease in margin (e.g. delta of sparsity change) can be conducted in order to pinpoint a well-balanced setup of parameters for a more convenient and easier testing of new

hypotheses concerning adversarial sparsity. It should be noted, however, that such a preliminary parameter setup requires individual tuning for each dataset, as per different levels of r-separability, and for each model due to their innately high intra-class variation.

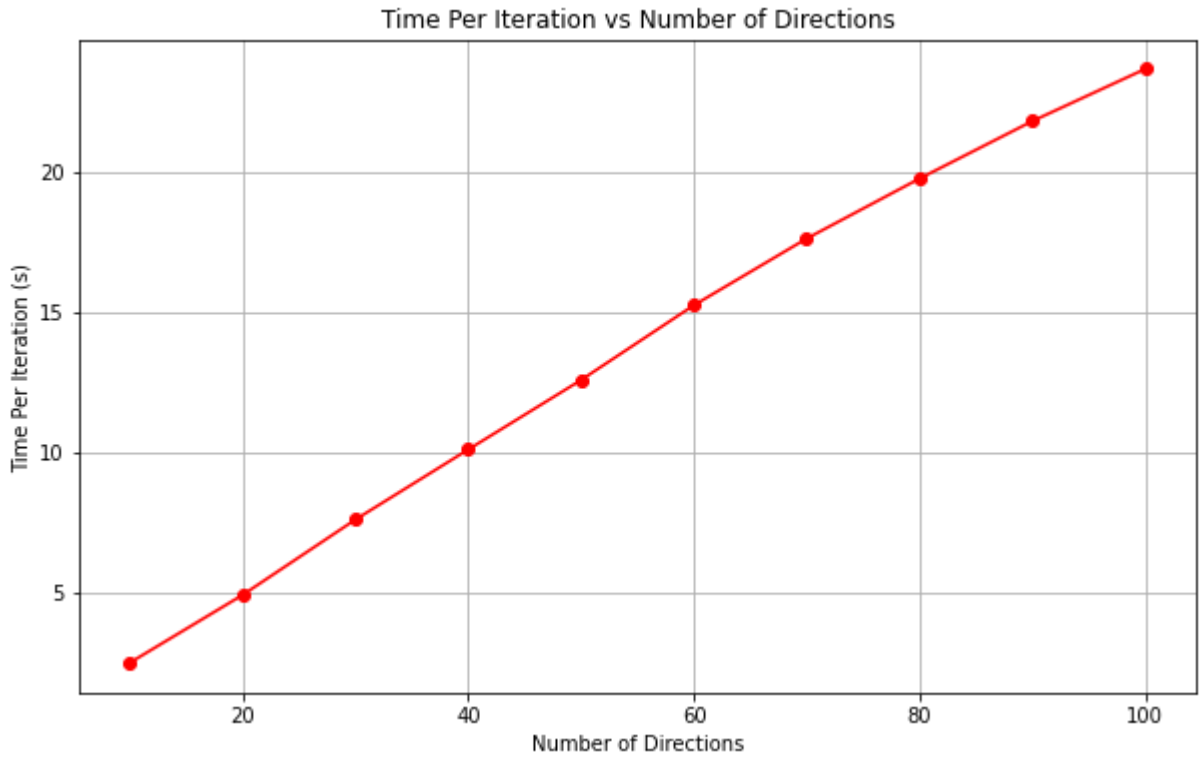

Figure 7. The plot of computational time spent against the number of directions. It is, expectedly, linear, therefore the selection of directions revolves around the number of samples in O(*N)* time complexity.

6 CONCLUSIONS

The study of adversarial robustness in machine learning models, particularly in the field of computer vision, has been the crux of our research. This important aspect of model development has far-reaching implications for a wide array of real-world applications, from autonomous vehicles to medical imaging, and has the potential to significantly influence the resilience of models in the face of adversarial attacks.

The research presented in this work revolved around adversarial sparsity estimation in computer vision models, aiming to further illuminate the underlying mechanisms contributing to their robustness or vulnerability. To this end, we introduced an improved n-Ary search variation of the traditional binary search algorithm. This novel modification successfully mitigated the inefficiencies associated with high-dimensional inputs, and demonstrated a more efficient computation process, while ensuring comparable sparsity estimates.

In our research, we explored two different approaches for the n-Ary configuration: a 5-ary search with 5 steps and a ternary search with 7 steps. One important observation is that the model with the best sparsity approximation always relies on a binary search-based approach.

Our experiments revealed that the use of a hybrid n-Ary search introduces a classic accuracy-speed trade-off. Increasing the arity and the number of steps in the n-Ary stage leads to faster sparsity approximation but at the cost of overestimating the sparsity. This means that the model may appear more robust than it is.

The key takeaways from our work are as follows. Employing a ternary search with 7 steps resulted in computational time savings of 30-34%, accompanied by a sparsity overestimation increase of 4-15%. On the other hand, the 5-ary search with 5 steps offered time savings of 7-12% with a smaller increase in sparsity overestimation of 0.3-4%. This configuration strikes a balance between time savings and sparsity approximation, making it an optimal choice for our experiments.

We also observed that the impact of sparsity overestimation is more pronounced in less trained models. MobileViTv2 exhibited a 15% overestimation, while WRN-70-16 showed only a 6% overestimation. XCiT, the most robust transformer, experienced a 4% overestimation. One of the key findings was an approximate 6.84% increase in inference time, a significant reduction compared to the binary search procedure, showcasing the efficacy and efficiency of the n-Ary search algorithm.

Furthermore, we found a strong correlation between sparsity ranges and robust accuracy values for each model. A higher sparsity indicates the need for a larger set of admissible perturbations to fool the model.

Lastly, we observed that the larger WRN model was less affected by its lighter transformer counterparts (MobileViTv2 and XCiT) due to the computational demands of the self-attention mechanism in transformers. The self-attention mechanism requires more resources and time for backpropagation and convergence.

These findings shed light on the trade-offs involved in the hybrid n-Ary search approach and its impact on sparsity approximation and computational efficiency. The insights gained from this research can contribute to the development of more efficient and accurate algorithms for adversarial sparsity calculation, ultimately improving the understanding and robustness of models against adversarial attacks.

REFERENCES

[1] Goodfellow et al., “Explaining and Harnessing Adversarial Examples”, *arXiv:*1412.6572, Dec. 2014.

[2] Madry et al., “Towards Deep Learning Models Resistant to Adversarial Attacks”, *arXiv:*1706.06083, Jun. 2017.

[3] Yang et al., “A Closer Look at Accuracy vs. Robustness”, *arXiv:*2003.02460, Mar. 2020.

[4] Carlini, Wagner “Towards Evaluating the Robustness of Neural Networks”, *arXiv:*1608.04644, Aug. 2016.

[5] Croce et al., “Reliable Evaluation of Adversarial Robustness with an Ensemble of Diverse Parameter-free Attacks”, *arXiv:*2003.01690, Mar. 2020.

[6] Croce et al., “Minimally distorted Adversarial Examples with a Fast Adaptive Boundary Attack”, *arXiv:*1907.02044, Jul. 2019.

[7] Andriushchenko et al., “Square Attack: a query-efficient black-box adversarial attack via random search”, *arXiv:*1912.00049, Jul. 2020.

[8] Chen et al., “RayS: A Ray Searching Method for Hard-label Adversarial Attack”, *arXiv:*2006.12792, Jun. 2020.

[9] Wu et al., “Adversarial Weight Perturbation Helps Robust Generalization”, *arXiv:*2004.05884, Oct. 2020.

[10] Rebuffi et al., “Fixing Data Augmentation to Improve Adversarial Robustness”, *arXiv:*2103.01946v2, Oct. 2021.

[11] Carmon et al., “Unlabeled Data Improves Adversarial Robustness”, *arXiv:*1905.13736v4, Jan. 2022.

[12] Raghunathan et al., “Understanding and mitigating the tradeoff between robustness and accuracy”, *arXiv:*2002.10716, Feb. 2020.

[13] Zhang et al., “Theoretically Principled Trade-off between Robustness and Accuracy”, *arXiv:*1901.08573, Jun. 2019

[14] Athalye et al., “Obfuscated gradients give a false sense of security: circumventing defenses to adversarial examples”, *arXiv:*1802.00420v4, Jul. 2018.

[15] R. Olivier, B. Raj, “How many perturbations break this model? Evaluating robustness beyond adversarial accuracy”, *arXiv:*2207.04129, Aug. 2022.

[16] Croce et al., “RobustBench: a standardized adversarial robustness benchmark”, *arXiv:*2010.09670v3, Oct. 2021.

[17] Hirano et al., “Simple iterative method for generating targeted universal adversarial perturbations”, *arXiv:*1911.06502v2, Nov. 2019.

[18] Wu et al., “Attacking Adversarial Attacks as A Defense”, *arXiv:*2106.04938, Jun. 2021.

[19] Brendel et al., “Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models”, *arXiv:*1712.04248, Dec. 2017

[20] Qin et al., “Adversarial robustness through local linearization”, in Advances in Neural Information Processing Systems, pages 13824–13833, 2019.

[21] Finlay et al., “Scaleable input gradient regularization for adversarial robustness”, arXiv:1905.11468v2, Oct. 2019.

[22] Wong et al., “Fast is better than free: Revisiting adversarial training”, arXiv:2001.03994, Jan. 2020.

[23] Addepalli et al., “Efficient and Effective Augmentation Strategy for Adversarial Training”, *arXiv:*2210.15318, Oct. 2022.

[24] Zeng et al., “A Data Augmentation-based Defense Method Against Adversarial Attacks in Neural Networks”, *arXiv:*2007.15290, Jul. 2020.

[25] Wu et al., “Adversarial Weight Perturbation Helps Robust Generalization”, *arXiv:*2004.05884v2, Oct. 2020.

[26] Liu et al., “Feature Distillation: DNN-Oriented JPEG Compression Against Adversarial Examples”, *arXiv:*1803.05787v2, Apr. 2019.

[27] O'Shea et al., “An Introduction to Convolutional Neural Networks”, *arXiv:*1511.08458v2, Dec. 2015.

[28] Dosovitskiy et al., “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, *arXiv:*2010.11929v2, Jun. 2021.

[29] Ho et al., “Denoising Diffusion Probabilistic Models”, *arXiv:*2006.11239v2, Dec. 2020.

[30] Debendetti et al., “A Light Recipe to Train Robust Vision Transformers”, *arXiv* 2209.07399, Feb. 2023.

[31] Mehta et al., “Separable Self-attention for Mobile Vision Transformers”, *arXiv:*2206.02680, Jun. 2022.

[32] Wang et al., “Better Diffusion Models Further Improve Adversarial Training”, *arXiv* 2302.04638, Feb. 2023.

[33] Ma et al, “Imbalanced Gradients: A Subtle Cause of Overestimated Adversarial Robustness”, *arXiv* 2006.13726, Jun. 2020.

[34] R. Olivier, B. Raj, Sparsity (implementation of [15]), 2022. [Online]. Available: https://github.com/RaphaelOlivier/sparsity

[35] Raskutti et al., “Early stopping and non-parametric regression: An optimal data-dependent stopping rule”, *arXiv:*1306.3574, Jun. 2013.

[36] Wang et al., “Analyzing the robustness of nearest neighbors to adversarial examples” in *International Conference on Machine Learning*, pages 5133–5142, Jul. 2018.

[37] Szegedy et al., “Intriguing properties of neural networks”, *arXiv:*1312.6199, Feb. 2014.

[38] Gu et al., “Effective and Efficient Vote Attack on Capsule Networks”, *arXiv:*2102.10055, Feb. 2021.

[39] Goval et al., “An Alternative Surrogate Loss for PGD-based Adversarial Testing”, *arXiv:*1910.09338, Oct. 2019.

[40] Tramèr et al., “The Space of Transferable Adversarial Examples”, *arXiv:*1704.03453v2, May 2017.

[41] Li et al., “Neural Architecture Dilation for Adversarial Robustness”, *arXiv:*2108.06885, Aug. 2021.

[42] O’shea et al., “An Introduction to Convolutional Neural Networks”, *arXiv:*1511.08458v2, Dec. 2015.

APPENDIX 1

```
def linf_sparsity(dataset, model, num_search_steps=10, min_pix=48, max_pix=289,
                  n_cones_sparsity=100, n_ary=5, n_ary_steps=5, eps=8/255):
    model.eval()
    sparsity_results = []
    robust_accuracy = []
    total_samples = 0
    correctly_classified_samples = 0

    for batch_idx, (x, y) in tqdm(enumerate(dataset)):
        x = x.to(device)
        y = y.to(device)
        total_samples += y.size(0)
        n_cones_short = n_cones_sparsity // n_ary

        def generate_random_directions(n_cones, shape):
            directions = torch.randn(n_cones, *shape)
            directions = directions.view(n_cones, -1)
            directions /= directions.norm(dim=1, keepdim=True)
            return directions.view(n_cones, *shape)

        directions_full = generate_random_directions(n_cones_sparsity, x.shape[1:]).to(device)
        directions_short = generate_random_directions(n_cones_short, x.shape[1:]).to(device)

        pixel_masks_idx = torch.stack([torch.randperm(x.numel()) // x.shape[0] for _ in range(n_cones_sparsity)])

        m_0, m_1 = torch.ones(n_cones_sparsity).mul_(min_pix), torch.ones(n_cones_sparsity).mul_(max_pix)
```

Figure 8. Data loading, util function for direction sampling, and initialization of directions, pixel masks, and m_0, m_1 parameters. The higher the arity, the fewer directions are to be explored during n-Ary stage of the algorithm (1/n dependency).

```
for i in range(num_search_steps):
    correctly_classified = True
    if i < n_ary_steps:  # perform n-ary search with fewer directions
        n_cones_current = n_cones_short
        directions = directions_short
        npixels_list = [(m_0 + k * (m_1 - m_0) // n_ary) for k in range(1, n_ary)]
    else:  # switch to binary search with all directions
        n_cones_current = n_cones_sparsity
        directions = directions_full
        npixels_list = [(m_0 + m_1) // 2]
```

Figure 9. The main part of the algorithm: a hybrid search which first performs an n-Ary search for a predefined number of steps in order approximate sparsity efficiently, and a binary counterpart which solidifies the expected outputs by approving the perturbations and directions and strengthening sparsity.

```
for npixels in npixels_list:
    constraint = torch.zeros_like(directions.view(n_cones_current, -1))

    for j in range(n_cones_current):
        current_mask_idx = pixel_masks_idx[j, :int(npixels[j])]
        constraint[j, current_mask_idx] = 1

    attack_successes = []
    for j in range(n_cones_current):
        direction = directions[j].unsqueeze(0)
        # attack radii are bound by `clamp`
        perturbation = (x + eps * direction * constraint[j].view(x.shape)).clamp(0, 1)
        pred = model(perturbation)
        attack_success = (pred.argmax(-1) != y).item()
        if attack_success:
            attack_successes.append(j)
            correctly_classified = False

    mask = torch.tensor(attack_successes, dtype=torch.long)

    if attack_success:
        m_1[mask] = npixels[mask - 1]
    else:
        m_0[mask] = npixels[mask - 1]
```

Figure 10. The technical part of the algorithm that loops over each direction, creates a unique perturbation by taking into account ε-boundary and PGD-constrained attack. The model then infers on the perturbed image; failing to correctly classify the class results in robust accuracy being reduced; once all the directions have been covered, a mask with successful attacks is created in order to update each direction in upper and lower boundaries (m_0, m_1) with the appropriate values from the n-Ary search list.

```
    if correctly_classified:
        correctly_classified_samples += y.size(0)
    sparsity_results.append(((m_1 + m_0) // 2).mean().item())

robust_accuracy = correctly_classified_samples / total_samples
sparsity = np.mean(sparsity_results)

return robust_accuracy, sparsity
```

Figure 11. The algorithm's step converges with a simple robust accuracy estimate and a mean of all the accumulated samples' sparsity during inference for the expected value of L_∞ sparsity.