Adversarial Vulnerability of Neural Networks Increases with Input Dimension

Carl-Johann Simon-Gabriel [1 2] **Yann Ollivier** [1] **Bernhard Schölkopf** [2] **Léon Bottou** [1] **David Lopez-Paz** [1]

Abstract

Over the past four years, neural networks have proven vulnerable to adversarial images: targeted but imperceptible image perturbations lead to drastically different predictions. We show that adversarial vulnerability increases with the gradients of the training objective when seen as a function of the inputs. For most current network architectures, we prove that the ℓ_1-norm of these gradients grows as the square root of the input-size. These nets therefore become increasingly vulnerable with growing image size. Over the course of our analysis we rediscover and generalize *double-backpropagation*, a technique that penalizes large gradients in the loss surface to reduce adversarial vulnerability and increase generalization performance. We show that this regularization-scheme is equivalent at first order to training with adversarial noise. Finally, we demonstrate that replacing strided by average-pooling layers decreases adversarial vulnerability. Our proofs rely on the network's weight-distribution at initialization, but extensive experiments confirm their conclusions after training.

1. Introduction

Since the trendsetting paper of Goodfellow et al. (2014), Convolutional Neural Networks (CNNs) have been found utterly vulnerable to adversarial examples: an adversary can drive the performance of state-of-the art CNNs down to chance-level with imperceptible changes of the inputs. A number of studies have tried to address this issue, but only few have stressed that, because adversarial examples are essentially small input changes that create large output variations, they are inherently caused by large gradients of the neural network with respect to its inputs. Two notable exceptions are Hein and Andriushchenko (2017) and Cisse et al. (2017), who offer adversarial robustness guarantees that do depend on these gradients.

Contributions. In the same spirit, we present both theoretical arguments and an empirical one-to-one relationship between the gradient norm of the training objective and adversarial vulnerability. By evaluating those norms based on the weight-statistics at initialization, we show that, *by design*, CNNs exhibit increasingly large gradients with input dimension d, which leaves them more and more vulnerable to adversarial noise. This vulnerability is dampened by average-pooling layers, but not by strided and max-pooling ones. Based on the link between large loss-gradients and adversarial vulnerability, we propose a regularizer that explicitely penalizes the norm of these gradients – a technique already proposed by Drucker and LeCun (1991) under the name of *double-backpropagation*. Just as Tikhonov regularization is equivalent to training with random noise (Bishop, 1995), we show that our proposed regularization-scheme is equivalent at frist order to training with adversarial noise, as proposed by Goodfellow et al. (2014). We confirm our theoretical results by extensive experiments. Overall, our findings call for a new line of research in the design of neural network architectures with inherently smaller gradients, as pioneered by Cisse et al. (2017). Our results may thereby serve as guidelines to practitioners and network-designers.

2. From Adversarial Examples to Large Gradients

Suppose that a given classifier φ classifies an image $\boldsymbol{x}$ as being in category $\varphi(\boldsymbol{x})$. By definition, an adversarial image is a small modification of $\boldsymbol{x}$, barely noticeable to the human eye, that suffices to fool the classifier into predicting a class different from $\varphi(\boldsymbol{x})$. It is a *small* perturbation of the inputs, that creates a *large* variation of outputs. Adversarial examples are thus seem inherently related to large gradients of the network. A connection, that we will now clarify.

Adversarial Vulnerability and its Variations. In practice, an adversarial image is constructed by adding a perturbation $\boldsymbol{\delta}$ to the original image $\boldsymbol{x}$ such that $\|\boldsymbol{\delta}\| \leq \epsilon$ for some (small) number ϵ and a given norm $\|\cdot\|$ over the input space. We call the perturbed input $\boldsymbol{x} + \boldsymbol{\delta}$ an ϵ-sized $\|\cdot\|$-attack and say that the attack was successfull when $\varphi(\boldsymbol{x} + \boldsymbol{\delta}) \neq \varphi(\boldsymbol{x})$. This motivates

[1]Facebook AI Research, Paris/New York [2]Empirical Inference Department, Max Planck Institute for Intelligent Systems, Tübingen, Germany. Correspondence to: Carl-Johann Simon-Gabriel <cjsimon@tue.mpg.de>.

Definition 1. Given a distribution P over the input-space, the *(total) adversarial vulnerability* of a classifier φ to an ϵ-sized $\|\cdot\|$-attack is the probability that there exists a perturbation $\boldsymbol{\delta}$ of $\boldsymbol{x}$ such that

$$\|\boldsymbol{\delta}\| \leq \epsilon \quad \text{and} \quad \varphi(\boldsymbol{x}) \neq \varphi(\boldsymbol{x}+\boldsymbol{\delta})\,. \tag{1}$$

When the classifier φ classifies all test images $\boldsymbol{x}$ drawn from P with 100% accuracy, the adversarial vulnerability of φ is exactly the average (for $\boldsymbol{x} \sim P$) increase $\Delta\mathcal{L}_{0/1}$ of the induced zero-one-loss $\mathcal{L}_{0/1}(\varphi(\boldsymbol{x}), c)$ after adversarial perturbation of the test inputs. Here, c designates the true label of $\boldsymbol{x}$; and as we shall only consider induced losses, let us agree to simply call them 'the loss' and write for instance $\mathcal{L}_{0/1}(\boldsymbol{x}, c)$. This equality between the average loss increase and adversarial vulnerability is only an approximate one if the classifier has only 90% accuracy on P, because switches between two wrong classes will be ignored and switchees from wrong to true classes may compensate switches in the other direction. In practice however, a practitionner will be more interested in the accuracy-drop after adversarial attacks on a trained and accurate classifier, than in the exact count of class-switches. So we may actually call this average of $\Delta\mathcal{L}_{0/1}$ the *relevant adversarial vulnerability* and remember that, with rising accuracy, it is an increasingly good approximation of the total adversarial vulnerability.

In practice, we usually do not train our classifiers with the zero-one loss. We replace it by a smoother loss $\mathcal{L}$, which we consider a good enough approximation of $\mathcal{L}_{0/1}$ – for example the cross-entropy (induced)-loss. But if we agree that $\mathcal{L}$ is a good enough approximation of $\mathcal{L}_{0/1}$, then we also ought to agree that the (non-infinitesimal) variations of $\mathcal{L}$ accurately reflect those of $\mathcal{L}_{0/1}$; and hence, that the average (for $\boldsymbol{x} \sim P$) of $\Delta\mathcal{L}(\boldsymbol{x}, c)$ faithfully approximates the relevant adversarial vulnerability. Consequently, a trained classifier φ can only be robust to adversarial examples if, on average over $\boldsymbol{x}$, a small adversarial perturbation $\boldsymbol{\delta}$ of $\boldsymbol{x}$ creates only a small variation $\delta\mathcal{L}$ of the loss. But if $\|\boldsymbol{\delta}\| \leq \epsilon$, then using a first order Taylor expansion in ϵ shows that

$$\begin{aligned}\delta\mathcal{L} &= \max_{\boldsymbol{\delta}\,:\,\|\boldsymbol{\delta}\|\leq\epsilon} |\mathcal{L}(\boldsymbol{x}+\boldsymbol{\delta}, c) - \mathcal{L}(\boldsymbol{x}, c)| \\ &\approx \max_{\boldsymbol{\delta}\,:\,\|\boldsymbol{\delta}\|\leq\epsilon} |\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\cdot\boldsymbol{\delta}| = \epsilon\,|||\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}|||,\end{aligned} \tag{2}$$

where $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$ denotes the gradient of $\mathcal{L}$ with respect to $\boldsymbol{x}$, and where the last equality is almost the definition of the dual norm $|||\cdot|||$ of $\|\cdot\|$. Now two remarks. First: the dual norm only kicks in because we let the input noise $\boldsymbol{\delta}$ optimally adjust to the coordinates of $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$ within its ϵ-constraint. This is the brandmark of *adversarial* noise: the different coordinates add up, instead of statiscally canceling each other out as they would with random noise. For example, if we impose that $\|\boldsymbol{\delta}\|_2 \leq \epsilon$, then $\boldsymbol{\delta}$ will stricly align with $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$. If instead $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$, then $\boldsymbol{\delta}$ will align with the sign of the coordinates of $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$. Second remark: the Taylor expansion in (2) may actually be dominated by higher-order terms. Nevertheless, the first-order term shows that *if* our loss surface has large gradients, *then* our net will be vulnerable to adversarial attacks. This explains the importance of:

Lemma 1. *At first order approximation in ϵ, an ϵ-sized adversarial attack generated with norm $\|\cdot\|$ increases the loss $\mathcal{L}$ at point $\boldsymbol{x}$ by $\epsilon\,|||\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}|||$, where $|||\cdot|||$ is the dual norm of $\|\cdot\|$. Consequently, we can evaluate the adversarial vulnerability of a trained classifier to ϵ-sized $\|\cdot\|$-attacks by $\epsilon\,\mathbb{E}_{\boldsymbol{x}}|||\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}|||$.*

In particular, an ϵ-sized ℓ_p-attack increases the loss by $\epsilon\,\|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q$ where $1 \leq p \leq \infty$ and $1/p + 1/q = 1$.

The second paragraph of Lemma 1 follows from the well-known property that the dual norm of an ℓ_p-norm is the corresponding ℓ_q-norm.

A new old regularizer. Since Lemma 1 shows that the loss of the network after an $\epsilon/2$-sized $\|\cdot\|$-attack is

$$\mathcal{L}_{\epsilon,|||\cdot|||}(\boldsymbol{x}, c) := \mathcal{L}(\boldsymbol{x}, c) + \frac{\epsilon}{2}\,|||\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}|||\,, \tag{3}$$

it is natural to take this loss-after-attack as a new training objective. Here we introduced a factor 2 for reasons that will become clear in a moment. Incidentally, for $\|\cdot\| = \|\cdot\|_2$, this new loss reduces to an old regularization-scheme proposed by Drucker and LeCun (1991) called *double-backpropagation*. At the time, the authors argued that slightly decreasing a function's or a classifier's sensitivity to input perturbations should improve generalization. In a sense, this is exactly our motivation when defending against adversarial examples. It is thus not surprising to end up with the same regularization term. Note that our reasoning only shows that training with one specific norm $|||\cdot|||$ in Equation 3 helps to protect against adversarial examples generated from $\|\cdot\|$. A priori, we do not know what will happen for attacks generated with another norm; but our experiments suggest that training with one norm also protects against other attacks (see Section 4.2).

Link to adversarially-augmented training. In Equation 1, ϵ designates an attack-size threshold, while in (3), it is a regularization-strength. Rather than a notational conflict, this reflects an intrinsic duality between two complementary interpretations of ϵ, which we now investigate further. Suppose that, instead of using the loss-after-attack, we augment our training set with ϵ-sized $\|\cdot\|$-attacks $\boldsymbol{x}+\boldsymbol{\delta}$, where for each training point $\boldsymbol{x}$, the perturbation $\boldsymbol{\delta}$ is generated on the fly to locally maximize the loss-increase. Then we are effectively training with

$$\tilde{\mathcal{L}}_{\epsilon,\|\cdot\|}(\boldsymbol{x}, c) := \frac{1}{2}(\mathcal{L}(\boldsymbol{x}, c) + \mathcal{L}(\boldsymbol{x}+\epsilon\,\boldsymbol{\delta}, c))\,, \tag{4}$$

where by construction $\boldsymbol{\delta}$ satisfies Equation 2. We will refer to this technique as *adversarially augmented training*. It

was first introduced by Goodfellow et al. (2014) with $\|\cdot\| = \|\cdot\|_\infty$ under the name of FGSM[1]-augmented training. Using the first order Taylor expansion in ϵ of Equation 2, this 'old-plus-post-attack' loss of Equation 4 simply reduces to our loss-after-attack.

Proposition 2. *Up to first-order approximations in ϵ,*

$$\tilde{\mathcal{L}}_{\epsilon,\|\cdot\|} = \mathcal{L}_{\epsilon,|||\cdot|||}\,.$$

For small enough ϵ, adversarially-augmented training with ϵ-sized $\|\cdot\|$-attacks amounts to penalizing the dual *norm $|||\cdot|||$ of $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$ with weight $\epsilon/2$.*

In particular, double-backpropagation corresponds to training with ℓ_2-attacks, while FGSM-augmented training corresponds to an ℓ_1-penalty on $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$.

This correspondance between training with preturbations and using a regularizer can be compared to Tikhonov regularization: Tikhonov regularization amounts to training with *random* noise (Bishop, 1995), while training with *adversarial* noise amounts to penalizing $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$.

Calibrating the threshold ϵ to the attack-norm $\|\cdot\|$. Going back to Lemma 1, we see that adversarial vulnerability depends on three main factors:

(i) $\|\cdot\|$, the norm chosen for the attack
(ii) ϵ, the size of the attack, and
(iii) $\mathbb{E}_{\boldsymbol{x}}|||\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}|||$, the expected *dual* norm of $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$.

We could see Point (i) as a measure of our sensibility to image perturbations, (ii) as our sensibility threshold, and (iii) as the classifier's expected marginal sensibility to a unit perturbation. $\mathbb{E}_{\boldsymbol{x}}|||\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}|||$ hence intuitively captures the discrepancy between our perception (as modeled by $\|\cdot\|$) and the classifier's perception for an input-perturbation of small size ϵ. Of course, this viewpoint supposes that we actually found a norm $\|\cdot\|$ (or more generally a metric) that faithfully reflects human perception – a project in its own right, far beyond the scope of this paper. However, it is clear that the threshold ϵ that we choose should depend on the norm $\|\cdot\|$ and hence on the input-dimension d. In particular, for a given pixel-wise order of magnitude of the perturbations $\boldsymbol{\delta}$, the ℓ_p-norm of the perturbation will scale like $d^{1/p}$. This suggests to write the threshold ϵ_p used with ℓ_p-attacks as:

$$\epsilon_p = \epsilon_\infty\, d^{1/p}\,, \tag{5}$$

where ϵ_∞ denotes a dimension-independent constant. In Appendix C we show that this scaling also preserves the average signal-to-noise ratio $\|\boldsymbol{x}\|_2 \,/\, \|\boldsymbol{\delta}\|_2$, both accross norms and dimensions, so that ϵ_p could correspond to a constant human perception-threshold. With this in mind, we now turn to our main contribution, Point (iii): estimating $\mathbb{E}_x\|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q$ for standard neural nets.

[1]FGSM = *F*ast *G*radient *S*ign *M*ethod

3. Evaluating Adversarial Vulnerability by Estimating $\|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q$

In this section, we evaluate the size of $\|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q$ for standard neural network architectures. We analyse fully-connected networks, followed by a more general theorem that in particular encompasses CNNs with or without strided convolutions. We finish by showing that, contrary to strided layers, average-poolings efficiently decrease the norm of $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$.

We start our analysis by showing how changing q affects the size of $\|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q$. Suppose for a moment that the coordinates of $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$ have typical magnitude $|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}|$. Then $\|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q$ scales like $d^{1/q}|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}|$. Consequently

$$\epsilon_p\,\|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q \;\propto\; \epsilon_p\, d^{1/q}\,|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}| \;\propto\; d\,|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}|\,. \tag{6}$$

This equation carries two important messages. First, we see how $\|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q$ depends on d and q. The dependance seems highest for $q = 1$. But once we account for the varying perceptibility threshold $\epsilon_p \propto d^{1/p}$, we see that adversarial vulnerability scales like $d \cdot |\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}|$, whatever ℓ_p-norm we use. Second, Equation 6 shows that to be robust against any type of ℓ_p-attack at any input-dimension d, the average absolute value of the coefficients of $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$ must grow slower than $1/d$. Now, here is the catch, which brings us to our core insight.

3.1. Core Idea: One Neuron with Many Inputs

In order to preserve the activation variance of the neurons from layer to layer, the neural weights are usually initialized with a variance that is inversely proportional to the number of inputs per neuron. Imagine for a moment that the network consisted only of one output neuron o linearly connected to all input pixels. For the purpose of this example, we assimilate o and $\mathcal{L}$. Because we initialize the weights with a variance of $1/d$, their average absolute value $|\boldsymbol{\partial}_x o| \equiv |\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}|$ grows like $1/\sqrt{d}$, rather than the required $1/d$. By Equation 6, the adversarial vulnerability $\epsilon\,\|\boldsymbol{\partial}_x o\|_q \equiv \epsilon\,\|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q$ therefore increases like $d/\sqrt{d} = \sqrt{d}$.

This toy example shows that the standard initialization scheme, which preserves the variance from layer to layer, causes the average coordinate-size $|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}|$ to grow like $1/\sqrt{d}$ instead of $1/d$. When an ℓ_∞-attack tweaks its ϵ-sized input-perturbations to align with the coordinate-signs of $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$, all coordinates of $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$ add up in absolute value, resulting in an output-perturbation that scales like $\epsilon\sqrt{d}$ and leaves the network increasingly vulnerable with growing input-dimension.

3.2. Generalization to Deep Networks

Our next theorems generalize the previous toy example to a very wide class of feedforward nets with ReLU activation functions. For illustrational purposes, we start with fully connected nets and only then proceed to the broader

class, which includes any succession of convolutional and/or strided layers. In essence, the proofs iterate our insight on one layer over a sequence of layers. They all rely on the following set ($\mathcal{H}$) of hypotheses:

- H1 There is 1 ReLU after each non-input neuron, which kills half of its inputs, independently of the weights.
- H2 Neurons are partitioned into layers, meaning groups that each path traverses at most once.
- H3 All weights have 0 expectation and variance $2/(\text{in-degree})$, as in the 'He-initialization'.
- H4 The weights from different layers are independent.
- H5 Two weights in a same layer are uncorrelated.

Here, two weights are considered in the same layer if their end-nodes belong to the same layer. If we follow common practice and initialize our nets as proposed by He et al. (2015), then H3-H5 are satisfied at initialization by design, while H1 is usually a very good approximation (Balduzzi et al., 2016). After training however, we cannot expect these hypotheses to hold. While if necessary we may try to enforce H1 and H3 by adding bathchnorms or penalties on the layerwise weight averages or variances, it is absurd to believe that we can keep all weights uncorrelated as implied by H4-H5. That is why, all our statements in this section are to be understood as *orders of magnitudes* that are very well satisfied at initialization in theory and in practice , and that we will confirm experimentally after training in Section 4. Said differently, while our theorems rely on the statistics of neural nets at initialization, our experiments confirm their conclusions after training.

Theorem 3 (Vulnerability of Fully Connected Nets). *Consider a succession of fully connected layers with ReLU activations which takes inputs $\boldsymbol{x}$ of dimension d, satisfies assumptions ($\mathcal{H}$), and outputs logits $f_k(\boldsymbol{x})$ that get fed to a final cross-entropy-loss layer $\mathcal{L}$. Then the coordinates of $\boldsymbol{\partial}_{\boldsymbol{x}} f_k$ grow like $1/\sqrt{d}$, and*

$$\|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q \propto d^{\frac{1}{q}-\frac{1}{2}} \quad \text{and} \quad \epsilon_p \|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q \propto \sqrt{d}\,. \tag{7}$$

Consequently, these networks are increasingly vulnerable to any kind of ℓ_p-attacks with growing input-dimension.

Proof. Let x designate a generic coordinate of $\boldsymbol{x}$. To evaluate the size of $\|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q$, we will evaluate the size of the coordinates $\partial_x\mathcal{L}$ of $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$ by decompose them into

$$\partial_x\mathcal{L} = \sum_{k=1}^{K} \frac{\partial\mathcal{L}}{\partial f_k}\frac{\partial f_k}{\partial x} =: \sum_{k=1}^{K} \partial_k\mathcal{L}\,\partial_x f_k,$$

where $f_k(\boldsymbol{x})$ denotes the logit-probability of $\boldsymbol{x}$ belonging to class k. We now investigate the statistical properties of the logit gradients $\boldsymbol{\partial}_{\boldsymbol{x}} f_k$, and then see how they shape $\partial_x\mathcal{L}$.

Step 1: Statistical properties of $\boldsymbol{\partial}_{\boldsymbol{x}} f_k$. Let $\mathcal{P}(x,k)$ be the set of paths $\boldsymbol{p}$ from input neuron x to output-logit k. Let $p-1$ and p be two successive neurons on path $\boldsymbol{p}$, and $\tilde{\boldsymbol{p}}$ be the same path $\boldsymbol{p}$ but without its input neuron. Let w_p designate the weight from $p-1$ to p and $\omega_{\boldsymbol{p}}$ be the *path-product* $\omega_{\boldsymbol{p}} := \prod_{p\in\tilde{\boldsymbol{p}}} w_p$ and W the set of all weights. Finally, let σ_p (resp. $\sigma_{\boldsymbol{p}}$) be equal to 1 if the ReLU of node p (resp. if path $\boldsymbol{p}$) is active for input $\boldsymbol{x}$, and 0 otherwise.

As previously noticed by Balduzzi et al. (2016) using the chain rule, we see that $\partial_x f_k$ is the sum of all $\omega_{\boldsymbol{p}}$ whose path is active, i.e. $\partial_x f_k(\boldsymbol{x}) = \sum_{\boldsymbol{p}\in\mathcal{P}(x,k)} \omega_{\boldsymbol{p}}\sigma_{\boldsymbol{p}}$. Consequently:

$$\begin{aligned}\mathbb{E}_{W,\sigma}\left[\partial_x f_k(\boldsymbol{x})^2\right] &= \sum_{\boldsymbol{p}\in\mathcal{P}(x,k)}\prod_{p\in\tilde{\boldsymbol{p}}} \mathbb{E}_W\left[w_p^2\right]\mathbb{E}_\sigma\left[\sigma_p^2\right] \\ &= |\mathcal{P}(x,k)|\prod_{p\in\tilde{\boldsymbol{p}}}\frac{2}{d_{p-1}}\frac{1}{2} = \prod_{p\in\tilde{\boldsymbol{p}}} d_p\cdot\prod_{p\in\tilde{\boldsymbol{p}}}\frac{1}{d_{p-1}} = \frac{1}{d}\,.\end{aligned} \tag{8}$$

The first equality uses H1 to decouple the expectations over weights and ReLUs, and then applies Lemma 9 of Appendix A.1, which uses H3-H5 to kill all cross-terms and take the expectation over weights inside the product. The second equality uses H3 and the fact that the resulting product is the same for all active paths. The third equality counts the number of paths from x to k and we conclude by noting that all terms cancel out, except d_{p-1} from the input layer which is d.

Step 2: Statistical properties of $\partial_k\mathcal{L}$ and $\partial_x\mathcal{L}$. Defining $q_k(\boldsymbol{x}) := \frac{e^{f_k(\boldsymbol{x})}}{\sum_{h=1}^{K} e^{f_h(\boldsymbol{x})}}$ (the probability of image x belonging to class k according to the network), we have, by definition of the cross-entropy loss, $\mathcal{L}(\boldsymbol{x},c) := -\log q_c(\boldsymbol{x})$, where c is the label of the target class. Thus:

$$\partial_k\mathcal{L}(\boldsymbol{x}) = \begin{cases} -q_k(\boldsymbol{x}) & \text{if } k\neq c \\ 1-q_c(\boldsymbol{x}) & \text{otherwise,}\end{cases} \quad \text{and}$$

$$\partial_x\mathcal{L}(\boldsymbol{x}) = (1-q_c)\,\partial_x f_c(\boldsymbol{x}) + \sum_{k\neq c} q_k\left(-\partial_x f_k(\boldsymbol{x})\right). \tag{9}$$

Using again Lemma 9, we see that the $\partial_x f_k(\boldsymbol{x})$ are K centered and uncorrelated variables. So $\partial_x\mathcal{L}(\boldsymbol{x})$ is approximately the sum of K uncorrelated variables with zero-mean, and its total variance is given by $\left((1-q_c)^2+\sum_{k\neq c} q_k^2\right)/d$. Hence the magnitude of $\partial_x\mathcal{L}(\boldsymbol{x})$ is $1/\sqrt{d}$ for all $\boldsymbol{x}$, so the ℓ_q-norm of the full input gradient is $d^{1/q-1/2}$. Equation 6 concludes. □

Remark 1. Equation 9 can be rewritten as

$$\partial_x\mathcal{L}(\boldsymbol{x}) = \sum_{k=1}^{K} q_k(\boldsymbol{x})\left(\partial_x f_c(\boldsymbol{x}) - \partial_x f_k(\boldsymbol{x})\right). \tag{10}$$

As the term $k=c$ disappears, the norm of the gradients $\partial_x\mathcal{L}(\boldsymbol{x})$ appears to be controlled by the total error probability. This suggests that, even without regularization, trying

to decrease the ordinary classification error is still a valid strategy against adversarial examples. It reflects the fact that when increasing the classification margin, larger gradients of the classifier's logits are needed to push images from one side of the classication boundary to the other. This is confirmed by Theorem 2.1 of Hein and Andriushchenko (2017). See also Equation 14 in Appendix B.

We now turn to a more general theorem, that covers a much wider class of nets and implies Theorem 3. To do so, we will use the following symmetry assumption on the neural connections. For a given path $\boldsymbol{p}$, let the *path-degree* $d_{\boldsymbol{p}}$ be the multiset of encountered in-degrees along path $\boldsymbol{p}$. For a fully connected network, this is the unordered sequence of layer-sizes preceding the last path-node, including the input-layer. Now consider the multiset $\{d_{\boldsymbol{p}}\}_{\boldsymbol{p}\in\mathcal{P}(x,o)}$ of all path-degrees when $\boldsymbol{p}$ varies among all paths from input x to output o. The symmetry assumption (relatively to o) is

($\mathcal{S}$) All input nodes x have the same multiset $\{d_{\boldsymbol{p}}\}_{\boldsymbol{p}\in\mathcal{P}(x,o)}$ of path-degrees from x to o.

Intuitively, this means that the statistics of degrees encountered along paths to the output are the same for all input nodes. This symmetry assumption is exactly satified by fully connected nets, almost satisfied by CNNs (up to boundary effects, which can be alleviated via periodic or mirror padding) and exactly satisfied by strided layers, if the layersize is a multiple of the stride.

Theorem 4 (Vulnerability of Feedforward Nets). *Consider any non-recurrent neural network with ReLU activation functions that satisfies assumptions ($\mathcal{H}$) and outputs logits $f_k(\boldsymbol{x})$ that get fed to the cross-entropy-loss $\mathcal{L}$. Then $\|\boldsymbol{\partial}_{\boldsymbol{x}} f_k\|_2$ is independent of the input dimension d. Moreover, if the net satifies symmetry assumption ($\mathcal{S}$), then the coordinates of $\boldsymbol{\partial}_{\boldsymbol{x}} f_k$ scale like $1/\sqrt{d}$ and Equation 7 still holds: $\|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q \propto d^{\frac{1}{q}-\frac{1}{2}}$ and $\epsilon_p \|\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}\|_q \propto \sqrt{d}$.*

Proof in Appendix A.1.

Corollary 5 (Vulnerability of CNNs). *Any succession of convolution and dense layers, strided or not, with ReLU activations, that satisfies assumptions ($\mathcal{H}$), and outputs logits that get fed to the cross-entropy-loss $\mathcal{L}$, has logit-coordinates that scale like $1/\sqrt{d}$ and satisfies Equation 7. In particular, it is increasingly vulnerable with growing input-resolution to attacks generated with any ℓ_p-norm.*

3.3. Effects of Strided and Average-Pooling Layers on Adversarial Vulnerability

It is common practice in CNNs to use average-pooling layers or strided convolutions to progressively decrease the number of pixels per channel. Corollary 5 shows that using strided convolutions does not protect against adversarial examples. However, how about if we replace strided convolutions by convolutions with stride 1 plus an average-pooling layer. An average-pooling introduces *deterministic* weights of size $1/(\text{pooling-window-size})$, so Theorem 4, which assumes random weights, does not apply anymore. Thus we may wonder: does this replacement help protecting against adversarial examples? The following theorem says it does.

Theorem 6 (Effect of Average-Poolings). *Consider any succession of possibly strided convolution and dense layers with ReLU activations satisfying assumptions ($\mathcal{H}$). Replacing a strided convolution that selects 1 neuron out of n by a convolution with stride 1 followed by an average-pooling over n neurons divides all output and loss gradients by n.*

Proof in Appendix A.2. The two previous statements suggest to try and replace any strided convolution by its non-strided counterpart, followed by an average-pooling layer. Furthermore, Theorem 6 shows that if we systematically reduce the number of pixels per channel down to 1 by combining convolutional with average-pooling layers, then the adversarial vulnerability to ℓ_∞ becomes independent of the input-resolution – provided that assumptions ($\mathcal{H}$) stay valid after training.

4. Empirical Results

In this section, we first empirically verify that both the average ℓ_1-norm of $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$ and the adversarial vulnerability grow like $\sqrt{d}$ with d the input dimension (Section 4.1) as predicted by Corollary 5. We then compare the loss-gradient regularization methods with adversarially-augmented training (Section 4.2) and finish by verifying the increased effectiveness of average-pooling over strided layers layers to decrease adversarial vulnerability (Section 4.3).

For all experiments, we only consider adversarial vulnerability to ℓ_∞-attacks, which we approximate using the FGSM-implementation of the python Foolbox-package (Rauber et al., 2017). For all test-attacks, we use $\epsilon = .006$. Our image datasets being globally normalized, for most images this perturbation is imperceptible. This *ϵ-threshold* should not be confused with the *regularization-strengths* ϵ appearing in (3) and (4), which will be varied in some experiments.

4.1. Vulnerability Grows with Input Resolution

Theorems 3-4 and Corollary 5 predict a linear growth of the average ℓ_1-norm of $\boldsymbol{\partial}_{\boldsymbol{x}}\mathcal{L}$ with the square root of the input dimension d, and therefore also of adversarial vulnerability (Lemma 1). To test these predictions, we created a 12-class dataset of approximately $80,000$ $256\times256\times3$-sized RGB-images by merging similar ImageNet-classes, resizing the smallest image-edge to 256 pixels and center-cropping the result. We then downsized the images to 32, 64, 128 and 256 pixels per edge, and trained 10 CNNs on each of these downsized datasets. We then computed adversarial

vulnerability and average $\|\partial_x \mathcal{L}\|_1$ for each network on a same held-out test-dataset. Figure 1 summarizes the results. The dashed-line follows the median of each group of 10 networks; the errorbars show the 10th and 90th quantiles. As predicted by our theorems, both $\|\partial_x \mathcal{L}\|_1$ and adversarial vulnerability grow almost linearly with $\sqrt{d}$.

The growth of $\epsilon \|\|\partial_x \mathcal{L}\|\|$ with d means that for large d we are outside the linear regime of Lemma 1. And indeed, this occurs at the largest resolution in our experiments: the cross-entropy loss $\mathcal{L}$ ranges from 0 to $\log 12 \approx 2.5$. Meanwhile, for $d = 3 \cdot 256 \cdot 256$ and $\epsilon = .006$, we see that $\mathbb{E}_x \|\partial_x \mathcal{L}\|_1 \approx 100$, which leads to $\epsilon \|\partial_x \mathcal{L}\|_1 \approx 2.7$. This is even bigger than the 2.5-range of $\mathcal{L}$. We are therefore way beyond the linear approximation validity.

All networks had exactly the same amount of parameters and very similar structure accross the various input-resolutions. The CNNs were a succession of 8 'convolution → batchnorm → ReLU' layers with 64 output channels, followed by a final full-connection to the 12 logit-outputs. We used 2×2-max-poolings after layers 2,4 and 6, and a final max-pooling after layer 8 that fed only 1 neuron per channel to the fully-connected layer. To ensure that the convolution-kernels cover similar ranges of the images accross each of the 32, 64, 128 and 256 input-resolutions, we respectively dilated all convolutions ('à trous') by a factor 1, 2, 4 and 8.

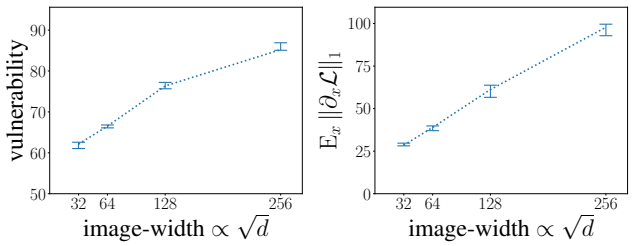

Figure 1. Both adversarial vulnerability (left) and $\mathbb{E}_x \|\partial_x \mathcal{L}\|_1$ (right) increase linearly with the square-root of the image-resolution d, as predicted by Corollary 5. Here, adversarial vulnerability gets dampened at higher dimension, because the first-order approximation made in Equation 2 becomes less and less valid.

4.2. Gradient Penalty and Adversarial Augmentation

In this section, we check the correspondance between gradient-regularization and adversarially-augmented training (Proposition 2), and compare these methods to another gradient-regularizer proposed by Hein and Andriushchenko (2017): the cross-Lipschitz regularizer. We train several CNNs with same architecture to classify CIFAR-10 images (Krizhevsky, 2009). For each net, we use a specific training method with a specific regularization value ϵ. The training methods used were ℓ_1- and ℓ_2-penalization of $\partial_x \mathcal{L}$ (Equation 3), adversarial augmentation with ℓ_∞- and ℓ_2- attacks (Equation 4) and the cross-Lipschitz regularizer (Equation 15 in Appendix B). Each network has 5 'convolution → batchnorm → ReLU' layers with 64 output-channels each, followed by an average-pooling that feeds only 1 neuron per channel to the final fully-connected linear layer. The results are summarized in Figure 2. Each point represents one net trained with a specific adversarial training method and a specific ϵ. Each curve groups all points of a same training method.

Illustration of Proposition 2. The upper row of Figure 2 plots $\mathbb{E}_x \|\partial_x \mathcal{L}_1\|$, adversarial vulnerability and accuracy as a function of $\epsilon\, d^{1/p}$. The excellent match between the adversarial augmentation curve with $p = \infty$ ($p = 2$) and its gradient-regularization dual counterpart with $q = 1$ (resp. $q = 2$) illustrates the duality between ϵ as a threshold for adversarially-augmented training and as a regularization constant in the regularized loss (Proposition 2).

Confirmation of Equation 5. Still on the upper row, the curves for $p = \infty, q = 1$ have no reason to match those for $p = 2, q = 2$ when plotted against ϵ, because ϵ is a threshold that is relative to a specific attack-norm. However, Equation 5 suggested that the rescaled thresholds $\epsilon d^{1/p}$ may approximately correspond to a same 'threshold-unit' accross ℓ_p-norms and accross dimension. This is well confirmed by the upper row plots: by rescaling the x-axis, the $p = q = 2$ and $q = 1 - 1/p = \infty$ curves get almost super-imposed.

$\mathbb{E}_x \|\partial_x \mathcal{L}\|$ measures adversarial vulnerability. Figure 2d shows that adversarial vulnerability is *almost a function* of $\mathbb{E}_x \|\partial_x \mathcal{L}\|_1$ which is independent of the training and regularization method used. It is a strikingly clear confirmation that adversarial examples are primarily caused by large gradients of the classifier as captured via the induced loss.

Adversarial regularization improves generalization. While adversarial vulnerability steadily decreases with growing ϵ (Figure 2b), Figure 2c shows that, whatever adversarial method used, the accuracy before attack on the held-out test set first increases (and eventually tumbles down). Decreasing the adversarial vulnerability thus improves generalization, which we recall was the original motivation for Drucker and LeCun (1991) to introduce double-backpropagation.

Accuracy-vs-Vulnerability Trade-Off. To concentrate on the accuracy versus vulnerability trade-offs, Figure 2f merges Figures 2b and 2c by taking out the irrelevant parameter ϵ. Following the curves from right to left now corresponds to implicitely increasing ϵ. The long horizontal plateaus confirm that adversarial vulnerability can be massively driven down without losing in generalization performance. While all methods perform equally well for small enought ϵ-values, on the long run, the best accuracy-to-vulnerability ratios are obtained with the traditional adversarially augmented training using FGSM ($p = \infty$). By

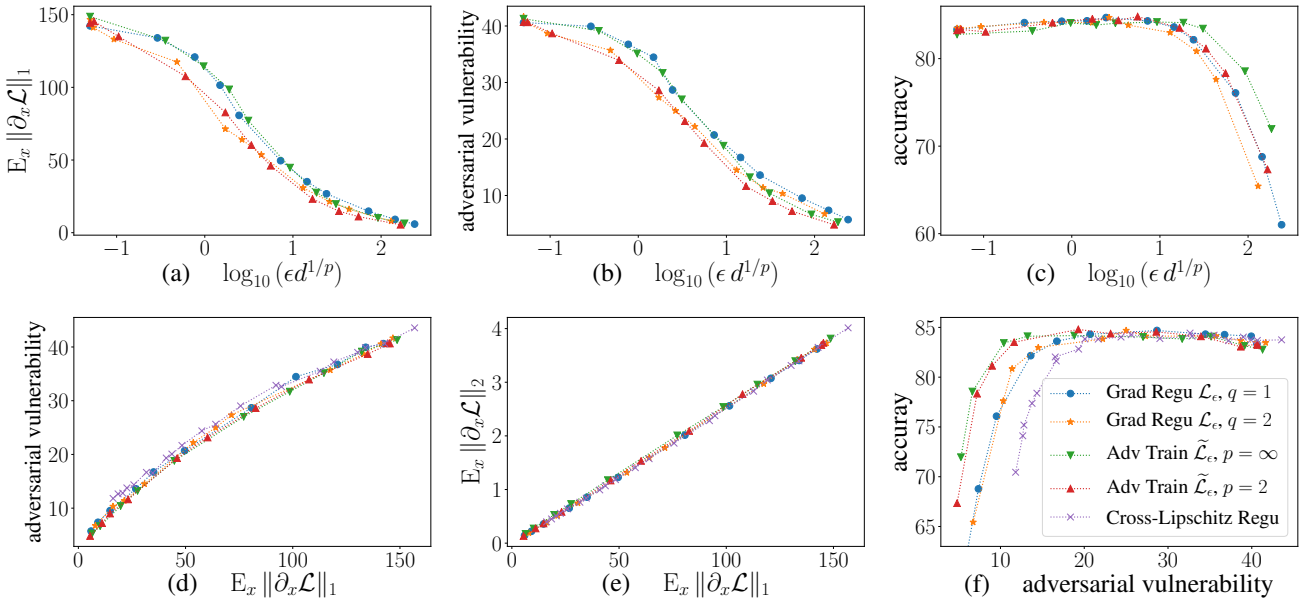

Figure 2. Average norm $\mathbb{E}_{\boldsymbol{x}} \|\boldsymbol{\partial}_{\boldsymbol{x}} \mathcal{L}\|$ of the loss-gradients, adversarial vulnerability and accuracy (before attack) of various networks trained with different adversarial regularization methods and regularization strengths ϵ. Each point represents a trained network, and each curve a training-method. *Upper row*: while accuracy first improves with rising ϵ, both the average gradient norms and adversarial vulnerability steadily decrease. A priori, the regularization parameter ϵ has different meanings for each method. The relatively good superposition of all curves in each upper-row plot illustrates (i) the dual-correspondance between adversarially-augmented training and gradient-loss penalization and (ii) confirms the rescaling of ϵ proposed in Equation 5. (d): adversarial vulnerability is almost a function of the average loss-gradient norm and does not depend on the training method used. It confirms that large gradients of the loss are the main cause of adversarial vulnerability. (f): This figure concentrates on the accuracy-vs-vulnerability trade-offs by taking out ϵ from Figures 2b and 2c . While all methods first perform similarly for small ϵ, the best ratios are eventually obtained by the adversarial-augmentation methods ($\widetilde{\mathcal{L}}$). (e): The almost perfect linear relation between the ℓ_1- and ℓ_2-norms of $\boldsymbol{\partial}_{\boldsymbol{x}} \mathcal{L}$ explain the striking similarity between the $q = 1$ and $q = 2$ curves on Figure 2f . It suggests that protecting against a given attack-norm also protects against others.

keeping ϵ inside the function $\mathcal{L}$ in Equation 4, $\tilde{\mathcal{L}}_{\epsilon,p}$ can also account for higher order Taylor-variation of the original loss $\mathcal{L}$, which might explain why these methods perform better for higher ϵ than their first-order counter-parts (Equation 3).

The penalty-norm does not matter. We were surprised to see that on Figures 2d and 2f , the $\mathcal{L}_{\epsilon,q}$ curves are almost identical for $q = 1$ and 2. This indicates that both norms can be used interchangeably in (3) (modulo proper rescaling of ϵ via Equation 5), and suggests that protecting agains a specific attack-norm also protects against others. Equation 6 may provide an explanation: if the coordinates of $\boldsymbol{\partial}_{\boldsymbol{x}} \mathcal{L}$ behave like centered, uncorrelated variables with equal variance –which follows from assumptions $(\mathcal{H})$ –, then the ℓ_1- and ℓ_2-norms of $\boldsymbol{\partial}_{\boldsymbol{x}} \mathcal{L}$ are simply proportional. Plotting $\mathbb{E}_x \|\boldsymbol{\partial}_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x})\|_2$ against $\mathbb{E}_x \|\boldsymbol{\partial}_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x})\|_1$ in Figure 2e confirms this explanation. The slope is independent of the training method. Therefore, penalizing the $\|\boldsymbol{\partial}_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x})\|_1$ during training will not only decrease $\mathbb{E}_{\boldsymbol{x}} \|\boldsymbol{\partial}_{\boldsymbol{x}} \mathcal{L}\|_1$ (as shown in Figure 2a), but also drive down $\mathbb{E}_{\boldsymbol{x}} \|\boldsymbol{\partial}_{\boldsymbol{x}} \mathcal{L}\|_2$ and vice-versa.

4.3. Averaging, Subsampling and Max-Pooling

Our theorems show that, contrary to strided layers, average-poolings should decrease adversarial vulnerability. We tested this hypothesis on CNNs trained on CIFAR-10, with 6 blocks of 'convolution $\rightarrow$ BatchNorm $\rightarrow$ReLU' with 64 output-channels, followed by a final average pooling feeding one neuron per channel to the last fully-connected linear layer. Additionally, after every second convolution, we placed a pooling layer with stride $(2, 2)$ (thus acting on 2×2 neurons at a time). We tested average-pooling, strided and max-pooling layers and trained 20 networks per architecture. Results are shown in Figure 3. As predicted, the networks with average pooling layers are much more robust to adversarial images than the others. Although all accuracies are very close, they are slightly better with average pooling than with striding, but slightly worse than with max-pooling.

5. Related Literature

Goodfellow et al. (2014) already stressed that adversarial vulnerability increases with growing dimension d. Their argument relies on a 'one-output-to-many-inputs'-model with

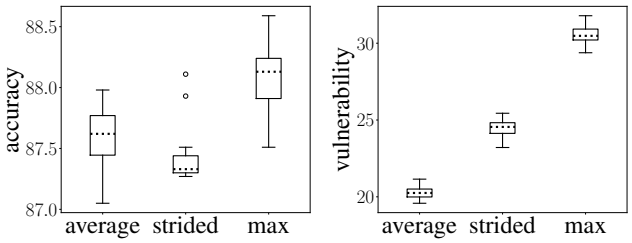

Figure 3. Compared effects of average-, strided- and max-pooling layers on the accuracy before attack (left) and adversarial vulnerability (right). As can be seen, average-pooling layers make networks more robust to adversarial examples, contrary to strided and max-pooling ones. This confirms Theorem 6.

dimension-independent weights. They therefore conclude on a linear growth of adversarial vulnerability with d and accuse our networks of being "too linear-like". Although this linear dependance becomes $\sqrt{d}$ when adjusting for a dimension-dependent weight-initialization, our theory and experiments nevertheless confirm this point of view, in the sense that a first-order Talyor expansion is indeed sufficient to explain the adversarial vulnerability of neural networks. As suggested by the one-output-to-many-inputs model, the culprit is that growing dimensionality gives the adversary more and more room to 'wriggle around' with the noise and adjust to the gradient of the output neuron. One of the contributions here however is to show that this wriggeling is still possible when the output is connected to all inputs only indirectly, even when no neuron is directly connected to all inputs, like in CNNs.

Incidently, Goodfellow et al. (2014) also already relate adversarial vulnerability to large gradients of the loss $\mathcal{L}$, an insight at the very heart of their FGSM-algorithm. They however do not propose any explicit penalizer on the gradient of $\mathcal{L}$ other than indirectly through adversarially-augmented training; possibly because the main deep learning libraries did not support automatized backpropation through gradients at that time. While our penalty stems from an approximation of the *relevant* adversarial vulnerability, Hein and Andriushchenko (2017) derived yet another gradient-based penalty –the *cross-Lipschitz*-penalty– by considering (and proving) formal guarantees on the *total* adversarial vulnerability. While both penalties are similar in spirit, focusing on the relevant adversarial vulnerability has two main advantages. First, it archieves better accuracy-to-vulnerability ratios, both in theory and practice, because it ignores class-switches between misclassified examples and penalizes only those that reduce the accuracy. Second, it allows to deal with one number only, $\Delta\mathcal{L}_{0/1}$ or $\Delta\mathcal{L}$, whereas Hein and Andriushchenko's cross-Lipschitz regularizer and theoretical guarantees explicitely involve *all* K logit-functions (and their gradients). See Appendix B. Penalizing network-gradients is also at the heart of contractive auto-encoders as proposed by Rifai et al. (2011), where it is used to regularize the encoder-features. Seeing adversarial training as a generalization method, let us also mention Hochreiter and Schmidhuber (1995), who propose to enhance generalization by searching for parameters in a "flat minimum region" of the loss. This leads to a penalty involving the gradient of the loss, but taken with respect to the weights, rather than the inputs. In the same vein, a gradient-regularization of the loss of generative models also appears in Proposition 6 of Ollivier (2014), where it stems from a codelength bound on the data (minimum description length). Finally, Cisse et al. (2017) propose new network-architectures that have small gradients by design, rather than by special training: an approach that makes all the more sense, considering the conclusion of Theorems 3 and 4. For further details and references on adversarial attacks and defenses, we refer to Yuan et al. (2017).

6. Conclusion

We first showed that adversarial vulnerability increases with the gradients $\boldsymbol{\partial}_x\mathcal{L}$ of the loss, which is confirmed by the near-perfect functional relationship between gradient norms and vulnerability (Figure 2d). We then evaluated the size of $\|\boldsymbol{\partial}_x\mathcal{L}\|_q$ and showed that usual convolutional or fully connected architectures are more and more vulnerable to ℓ_p-attacks with growing input dimension d (the image-size). While using strided convolutions does not help, using sufficiently many average-poolings may significantly robustify the network. Our results rely on the statistical weight-distribution at initialization, but our experiments confirm their conclusions even after training. We also proposed to regularize the training by penalizing the (ℓ_1-)norm of $\boldsymbol{\partial}_x\mathcal{L}$. Like Tikhonov-regularization being approximately equivalent to training with random noise, we showed that our penalization is equivalent to training with adversarial noise, both theoretically at first order, and empirically (Figure 2). We thereby linked double-backpropagation to FGSM-augmented training. However, even by combining all these tricks, the networks remain surprisingly vulnerable to adversarial attacks, which suggests that on the long run, we may need to design new network architectures that are inherently more resolution-invariant.

Acknowledgements

We thank Martín Arjovsky, Ilya Tolstikhin and Diego Fioravanti for helpful discussions.

References

D. Balduzzi, B. McWilliams, and T. Butler-Yeoman. Neural Taylor Approximations: Convergence and Exploration in Rectifier Networks. *arXiv:1611.02345*, 2016. arXiv:

1611.02345.

C. M. Bishop. Training with noise is equivalent to Tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

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

H. Drucker and Y. LeCun. Double backpropagation increasing generalization performance. In *International Joint Conference on Neural Networks*, 1991.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572*, 2014.

K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852*, 2015.

M. Hein and M. Andriushchenko. Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation. *arXiv:1705.08475*, 2017.

S. Hochreiter and J. Schmidhuber. Simplifying neural nets by discovering flat minima. In *NIPS*, 1995.

J. Huang. *Statistics of Natural Images and Models*. PhD thesis, Brown University, Providence, RI, 2000.

A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, 2009.

Y. Ollivier. Auto-encoders: reconstruction versus compression. *arXiv:1403.7752*, 2014.

J. Rauber, W. Brendel, and M. Bethge. Foolbox v0.8.0: A Python toolbox to benchmark the robustness of machine learning models. *arXiv:1707.04131*, 2017.

S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive Auto-encoders: Explicit Invariance During Feature Extraction. In *ICML*, 2011.

X. Yuan, P. He, Q. Zhu, R. R. Bhat, and X. Li. Adversarial Examples: Attacks and Defenses for Deep Learning. *arXiv:1712.07107*, 2017.

A. Proofs

A.1. Proof of Theorem 4

The proof of Theorem 4 is very similar to the one of Theorem 3, but we will need to first generalize the equalities appearing in Equation 8. To do so, we identify the computational graph of a neural network to an abstract Directed Acyclic Graph (DAG) which we use to prove the needed algebraic equalities. We then concentrate on the statistical weight-interactions implied by assumption $(\mathcal{H})$, and finally throw these results together to prove the theorem. In all the proof, o will designate one of the output-logits $f_k(\boldsymbol{x})$.

Lemma 7. *Let $\boldsymbol{x}$ be the vector of inputs to a given DAG, o be any leaf-node of the DAG, x a generic coordinate of $\boldsymbol{x}$. Let $\boldsymbol{p}$ be a path from the set of paths $\mathcal{P}(x,o)$ from x to o, $\tilde{\boldsymbol{p}}$ the same path without node x, p a generic node in $\tilde{\boldsymbol{p}}$, and d_p be its input-degree. Then:*

$$\sum_{x\in\boldsymbol{x}}\sum_{\tilde{\boldsymbol{p}}\in\mathcal{P}(x,o)}\prod_{p\in\tilde{\boldsymbol{p}}}\frac{1}{d_p}=1 \tag{11}$$

Proof. We will reason on a random walk starting at o and going up the DAG by choosing any incoming node with equal probability. The DAG being finite, this walk will end up at an input-node x with probability 1. Each path $\boldsymbol{p}$ is taken with probability $\prod_{p\in\tilde{\boldsymbol{p}}}\frac{1}{d_p}$. And the probability to end up at an input-node is the sum of all these probabilities, i.e. $\sum_{x\in\boldsymbol{x}}\sum_{\boldsymbol{p}\in\mathcal{P}(x,o)}\prod_{p\in\boldsymbol{p}}d_p^{-1}$, which concludes. □

The sum over all inputs x in Equation 11 being 1, on average it is $1/d$ for each x, where d is the total number of inputs (i.e. the length of $\boldsymbol{x}$). It becomes an equality under assumption $(\mathcal{S})$:

Lemma 8. *Under the symmetry assumption $(\mathcal{S})$, and with the previous notations, for any input $x\in\boldsymbol{x}$:*

$$\sum_{\boldsymbol{p}\in\mathcal{P}(x,o)}\prod_{p\in\tilde{\boldsymbol{p}}}\frac{1}{d_p}=\frac{1}{d}\,. \tag{12}$$

Proof. Let us denote $\mathcal{D}(x,o):=\{d_{\boldsymbol{p}}\}_{x\in\mathcal{P}(x,o)}$. Each path $\boldsymbol{p}$ in $\mathcal{P}(x,o)$ corresponds to exactly one element $d_{\boldsymbol{p}}$ in $\mathcal{D}(x,o)$ and vice-versa. And the elements d_p of $d_{\boldsymbol{p}}$ completely determine the product $\prod_{p\in\tilde{\boldsymbol{p}}}d_p^{-1}$. By using Equation 11 and the fact that, by $(\mathcal{S})$, the multiset $\mathcal{D}(x,o)$ is independent of x, we hence conclude

$$\begin{aligned}\sum_{x\in\boldsymbol{x}}\sum_{\boldsymbol{p}\in\mathcal{P}(x,o)}\prod_{p\in\tilde{\boldsymbol{p}}}\frac{1}{d_p}&=\sum_{x\in\boldsymbol{x}}\sum_{d_{\boldsymbol{p}}\in\mathcal{D}(x,o)}\prod_{d_p\in d_{\boldsymbol{p}}}\frac{1}{d_p}\\&=d\sum_{d_{\boldsymbol{p}}\in\mathcal{D}(x,o)}\prod_{d_p\in d_{\boldsymbol{p}}}\frac{1}{d_p}=1\,.\end{aligned}$$ □

Now, let us relate these considerations on graphs to gradients and use assumptions $(\mathcal{H})$. We remind that path-product $\omega_{\boldsymbol{p}}$ is the product $\prod_{p\in\tilde{\boldsymbol{p}}}w_p$.

Lemma 9. *Under assumptions $(\mathcal{H})$, each path-product has expectation 0, two distinct path-products $(\omega_{\boldsymbol{p}},\omega_{\boldsymbol{p}'})$ decorrelate, and the variance of a path-product is the product of its variances:*

$$\begin{aligned}\mathbb{E}_W\left[\omega_{\boldsymbol{p}}\right]&=\mathbb{E}_W\left[\omega_{\boldsymbol{p}}\,\omega_{\boldsymbol{p}'}\right]=0\\\mathbb{E}_W\left[\omega_{\boldsymbol{p}}^2\right]&=\prod_{p\in\tilde{\boldsymbol{p}}}\mathbb{E}_W\left[w_p^2\right]\,.\end{aligned}$$

Proof. Applying H4 then H3 shows that $\mathbb{E}_W\left[\omega_{\boldsymbol{p}}\right]=0$. Now, take two different paths $\boldsymbol{p}$ and $\boldsymbol{p}'$ that end at a same node o. Going back from o, consider the first node p at which $\boldsymbol{p}$ and $\boldsymbol{p}'$ part. This gives two different incoming weights w_p on $\boldsymbol{p}$ and w'_p on $\boldsymbol{p}'$ which are in a same layer. Then:

$$\mathbb{E}_W\left[\omega_{\boldsymbol{p}}\,\omega_{\boldsymbol{p}'}\right]=\mathbb{E}_W\left[\omega_{\boldsymbol{p}\setminus p}\,\omega_{\boldsymbol{p}'\setminus p}\right]\mathbb{E}_W\left[w_p\,w'_p\right]=0\,,$$

where the first equality uses H4 and the second uses H5 and the fact that w_p and w'_p have 0 expectation by H3. Moreover

$$\mathbb{E}_W\left[\omega_{\boldsymbol{p}}^2\right]=\mathbb{E}_W\left[\prod_{p\in\tilde{\boldsymbol{p}}}w_p^2\right]=\prod_{p\in\tilde{\boldsymbol{p}}}\mathbb{E}_W\left[w_p^2\right]\,,$$

where the second equality uses H4. □

We now have all elements to prove Theorem 4.

Proof. (**of Theorem 4**) For a given neuron p in $\tilde{\boldsymbol{p}}$, let $p-1$ designate the previous node in $\boldsymbol{p}$ of p. Let σ_p (resp. $\sigma_{\boldsymbol{p}}$) be a variable equal to 0 if neuron p gets killed by its ReLU (resp. path $\boldsymbol{p}$ is inactive), and 1 otherwise. Then:

$$\partial_x o\;=\sum_{\boldsymbol{p}\in\mathcal{P}(x,o)}\prod_{p\in\tilde{\boldsymbol{p}}}\partial_{p-1}\,p\;=\sum_{\boldsymbol{p}\in\mathcal{P}(x,o)}\omega_{\boldsymbol{p}}\,\sigma_{\boldsymbol{p}}$$

Consequently:

$$\begin{aligned}\mathbb{E}_{W,\sigma}\left[(\partial_x o)^2\right]&=\sum_{\boldsymbol{p},\boldsymbol{p}'\in\mathcal{P}(x,o)}\mathbb{E}_W\left[\omega_{\boldsymbol{p}}\,\omega_{\boldsymbol{p}'}\right]\mathbb{E}_\sigma\left[\sigma_{\boldsymbol{p}}\sigma_{\boldsymbol{p}'}\right]\\&=\sum_{\boldsymbol{p}\in\mathcal{P}(x,o)}\prod_{p\in\tilde{\boldsymbol{p}}}\mathbb{E}_W\left[\omega_p^2\right]\mathbb{E}_\sigma\left[\sigma_p^2\right]\\&=\sum_{\boldsymbol{p}\in\mathcal{P}(x,o)}\prod_{p\in\tilde{\boldsymbol{p}}}\frac{2}{d_p}\,\frac{1}{2}=\frac{1}{d}\,,\end{aligned} \tag{13}$$

where the firs line uses the independence between the ReLU killings and the weights (H1), the second uses Lemma 9 and the last uses Lemma 8. The gradient $\boldsymbol{\partial_x} o$ thus has coordinates whose squared expectations scale like $1/d$. Thus each coordinate scales like $1/\sqrt{d}$ and $\|\boldsymbol{\partial_x} o\|_q$ like $d^{1/2-1/q}$.

Conclude on $\|\partial_{\boldsymbol{x}}\mathcal{L}\|_q$ and $\epsilon_p \|\partial_{\boldsymbol{x}}\mathcal{L}\|_q$ by using Step 2 of the proof of Theorem 3.

Finally, note that, even without the symmetry assumption $(\mathcal{S})$, we have:

$$\mathbb{E}_W\left[\|\partial_{\boldsymbol{x}} o\|_2^2\right] = \sum_{x\in\boldsymbol{x}} \sum_{\boldsymbol{p}\in\mathcal{P}(x,o)} \prod_{p\in\tilde{\boldsymbol{p}}} \frac{2}{d_p}\frac{1}{2} = 1$$

Thus, even without $(\mathcal{S})$, $\|\partial_{\boldsymbol{x}} o\|_2$ is independent of the input-dimension d. □

A.2. Proof of Theorem 6

Proof. Pick any strided convolution that selects only 1 neuron out of n, replace it by a convolution with stride 1 plus average-pooling and consider Equation 13. Going from the first to the second line uses Lemma 9, which does not apply because the average-pooling weights obviously do not satisfy H3-H5. Instead, we apply Lemma 10 –a variation of Lemma 9 proven below–, and proceed with the second line. Now there are n times more neurons after the convolution, and therefore n times more paths in each $\mathcal{P}(x,o)$, which multiplies the overall variance by n. But the average pooling divides each path-product $\omega_{\boldsymbol{p}}$ by n, which reduces each path's variance by n^2. Applying the two last lines of Equation 13, we see that the overall variance thus gets multiplied by $n/n^2 = 1/n$, and the average coordinate size therefore by $1/\sqrt{n}$. □

Lemma 10 (Variation of Lemma 9)**.** *Consider any succession of possibly strided convolution and dense layers with ReLU activations satisfying assumptions $(\mathcal{H})$. Replace an arbitrary number of strided convolutions by the corresponding non-strided convolution and average-pooling layer. Then, for any two different paths $\boldsymbol{p}, \boldsymbol{p}' \in \mathcal{P}(x,o)$ from an input-coordinate x to an output o, the path-products $\omega_{\boldsymbol{p}}, \omega'_{\boldsymbol{p}}$ satisfy:*

$$\mathbb{E}_W[\omega_{\boldsymbol{p}}] = \mathbb{E}_W[\omega_{\boldsymbol{p}}\,\omega_{\boldsymbol{p}'}] = 0$$
$$\mathbb{E}_W\left[\omega_{\boldsymbol{p}}^2\right] = \prod_{p\in\tilde{\boldsymbol{p}}} \mathbb{E}_W\left[w_p^2\right] .$$

Proof. All path-products are a product of variables satisfying assumptions $(\mathcal{H})$, divided by a product of constants, which correspond to the different average-pooling weights. Therefore, like in Lemma 9: $\mathbb{E}_W[\omega_{\boldsymbol{p}}] = 0$ and $\mathbb{E}_W\left[\omega_{\boldsymbol{p}}^2\right] = \prod_{p\in\tilde{\boldsymbol{p}}} \mathbb{E}_W\left[w_p^2\right]$. The difference with Lemma 9 is that, by introducing average-pooling layers, we introduced paths that may have *exactly* the same path-products: the reasoning we did in Lemma 9 –"going back from o, consider the first node p at which $\boldsymbol{p}$ and $\boldsymbol{p}'$ part..."– does not work, because this node p could be an average-pooling, in which case we do not get two different weights w_p, w'_p. But starting from o now works: consider the two first nodes $p \in \boldsymbol{p}$ and $p' \in \boldsymbol{p}'$ that do not both belong to $\boldsymbol{p}$ and $\boldsymbol{p}'$. Their weights w_p and $w_{p'}$ are in a same layer, because we assume that the architecture is a simple succession of layers; and they cannot be average-pooling weights, because average-poolings cannot separate one incoming path into two or more paths. Thus w_p and $w_{p'}$ satisfy H3-H5, so we get, as in the proof of Lemma 9:

$$\mathbb{E}_W[\omega_{\boldsymbol{p}}\,\omega_{\boldsymbol{p}'}] = \mathbb{E}_W\left[\omega_{\boldsymbol{p}\setminus p}\,\omega_{\boldsymbol{p}'\setminus p'}\right]\mathbb{E}_W[w_p\,w_{p'}] = 0\,. \quad \square$$

B. Comparison to the Cross-Lipschitz Regularizer

In their Theorem 2.1, Hein and Andriushchenko show that the minimal $\epsilon = \|\boldsymbol{\delta}\|_p$ perturbation to fool the classifier must be bigger than:

$$\min_{k\neq c} \frac{f_c(\boldsymbol{x}) - f_k(\boldsymbol{x})}{\max_{y\in B(x,\epsilon)} \|\partial_{\boldsymbol{x}} f_c(y) - \partial_{\boldsymbol{x}} f_k(y)\|_q}\,. \tag{14}$$

They argue that the training procedure typically already tries to maximize $f_c(\boldsymbol{x}) - f_k(\boldsymbol{x})$, thus one only needs to additionnally ensure that $\|\partial_{\boldsymbol{x}} f_c(\boldsymbol{x}) - \partial_{\boldsymbol{x}} f_k(\boldsymbol{x})\|_q$ is small. They then introduce what they call a Cross-Lipschitz Regularization, which corresponds to the case $p = 2$ and involves the gradient differences between *all* classes:

$$\mathcal{R}_{\text{xLip}} := \frac{1}{K^2}\sum_{k,h=1}^{K} \|\partial_{\boldsymbol{x}} f_h(\boldsymbol{x}) - \partial_{\boldsymbol{x}} f_k(\boldsymbol{x})\|_2^2 \tag{15}$$

In contrast, using (10), (the square of) our proposed regularizer $\|\partial_{\boldsymbol{x}}\mathcal{L}\|_q$ from Equation 3 can be rewritten, for $p = q = 2$ as:

$$\mathcal{R}_{\|\cdot\|_2}(f) = \sum_{k,h=1}^{K} q_k(\boldsymbol{x}) q_h(\boldsymbol{x}) \left(\partial_{\boldsymbol{x}} f_c(\boldsymbol{x}) - \partial_{\boldsymbol{x}} f_k(\boldsymbol{x})\right)\cdot \cdot \left(\partial_{\boldsymbol{x}} f_c(\boldsymbol{x}) - \partial_{\boldsymbol{x}} f_h(\boldsymbol{x})\right) \tag{16}$$

Although both (15) and (16) consist in K^2 terms, corresponding to the K^2 cross-interaction between the K classes, the big difference is that while in (15) all classes play exactly the same role, in (16) the summands all refer to the target class c in at least two different ways. First, all gradient differences are always taken with respect to $\partial_{\boldsymbol{x}} f_c$. Second, each summand is weighted by the probabilities $q_k(\boldsymbol{x})$ and $q_h(\boldsymbol{x})$ of the two involved classes, meaning that only the classes with a non-negligeable probability get their gradient regularized. This reflects the idea that only points near the margin need a gradient regularization, which incidentally will make the margin sharper.

C. Perception Threshold

To keep the average pixel-wise variation constant accross dimensions d, we saw in Equation 5 that the threshold ϵ_p of

an ℓ_p-attack should scale like $d^{1/p}$. We will now see another justification for this scaling. Suppose that given an ℓ_p-attack norm, we want to choose ϵ_p such that the signal-to-noise ratio (SNR) $\|\boldsymbol{x}\|_2 / \|\boldsymbol{\delta}\|_2$ of a perturbation $\boldsymbol{\delta}$ with ℓ_p-norm $<= \epsilon_p$ is never greater than a given SNR threshold $1/\epsilon$. For $p = 2$ this imposes $\epsilon_2 = \epsilon \|\boldsymbol{x}\|_2$. More generally, studying the inclusion of ℓ_p-balls in ℓ_2-balls yields

$$\epsilon_p = \epsilon \|\boldsymbol{x}\|_2 d^{1/p-1/2} . \tag{17}$$

Note that this gives again $\epsilon_p = \epsilon_\infty d^{1/p}$. This explains how to adjust the threshold ϵ with varying ℓ_p-attack norm.

Now, let us see how to adjust the threshold of a given ℓ_p-norm when the dimension d varies. Suppose that $\boldsymbol{x}$ is a natural image and that decreasing its dimension means either decreasing its resolution or cropping it. Because the statistics of natural images are approximately resolution and scale invariant (Huang, 2000), in either case the average squared value of the image pixels remains unchanged, which implies that $\|\boldsymbol{x}\|_2$ scales like $\sqrt{d}$. Pasting this back into Equation 17, we again get:

$$\epsilon_p = \epsilon_\infty d^{1/p} .$$

In particular, $\epsilon_\infty \propto \epsilon$ is a dimension-free number, exactly like in Equation 5 of the main part.

Now, why did we choose the SNR as our invariant reference quantity and not anything else? One reason is that it corresponds to a physical power ratio between the image and the perturbation, which we think the human eye is sensible to. Of course, the eye's sensitivity also depends on the spectral frequency of the signals involved, but we are only interested in orders of magnitude here.

D. A Variant of Adversarially-Augmented Training

In usual adversarially-augmented training, the adversarial image $\boldsymbol{x} + \boldsymbol{\delta}$ is generated on the fly, but is nevertheless treated as a fixed input of the neural net, which means that the gradient does not get backpropagated through $\boldsymbol{\delta}$. This need not be. As $\boldsymbol{\delta}$ is itself a function of $\boldsymbol{x}$, the gradients could actually also be backpropagated through $\boldsymbol{\delta}$. As it was only a one-line change of our code, we used this opportunity to test this variant of adversarial training (FGSM-variant in Figure 2) and thank Martín Arjovsky for suggesting it. But except for an increased computation time, we found no significant difference compared to usual augmented training.