Adversarial Robustness is at Odds with Lazy Training

Yunjuan Wang Enayat Ullah Poorya Mianjy Raman Arora

Summarised: 08.01.2024 by Partow Moradi

1 Justification

Recent research has highlighted the vulnerability of random neural networks to adversarial attacks, showing that a single gradient descent step can easily find such vulnerabilities. This study goes further by demonstrating that networks trained in the "lazy regime," despite maintaining weights close to initialization and achieving low generalization error, are still susceptible to adversarial examples. This challenges the prevailing notion that networks in the lazy regime are efficiently learnable, indicating a need for further examination of neural network security.It has already uncovered that neural networks are highly vulnerable to inference-time attacks, where adversaries can manipulate data during prediction. These attacks, demonstrated across various domains including computer vision and natural language processing, can cause misclassifications even with imperceptible perturbations added to inputs.Adversarial examples pose a significant threat to the security of neural network-based models in real-world systems.While neural networks perform well on clean data, they are alarmingly vulnerable to strategically induced perturbations. Early research by Szegedy et al. [2013] introduced adversarial examples, prompting subsequent studies by Goodfellow et al. [2014], Papernot et al. [2016], Madry et al. [2017], and Carlini and Wagner [2017a,b]. Despite efforts to develop defense algorithms, many are easily bypassed by stronger attacks [Carlini and Wagner, 2017a; Carlini et al., 2019; Tramer et al., 2020].

Recent theoretical work, inspired by Shamir et al. [2019], demonstrated that both small and large networks are susceptible to adversarial attacks, even with single-step gradient descent methods [Daniely and Schacham, 2020; Bubeck et al., 2021].In their study, Bartlett et al. [2021] extend the findings to random multilayer ReLU networks, but fail to explain why adversarial examples persist in neural networks trained with stochastic gradient descent, despite their ability to generalize well.

Adversarial examples can be found for random neural networks using a single step of gradient ascent, and this vulnerability extends to overparametrized neural networks that are provably efficiently learnable in lazy training. Adversarial Examples are inputs to a neural network

that have been intentionally perturbed in a manner imperceptible to humans but can cause the network to make incorrect predictions. Adversarial examples challenge the robustness and reliability of neural networks.Also gradient ascent is an optimization technique used to maximize a given objective function by iteratively adjusting the parameters in the direction of the gradient. In the context of adversarial attacks, adversaries use gradient ascent to perturb inputs in a way that maximizes the loss function, causing misclassification and Lazy training refers to a regime where neural networks are trained with minimal parameter updates, often maintaining their weights close to initialization. This approach has been shown to lead to networks with small generalization error.

The Lazy training is something that is first studied for neural network under the name of tangent Kernel. It says that if you run gradient descent on a very wide neural network and the neural network is actually going to behave like a kernel and you can prove that the neurons will not move very far , so kind of each neuron after you randomly initialize them is going to move within a local neighborhood and one can analyze the dynamics of training in this neural tangent kernel region. later people have observed that very similar phenomenan would happen in many different settings of training. lazy training regime appears when you have a very large initialization and whatever you do is similar to a kernel.ªlazy trainingº phenomenon is due to a choice of scaling that makes the model behave as its linearization around the initialization. Interestingly, linearity has been hypothesized as key reason for existence of adversarial examples in neural networks.

2 Main Point

This paper aims to address that gap by demonstrating that adversarial examples can be easily found with just a single gradient step for neural networks trained using first-order methods, focusing specifically on over-parametrized two-layer ReLU networks. In the "lazy regime" of training, where networks remain locally linear, computational guarantees for optimization and generalization exist. However, the study reveals that even networks trained in this regime are vulnerable to adversarial attacks, as a perturbation of certain size in the direction of the gradient can alter predictions significantly. Empirical validation confirms the theoretical findings. Adversarial examples can be found easily using a single step of gradient ascent for neural networks trained using first-order methods.The paper focuses on over-parametrized two-layer ReLU. (size of the network is polynomial in the size of the training sample).and the weights of two-layer ReLU networks stay close to initialization throughout the training of the network using gradient descent-based methods and shows that such trained neural networks still suffer from adversarial examples which can be found using a single step of gradient ascent.this is the first work which shows that networks with small generalization error are still vulnerable to adversarial attacks.

in the direction of the gradient suffices to flip the prediction of two-layer ReLU networks trained in the lazy regime – these are networks with all weights at a distance of $O(\sqrt{\frac{1}{m}})$ from their initialization, where m is the width of the network. To the best of our knowledge, this is the first work which shows that networks with small generalization error are still vulnerable to adversarial attacks. Further, They validate Their theory empirically and they confirm that a perturbation of size $O\left(\frac{1}{\sqrt{d}}\right)$ suffices to generate a strong adversarial example for a trained two-layer ReLU network.Most related to their work are those on proving the existence of adversarial examples on random neural networks. this paper shows that independent of the choice of the training algorithm, all neural networks trained in the lazy regime remain vulnerable to adversarial attacks. the investigation focuses on the settings where in the weights of the trained neural network stay close to the initialization.]In particular, They show that an adversarial perturbation of size

$$O\left(\frac{\|x\|}{\sqrt{d}}\right)$$

in the direction of the gradient suffices to flip the prediction of two-layer ReLU networks trained in the lazy regime – these are networks with all weights at a distance of $O(\sqrt{\frac{1}{m}})$ from their initialization, where m is the width of the network. To the best of our knowledge, this is the first work which shows that networks with small generalization error are still vulnerable to adversarial attacks. Further, They validate Their theory empirically and they confirm that a perturbation of size $O\left(\frac{1}{\sqrt{d}}\right)$ suffices to generate a strong adversarial example for a trained two-layer ReLU network.Most related to their work are those on proving the existence of adversarial examples on random neural networks. this paper shows that independent of the choice of the training algorithm, all neural networks trained in the lazy regime remain vulnerable to adversarial attacks. the investigation focuses on the settings where in the weights of the trained neural network stay close to the initialization.

3 Preliminaries

Let's break down the notation used in the paper:

1. Scalars: Denoted by lowercase italics, e.g., u.
2. Vectors: Denoted by lowercase bold, e.g., $\mathbf{u}$.
3. Matrices: Denoted by uppercase bold Roman letters, e.g., $\mathbf{U}$.
4. Set notation: $[m]$ denotes the set $\{1,2,\ldots,m\}$.
5. Norm notation:$\|.\|$ or $\|.\|_2$ denotes the L^2 norm or Euclidean norm.
6. For a matrix U = [u$_1, \ldots, u_m] \in R^{d\times m}$, $\|\mathbf{U}\|_2, \infty = \max_{s\in[m]} \|\mathbf{u}_s\| This expression describes the L^2$ to L^∞ operator norm of a matrix $\mathbf{U}$, denoted as $\|\mathbf{U}\|_{2,\infty}$. Here's what it means:

$$\mathbf{U} = [u_1, \ldots, u_m] \in R^{d\times m}$$

is a matrix with d rows and m columns. Each column vector $\mathbf{u}_s$ represents one of the m columns of the matrix. The operator norm $\|\cdot\|_{2,\infty}$ is defined as the maximum L^2 norm of the column vectors of the matrix. It measures how much the matrix stretches vectors in Euclidean space. Mathematically, it is defined as

$$\|\mathbf{U}\|_{2,\infty} = \max_{s\in[m]} \|\mathbf{u}_s\|,$$

where $\|\mathbf{u}_s\|$ represents the L^2 norm (or Euclidean norm) of the s-th column vector. In other words, $\|\mathbf{U}\|_{2,\infty}$ gives you the maximum length of any column vector in $\mathbf{U}$ when measured with the L^2 norm. It provides insight into how "big" the matrix can make vectors when acting on them.

7. $B_2(\mathbf{u}, R)$ denotes the L_2 ball centered at $\mathbf{u}$ of radius R.

8. For any $\mathbf{U} \in R^{d\times m}$, they use $B_{2,\infty}(\mathbf{U}, R) = \{\mathbf{U}_0 \in R^{d\times m} \mid \|\mathbf{U}_0 - \mathbf{U}\|_{2,\infty} \leq R\}$ to denote the $L^{2,\infty}$ ball centered at $\mathbf{U}$ of radius R. The expression defines a set in the context of matrix norms:

   - $\mathbf{U} \in R^{d\times m}$ denotes a matrix with d rows and m columns.
   - $B_{2,\infty}(\mathbf{U}, R)$ represents a set of matrices $\mathbf{U}_0$ in $R^{d\times m}$.
   - $\|\mathbf{U}_0 - \mathbf{U}\|_{2,\infty}$ denotes the $L^{2,\infty}$ norm between matrices $\mathbf{U}_0$ and $\mathbf{U}$.
   - The set $B_{2,\infty}(\mathbf{U}, R)$ includes all matrices $\mathbf{U}_0$ such that the $L^{2,\infty}$ norm between $\mathbf{U}_0$ and $\mathbf{U}$ is less than or equal to R.

   In simpler terms, $B_{2,\infty}(\mathbf{U}, R)$ represents the collection of matrices in $R^{d\times m}$ that are within a certain distance (measured using the $L^{2,\infty}$ norm) from the matrix $\mathbf{U}$. The distance is bounded by R. This concept is often used in the study of matrix norms and their properties.

9. For any function $f : R^d \to R$, ∇f denotes the gradient vector. The gradient vector represents the vector of partial derivatives of the function f with respect to each of its input variables.

10. Standard normal distribution: $N(0,1)$.

11. Standard multivariate normal distribution: $N(\mathbf{0}, \mathbf{I}_d)$, where $\mathbf{0}$ is the zero vector and $\mathbf{I}_d$ is the identity matrix of size $d \times d$.

12. S^{d-1} denotes the unit sphere in d dimensions. This notation represents the set of all points in d-dimensional space that are a unit distance away from the origin.

13. Standard O-notation: O and Ω.

    O notation (Big O notation) is used to denote the upper bound or worst-case scenario of the growth rate of a function.

    Ω notation (Big Omega notation) is used to denote the lower bound or best-case scenario of the growth rate of a function.

4 Problem Setup

Let $X \subseteq R^d$ and Y denote the input space and the label space, respectively. In this paper, They focus on the binary classification setting where $Y = \{-1, +1\}$. Assume that the data (x, y) is drawn from an unknown joint distribution D on $X \times Y$. For a function $f_w : X \to Y$ parameterized by w in some parameter space W, the generalization error captures the probability that f_w makes a mistake on a sample drawn from D:

$$generalizationerror := P(x, y) \sim D(yf_w(x) \leq 0).$$

This measures the probability that a trained model makes a mistake on a sample drawn from D. It's denoted by $P(x, y) \sim D(yf_w(x) \leq 0)$, where f_w is the model's prediction.Essentially, the generalization error gives us an idea of how well the model will perform in the real world, beyond the data it was trained on. It's a critical measure of the model's effectiveness and its ability to generalize its learned patterns to new, unseen data points.

For a fixed $x \in X$, we consider norm-bounded adversarial perturbations given by $B_2(x, R)$, for some fixed perturbation budget R. The robust loss on x is defined as

$$\mathcal{R}(w; x, y) = 1\left(\max_{x_0 \in B_2(x,R)} yf_w(x_0) \leq 0\right)$$

The expression represents the robust loss function for a given sample (x, y) and a parameter vector w.

In this expression:

- $\mathcal{R}(w; x, y)$ is the robust loss function for the sample (x, y) and parameter vector w.
- The notation $\max_{x_0 \in B_2(x,R)} yf_w(x_0) \leq 0$ denotes the maximum value of $yf_w(x_0)$ over all points x_0 in the L^2 ball centered at x with radius R.
- If the maximum value of $yf_w(x_0)$ over all x_0 in the ball $B_2(x, R)$ is less than or equal to 0, then $\mathcal{R}(w; x, y)$ is equal to 1. Otherwise, it is equal to 0.

In simpler terms, the robust loss function $\mathcal{R}(w; x, y)$ is a binary indicator function. It equals 1 if the maximum value of $yf_w(x_0)$ over all perturbed points x_0 within the L^2 ball centered at x with radius R is less than or equal to 0, indicating that the model's prediction $f_w(x_0)$ is on the correct side of the decision boundary for the given label y. Otherwise, it equals 0, indicating that there exists at least one perturbed point x_0 for which the model's prediction $f_w(x_0)$ is on the wrong side of the decision boundary.

The robust error then captures the probability that there exists an adversarial perturbation on samples drawn from D such that f_w makes a mistake on the perturbed point:

$$\mathcal{L}_R(w; x, y) := E(x, y) \sim D[L_R(w, x, y)].$$

The expression represents the robust error, denoted by $\mathcal{L}_R(w; x, y)$, for a given model parameter vector w, input sample x, and label y.

In this expression:

- $\mathcal{L}_R(w; x, y)$ is the robust error, which measures the expected value of the robust loss function $\mathcal{R}(w, x, y)$.
- $E(x, y)$ denotes the expectation operator over samples (x, y) drawn from the distribution D.
- D represents the joint distribution from which the samples (x, y) are drawn.
- $\mathcal{R}(w, x, y)$ is the robust loss function, which evaluates the model's performance on a given sample (x, y) and parameter vector w.

In summary, the robust error $\mathcal{L}_R(w; x, y)$ captures the expected value of the robust loss function over samples drawn from the distribution D. It provides insight into the overall performance of the model in handling adversarial perturbations.

In this work, they focus on two-layer neural networks with ReLU activation, parameterized by a pair of weight matrices (a, W), computing the following function:

$$f(x; a, W) := \frac{1}{\sqrt{m}} \sum_{s=1}^{m} a_s \sigma(w_s^\top x).$$

Here, m corresponds to the number of hidden nodes, i.e., the width of the network, $\sigma(x) = \max\{0, x\}$ is the ReLU activation function, and $W = [w_1, \ldots, w_m] \in R^{d\times m}$ and $a = [a_1, \ldots, a_m] \in R^m$ denote the top and the bottom layer weight matrices, respectively. The expression describes the operation of a two-layer neural network with ReLU (Rectified Linear Unit) activation function. Here's a breakdown of its components

Input: The input to the network is denoted by x. It could represent features, pixels of an image, or any other kind of input data.

Weight Matrices: The network has two sets of weights, represented by W and a.

- W is the weight matrix of the top layer, where each column $\mathbf{w}_s$ represents the weights connecting the input features to the s-th hidden neuron. It's a matrix of size $d \times m$, where d is the dimensionality of the input and m is the number of hidden nodes.
- a is the weight vector of the bottom layer. Each element a_s represents the weight associated with the s-th hidden neuron. It's a vector of size m.

ReLU Activation Function: The function $\sigma(x) = \max\{0, x\}$ is the ReLU activation function. It introduces non-linearity into the network. It replaces negative values with zeros, allowing the network to learn complex relationships in the data.

Neural Network Output: The function $f(x; a, W)$ computes the output of the neural network for a given input x. It linearly combines the outputs of the hidden neurons using the ReLU activation function and the weights a_s and $\mathbf{w}_s$.

The output is a weighted sum of ReLU-activated linear combinations of the input features.

In this work, They study the robustness of neural networks which are close to their initialization. In particular, They are interested in the lazy regime, defined as follows: Definition 2.1 (Lazy Regime). They initialize the top and bottom layer weights as $a_s \sim unif(\{-1, +1\})$ and $w_{s,0} \sim \mathcal{N}(0, I_d)$, $\forall s \in [m]$. Let $W_0 = [w_{s,1}, \ldots, w_{s,m}] \in R^{d\times m}$. The lazy regime is the set of all networks parametrized by (a, W), such that $W \in B_{2,\infty}(W_0, C_0/\sqrt{m})$, for some constant C_0. In this work, the researchers investigate the robustness of neural networks when they are initialized in a specific manner, known as the "lazy regime."

In the lazy regime, the initialization of the neural network's weights follows specific guidelines:

- The top layer weights a_s are drawn from a uniform distribution with values $\{-1, +1\}$.
- The bottom layer weights $w_{s,0}$ are drawn from a normal distribution with mean 0 and identity covariance matrix I_d.
- These initializations are applied for all s in the range from 1 to m, where m represents the number of hidden nodes.
- Let $W_0 = [w_{s,1}, \ldots, w_{s,m}] \in R^{d\times m}$ denote the weight matrix of the top layer.
- The lazy regime encompasses all networks parameterized by (a, W) such that W belongs to the L^2, ∞ ball centered at W_0, with a radius of $C_0/\sqrt{m}$, where C_0 is a constant.

In essence, the lazy regime explores the behavior of neural networks when initialized in a specific way, focusing on networks where the weights are close to their initialization values. The investigation aims to understand how these initialization schemes affect the robustness of the neural networks during training and inference

5 Results

In this section, They present Their main result. They show that under the lazy regime assumption, a single gradient step on f suffices to find an adversarial example to flip the prediction, given the network width is sufficiently wide but not excessively wide. Let $w_{s,0} \sim \mathcal{N}(0, I_d)$, $a \sim unif\{1, +1\}$, and $\gamma \in (0, 1)$. For any given x, with probability at least $1 - \gamma$, the following holds simultaneously for all $W \in B_{2,\infty}(W_0, C_0/\sqrt{m})$:

$$sign(f(x; a, W)) \neq sign(f(x + \delta; a, W))$$

where $\delta = \eta\nabla_x f(x; a, W)$, provided that the following conditions are satisfied:

Step Size: $|\eta| = \frac{C2}{\|\nabla f(x;a,W)\|_2^2}$, Width requirement: $n^{d2.4}, C3 \log\left(\frac{1}{\gamma}\right) \leq m \leq \min\left\{n, C4 \exp\left(d^{0.24}\right)\right\}, C5\gamma s \exp\left(d \log\left(\frac{1}{\gamma}\right)^2\right)$,

here they show that a single step of gradient suffices with a small step size of O(1/d). an attack size of O(1/ d) suffices, i.e., for any x a perturbation of size = O(1/ d) can flip the sign of f(x). O(1/ d) is the ªrightº perturbation budget in the sense that there is a sharp drop in robust accuracy. The paper focuses on the settings where in the weights of the trained neural network stay close to the initialization. The O(1/ m) deviation of the incoming weight vector at any neuron of the trained network from its initialization. This O(1/ m) deviation bound is also the largest radius that we can allow in our analysis. for large enough *d*, a step of gradient ascent can change the sign of *f*. the passage is highlighting a property of the linear model in a high-dimensional setting. The model's predictions are bounded with high probability due to concentration arguments, but the gradient of the model with respect to the input is relatively large in magnitude, making it easy for a step of gradient ascent to change the sign of the function. This can have implications for optimization in high-dimensional spaces.

where $C0, C1, C2, C3, C4, C5$ are constants independent of width m and dimension d. This statement describes conditions under which certain properties hold for a given neural network $f(x; a, W)$. Here's a breakdown: **Initialization:** The weights $w_{s,0}$ are sampled from a normal distribution with mean 0 and identity covariance matrix I_d. The parameter a is uniformly sampled from the set $\{1, +1\}$. The parameter γ lies in the open interval $(0, 1)$.

Probability Statement: For any given input x and with probability at least $1 - \gamma$, the following condition holds simultaneously for all weight matrices W belonging to a specific ball centered at W_0 with a radius of $C_0/\sqrt{m}$. This condition relates to the sign of the function $f(x; a, W)$ and $f(x + \delta; a, W)$.

Perturbation and Gradient: Here, δ is defined as the product of a scalar η and the gradient of the function f with respect to x. The gradient $\nabla_x f(x; a, W)$ provides information about the rate of change of f concerning changes in x.

Conditions: There are two conditions that need to be met:

(a) **Step Size Condition:** The magnitude of the scalar η is defined relative to the squared L^2 norm of the gradient of f with respect to x.

(b) **Width Requirement:** There are constraints on the width of the network, involving n (the number of inputs or features) and the dimensionality d of the input space. This condition imposes limitations on the network's width based on the input dimensions and other parameters.

Constants $C0, C1, C2, C3, C4, C5$ are introduced, which are independent of the network's width m and dimension d. Despite common practice involving multiple steps of projected gradient descent, the study reveals that even a single gradient step with a small step size can lead to adversarial examples. This emphasizes the vulnerability of over-parameterized neural networks trained under the lazy regime to adversarial attacks. The analysis suggests that a perturbation budget on the order of $O\left(\frac{1}{\sqrt{d}}\right)$ is sufficient to flip the sign of the network's output $f(x)$. This budget is relatively small, particularly for high-dimensional data, but remains effective in generating adversarial examples. The paper confirms these findings through empirical experiments. The results indicate that there exists a delicate balance between the width of the neural network and the dimensionality of the input data. The network needs to be wide enough to accommodate the complexity of the input features but not excessively wide to avoid computational inefficiencies. The analysis suggests that the network width should be polynomially large in the input dimension *d*. This requirement ensures that the network has sufficient capacity to capture complex patterns in the data. While the network needs to be sufficiently wide, it should not be excessively wide, meaning its growth should be sub-exponential with respect to the input dimension. Excessive width can lead to computational inefficiencies and overfitting. The upper bound on the network width allows for the over-parameterization necessary in settings like the Neural Tangent Kernel (NTK) regime. Over-parameterization refers to the use of more parameters than strictly necessary for the model to learn the data. The NTK regime is a theoretical framework used to analyze the behavior of neural networks during training. The explanation draws from previous works by Bubeck et al. and Bartlett et al. which provide theoretical insights into the behavior of neural networks, particularly in the context of random weights and linear models. These insights are extended to networks with weights close to random initialization.

6 Hit the Goal

Adversarial perturbation for any given input $\mathbf{x}$ such that $\delta < \mathbf{x}$ and the sign of the neural network output changes. Let's break down the key points: The main goal is to find a perturbation δ such that $\delta < x$ and $sign(f(x; a, W)) \neq sign(f(x + \delta; a, W))$. The expression is checking whether the sign of the neural network's output for the input x is not equal to the sign of the neural network's output for the perturbed input $x + \delta$. In other words, it's investigating whether the network's prediction changes its sign when the input is perturbed by δ. If the signs are different, it indicates a change in the prediction, implying vulnerability to adversarial examples.

We have the following:

$$f(x + \eta\nabla f(x; a, W); a, W) \leq \Big\{ f(x; a, W) - |\eta|\Omega(\sqrt{d})\|\nabla f(x; a, W)\| + |\eta| \sup_{\delta \in R^d : \|\delta\| \leq \eta \|\nabla f(x;a,W)\|} \|\nabla f(x + \delta; a, W) - \nabla f(x; a, W)\| \Big\}$$

Given any x and $W \in \mathcal{B}_{2,\infty}(W0, \sqrt{C0m})$, we need to control:

(a) The function value of neural network $f(x; a, W)$

(b) The gradient of neural network $\|\nabla f(x; a, W)\|$

(c) The difference of gradients $\|\nabla f(x; a, W) - \nabla f(x + \delta; a, W)\|$ for $\|\delta\| \leq O(1/\sqrt{d})$

Bounding the function value: For any x, with probability at least $1-\gamma$, $|f(x;a,W)| \leq 2\sqrt{\log(2/\gamma)+C0}$ holds for all $W \in \mathcal{B}_{2,\infty}(W0, \sqrt{C_0}/m)$, provided that $m \geq \Omega(\log(2/\gamma))$. This set of equations defines an inequality. It describes a bound on the output of the neural network $f(x+\eta\nabla f(x;a,W);a,W)$ in terms of the function value $f(x;a,W)$, the norm of the gradient $\|\nabla f(x;a,W)\|$, and the difference between gradients $\|\nabla f(x+\delta;a,W)-\nabla f(x;a,W)\|$, where δ is a perturbation. **Conditions for Control**: Given any x and W belonging to a certain set $B_{2,\infty}(W0, \sqrt{C0m})$, it is necessary to control three aspects:

(a) The function value of the neural network $f(x;a,W)$.

(b) The gradient of the neural network $\|\nabla f(x;a,W)\|$.

(c) The difference of gradients $\|\nabla f(x;a,W)-\nabla f(x+\delta;a,W)\|$ for $\|\delta\| \leq O(1/\sqrt{d})$.

Lemma Introduction: Lemma provides a bound on the function value of the neural network. It states that with a certain probability, the absolute value of the function $f(x;a,W)$ is bounded by a certain expression, given certain conditions on W and m.

The goal they reached, is now to control each of the three terms as shown in the display above. Note that, at a high-level, the above bounds suffice to flip the label, thereby establishing the main result.