Adversarial Robustness

Ekaterina Mozhegova

7th April

The study is based on the tutorial "Adversarial Robustness - Theory and Practice".

Abstract

Adversarial attacks are small yet malicious perturbations applied to input datasets. Often invisible to the human eye, they can damage the internal structure of the model and consequently its accuracy and reliability. These attacks pose a threat to security-critical tasks, including computer vision and natural language processing, among others. Therefore, the objective of the new field of study is to ensure that models remain resilient to such perturbations. Zico Kolter and Aleksander Madry have observed several types of adversarial attacks and proposed methods for building robust models. They are all covered in this study.

Contents

1 Chapter 1 - Introduction to adversarial robustness 3
1.1 Targeted attacks . . . 3
1.2 Adversarial risks . . . 3

2 Chapter 2 - Linear models 4
2.1 Linear models . . . 4
2.2 Practical results . . . 5

3 Chapter 3 - Neural Networks 5
3.1 Norms . . . 5
3.2 Aspects of Neural Networks . . . 5
3.3 Lower Bounding the inner optimization . . . 6
3.3.1 The Fast Gradient Sign Method (FGSM) . . . 6
3.3.2 Projected Gradient Descent . . . 7
3.3.3 Targetted attacks . . . 7
3.3.4 Non-l_∞ norms . . . 7
3.4 Exactly solving the inner maximization (combinatorial optimization) . . . 8
3.4.1 Certifying robustness . . . 8

3.4.2 Upper and Lower bounds 8
3.5 Upper bounding the inner maximization 8
3.5.1 Convex relaxation 8

1 Chapter 1 - Introduction to adversarial robustness

Let's define the model for a classification problem with the hypothesis function h_Θ. The loss function (softmax loss) of the model is the following: $l(h_\theta(x_i), y_i)$.

The common approach for the classification problem is to solve the optimization problem:

$$\min_\theta \frac{1}{m} \sum_{i=1}^{m} l(h_\theta(x_i), y_i)$$

It is typically solved by calculating the gradient of our loss function:

$$\Theta := \Theta - \frac{\alpha}{\beta} \sum_{i \in \beta} \nabla_\Theta l(h_\theta(x_i), y_i)$$

In standard optimization tasks, we aim to minimize the loss function. However, for creating adversarial attacks, our objective is to maximize it:

$$\max_{\hat{x}} l(h_\Theta(\hat{x}, y))$$

$\hat{x}$ here describes the adversarial example.

What we are specifically interested in is the gradient:

$$\nabla_\Theta l(h_\theta(x_i), y_i)$$

Adversarial example $\hat{x}$ needs to be similar to the original input x to stay meaningful, so we optimize over the pertrubation to x.

$$\max_{\delta \in \Delta} l(h_\Theta(x + \delta), y)$$

For intuition, Δ should be the range in which the input is the same as the original x.

1.1 Targeted attacks

Using this technique we can deceive the model into predicting an incorrect class. To achieve this, we aim to maximize the loss function for the correct label while minimizing the loss function for the targeted class.

$$\text{maximize} \left(l(h_\theta(x + \delta), y) - l(h_\Theta(x + \delta), y_{\text{target}}) \right)$$

1.2 Adversarial risks

The concept of adversarial risk is introduced to enhance the robustness of the model. We can evaluate adversarial risks alongside traditional empirical risks. They are estimated using a finite set of data.

$$R_{adv}(h\theta) = E_{(x,y) \sim D} \left[\max_{\delta \in \Delta(x)} \mathcal{L}(h\theta(x + \delta), y) \right]$$

Evaluating adversarial risks allows us to assess the accuracy of the model even in cases where it is subjected to attacks or adversarial manipulation. This evaluation can help us predict the behaviour of our model in case of attacks.

$$\Theta := \Theta - \frac{\alpha}{|\beta|} \sum_{(x,y)\in\beta} \nabla_\Theta \max_{\delta\in\Delta(x)} l(h_\theta(x_i+\delta), y_i)$$

The gradient of the inner term, which involves a maximization problem, is computed as follows, taking into account Danskin's theorem.

$$\delta^* = \text{argmax}_{\delta\in\Delta(x)} \mathcal{L}(h_\Theta(x+\delta^*), y)$$

$$\Delta_\Theta \max_{\delta\in\Delta(x)} l(h_\Theta(x+\delta), y) = \Delta_\Theta l(h_\Theta(x+\delta^*), y)$$

2 Chapter 2 - Linear models

2.1 Linear models

Let's define the model for a binary classification problem with the hypothesis function h_Θ. The loss function (logistic loss) of the model is the following:

The hypothesis function is:

$$h_\Theta(x) = w^T x + b$$

And the loss function is:

$$l(h_\Theta(x), y) = \log(1 + \exp(-y \cdot h_\Theta(x))) \equiv L(y \cdot h_\Theta(x)),$$

where $L(z) = \log(1 + \exp(-z))$, a decreasing function.

$$\max_{\|\delta\|\leq\epsilon} l(w^T(x+\delta), y) \equiv \max_{\|\delta\|\leq\epsilon} L(y \cdot (w^T(x+\delta)) + b)$$

Due to L being a decreasing function, its argument must be minimized in order for the function to reach its maximum.

$$\max_{\|\delta\|\leq\epsilon} L(y\cdot(w^T(x+\delta))+b) = L \min_{\|\delta\|\leq\epsilon} (y\cdot(w^T(x+\delta))+b) \equiv y\cdot(w^T(x+b)) + \min_{\|\delta\|\leq\epsilon} (y\cdot w^T\cdot\delta)$$

For solving this optimization problem:

$$\min_{\|\delta\|\leq\epsilon} (y \cdot w^T \cdot \delta),$$

let's first assume that $y = 1$.

Since $|\delta|_\infty \leq \epsilon$, where $\epsilon > 0$, the obvious approach to solve the given minimization problem is to assign $\delta = -\epsilon$ for $w^T \geq 0$ and $\delta = \epsilon$ for $w^T \leq 0$.

$$\delta^* = -y\epsilon \cdot \text{sign}(w)$$

Let's denote the optimized δ^* as

$$\delta^* = -\epsilon||w||_1$$

and substitute it in the initial equation, so that:

$$\max_{\|\delta\|_\infty \leq \epsilon} L(y \cdot (w^T(x+\delta)+b)) = L(y \cdot (w^T x + b) - \epsilon\|w\|_1)$$

2.2 Practical results

I observed a tradeoff while testing the robustness of the model on MNIST dataset for images labeled as 0 and 1. Surprisingly, the model made more mistakes on non-adversarial data, despite previously performing better on this type of data.

3 Chapter 3 - Neural Networks

3.1 Norms

1. l_∞
2. l_2
3. l_1

3.2 Aspects of Neural Networks

The application and performance of adversarial attacks are highly relevant to neural networks.

The form of the optimization problem (inner maximization problem) remains the same:

$$\max_{\|\delta\| \leq \epsilon} \ell(h_\theta(x+\delta), y)$$

What is different from the previous examples is $h(\theta)$, which now represents a neural network.

The complexity of neural networks' architecture makes them more challenging to be robust, while simultaneously rendering them susceptible to adversarial attacks.

First of all, loss surfaces do not often guide to the optimal solution, while they can be too steep, which often corresponds to local optima convergence.

Secondly, the inner maximization problem for neural networks is also more challenging due to the non-convexity of the cost surface.

Moreover, it is more difficult to navigate the cost surface.

To further explore the topic of adversarial attacks in neural networks, let's propose that adversarial attacks are based on two main components:

1. The norm of the perturbation ball.

2. The optimization method used within that norm ball.

Three main approaches to adversarial attacks on neural networks exist:

1. Lower Bounding the inner optimization,
2. Exactly solving the inner maximization (combinatorial optimization),
3. Upper Bound optimization

3.3 Lower Bounding the inner optimization

The Lower Bound method relying on empirical solutions is considered the most common approach.

3.3.1 The Fast Gradient Sign Method (FGSM)

The idea of adversarial attacks via the FGSM relies on calculating an optimal perturbation δ that maximizes the loss function. This calculation is based on the gradient obtained through backpropagation. This gradient indicates how the loss function changes when we make small adjustments to δ.

Based on the gradient, we adjust δ to maximize the loss function.

$$g = \nabla_\delta l(h_\theta(x + \delta), y)$$

This is performed by adding a fraction of the gradient to δ, scaled by a small step size parameter α.

$$\delta = \delta + \alpha \cdot g$$

We adjust δ based on the sign of the gradient: when $g < 0$, $\delta < 0$ and when $g > 0$, $\delta > 0$. Thus, we have

$$\delta := \epsilon \cdot \text{sign}(g)$$

.

However, we also need to stick to the constraints. So, we might need to project δ back into a feasible region after adjusting it.

The l_∞ norm ball, which selects the maximum absolute value among the elements in the set, is well-suited for this task.

We must ensure δ remains within the norm ball: $||\delta||_\infty \leq \epsilon$. Therefore, we project δ back into the norm ball after each adjustment.

So, then we project δ again into the norm ball.

FGSM is specifically designed to attack under l_∞ norm.

3.3.2 Projected Gradient Descent

An iterative method that ensures to adjust the perturbation δ in the direction that maximizes the loss function while ensuring it stays within the specified constraints.

Both FGSM and PGD generate adversarial examples that are optimal since they maximize the loss function. While FGSM is a straightforward, one-step process, which takes a single step in the direction of the goal, PGD iteratively updates the perturbation to generate adversarial examples.

Thus, we can assume PGD as an iterative FGSM, which iterates over multiple steps.

$$\delta := \mathrm{P}(\delta + \alpha \nabla_\delta \ell(h_\theta(x+\delta), y))$$

P is the projection of the value onto the ball norm.

Although PGD can find the optima more effectively than FGSM is is proved to be slower. PGD has challanges due to the fact that it starts iterating from $\delta = 0$, where the gradient is small. However, it has the risk of overshooting the optimal perturbation when the gradients grow fast outside this neighborhood.

3.3.3 Targetted attacks

The aim of untargeted attacks is to change the predicted label to any alternative label by maximizing the loss for the actual (true) label and minimizing the loss for any alternative label.

The aim of the targeted attacks is to change the predicted label to a particular alternative label by maximizing the loss for the actual (true) label and minimizing the loss for the targeted label.

3.3.4 Non-l_∞ norms

Note: projecting onto the norm ball means clipping values of δ to lie within $[-\epsilon, \epsilon]$

In the methods described above, the optimization was performed under the l_∞ norm; however, the other norm types can also be used for the same task.

For example, we can transfer the task to the l_2 norm by creating a constraint that $x + \delta$ lies in the range [0, 1].

While working with l_2 norm, we simply project normalized steepest descent for the l_2 ball.

$$\delta := P_\epsilon \left(\delta - \alpha \frac{\nabla_\delta \ell(h_\theta(x+\delta), y)}{\|\nabla_\delta \ell(h_\theta(x+\delta), y)\|_2} \right)$$

The key behind the l_∞ and l_2 norms is the fact that while l_∞ attacks are distributed smoothly in the image, l_2 attacks are typically localized in the specific area of the image.

3.4 Exactly solving the inner maximization (combinatorial optimization)

We assume attacks against each and every class and determine whether an adversarial example exists in the neighbourhood of some point.

In this method, we do not actually solve the inner maximization problem, but we can determine exactly whether or not an adversarial example exists within a certain radius.

3.4.1 Certifying robustness

3.4.2 Upper and Lower bounds

Due to the high computation complexity of the excat solution method, this approach is not a good choice since it can be barely scaled to larger networks.

3.5 Upper bounding the inner maximization

The other method is Upper Bounding, which incorporates different approaches to forming an upper bound. We will discuss two apppraches: convex relaxation of the integer programming and the other is based on bound propagation.

3.5.1 Convex relaxation

The challenge that arose in the previous method is the binary constraint $v_i \in 0, 1$, which captures the ReLU operation. This problem is hard to solve since the set is non-convex. So, we can ease the problem by relaxing the constraints and dealing with a convex set instead.

Instead of requiring v_i to be 0 or 1, we allow it to take fractional values, i.e.:

$$0 \leq v_i \leq 1$$

Despite this change, the overall optimization problem remains unchanged from before.

The key idea is that this relaxation helps us better understand the problem.

For instance, if we solve the relaxed version of a targeted attack problem and find that the objective (the value we're trying to optimize) is still positive, it tells us something important. Specifically, it suggests that the attack won't be effective.

If no "adversarial example" exists in the relaxed set, it won't exist in the original set either. So, even though we've simplified the problem by relaxing the constraints, we can still draw meaningful conclusions from it.