Project for the exam - Optimization for Data Science

Frank-Wolfe for White Box Adversarial Attacks

Eleonora Brasola
Student n° 2007717

Alberto Cocco
Student n° 2020357

Greta Farnea
Student n° 2019052

1. Introduction

Machine learning models, like neural networks, are found to be vulnerable to *adversarial examples*, that are obtained by modifying examples drawn from the data distribution. Models tend to misclassify such examples that are slightly different from the correctly classified examples. The surprising thing is that this behaviour is proper of a wide variety of machine learning models: adversarial examples reveal some blind spots in the state-of-the-art training algorithms.

Many hypothesis have been made to explain such behaviour, a lot of which are related with complexity and depth of the neural networks. In Goodfellow [4], it is shown that it is sufficient to have a simple linear model in order to fool it with adversarial examples. It is interesting to notice that an adversarial example generated for one model is often misclassified by other models, moreover these models generally agree on the output by missclassifying a single sample into the same class. This can be explained as a result of 'adversarial directions' being highly related to the weights vectors of the model and the fact that in order to provide an adversarial example what matter the most are these directions and not the specific points in space. Moreover different models learn similar function when trained to perform the same task. We refer to this property as *transferability* of the adversarial example.

In this project, our goal is to implement four different algorithms for Adversarial Attacks. According to Rinaldi [5], Goodfellow [4], Dong [3] and Chen [2], we use the *Projected Gradient Descent*, the *Fast Gradient Sign*, the *Momentum Iterative Fast Gradient Sign* and the *Frank-Wolfe-white* methods.

All of these algorithms belong to the constrained optimization theory, thus their aim is to minimize (or maximize) a function and to comply some restrictions on the domain of the function.

1.1. Constrained optimization

The most general definition of a constrained problem is:

$$\min f(x) \qquad (1)$$
$$\text{subject to } x \in C$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function and $C \subseteq \mathbb{R}$ is a convex set.

We can define the *set of feasible directions* $F(\bar{x})$ of a point $\bar{x} \in C \neq \emptyset$ as:

$$F(\bar{x}) = \{d \in \mathbb{R}^n, d \neq 0 : \exists\, \delta > 0 \text{ s.t. } \bar{x} + \alpha\, d \in C, \quad \forall\, \alpha \in (0, \delta)\}. \qquad (2)$$

We recall this proposition:

Proposition 1.1.1. *Let $x^* \in C$ be local minimum for problem 1 with $C \subseteq \mathbb{R}^n$ convex and $f \in C^1(\mathbb{R}^n)$. Then*

$$\nabla f(x^*)^T (x - x^*) \geq 0 \quad \forall x \in C\,. \qquad (3)$$

We can extend this proposition with another one that gives a necessary and sufficient condition for the global minimum:

Proposition 1.1.2. *Let $C \subseteq \mathbb{R}^n$ be a convex set and $f \in C(\mathbb{R}^n)$ be a convex function. A point $x^* \in C$ is a global minimum of problem 1 if and only if*

$$\nabla f(x^*)^T (x - x^*) \geq 0 \quad \forall x \in C\,.$$

All these properties are valid for a minimization constrained problem such as 1.

They can be extended also for a maximization constrained problem, defined as:

$$\max f(x)$$
$$\text{subject to } x \in C\,.$$

It is an easy derivation since $\max f(x) \equiv \min -f(x)$.

One final key ingredient for constrained optimization is:

Weierstrass Theorem A continuous function in $\mathbb{R}$ over a convex set always admits both global maximum and minimum.

1.2. Adversarial attacks

Adversarial examples are used to evaluate the robustness of machine/deep learning models before they are effectively run. They are generated by little translations along the gradient direction and, in this way, they add small noise to the examples that are 'invisible' to the human eye. However, giving adversarial examples as input to machine learning models fool them making their output wrong.

One important property of the adversarial examples is their transferability. In fact, an adversarial example created for a single model usually is adversarial also for others. This reveals that different machine learning models learn the same features and capture the same important characteristics of their training data distribution.

There exists a variety of algorithms to create adversarial attacks and in this work we only see few that are based on gradient directions. The main idea is that, given an image as input, we modify each pixel moving along the gradient direction. Furthermore, we need to satisfy some restrictions in order not to go too far from the original input and to obtain an image that, to the human eye, is not different from the original one.

In order to define the constrained problem for adversarial attacks, we recall that in this paper we consider classifier models.

A classifier learns a function $f(x) : x \in \mathcal{X} \to y \in \mathcal{Y}$, where x is the input image and y is the label of the prediction and $\mathcal{X}$ and $\mathcal{Y}$ are their domain sets.

As an error measure, we define a loss function $\ell(y, y_{\text{true}}) : (y, y_{\text{true}}) \in \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, that usually, for multi-classification purpose, is known as *Categorical Cross Entropy*. The variable y is the predicted label and the variable y_{true} is the ground-truth target label of the classifier's input x.

The main purpose of the adversarial attack algorithms is to make the classifier's output as wrong as possible. Thus, the problem we have to optimize is defined as:

$$\begin{aligned} &\max \ell(x) \\ &\text{subject to } \|x - x_{\text{ori}}\| \leq \varepsilon \end{aligned} \tag{4}$$

where $\ell(x)$ is the loss function of the classifier, x_{ori} is the input image, x is the adversarial image and the condition is given by a norm $\|\cdot\|$ and a tolerance value ε. We keep in mind that this problem can be transformed into a minimization problem by taking the opposite of $\ell(x)$ as objective function.

The notation $\ell(x)$ can be confusing, as we are supposed to have the labels y and y_{true} as input. For simplicity, we write $\ell(x)$ instead of $\ell(y(x), y_{\text{true}}(x))$, referring to the classifier's loss function for the input x.

The constraint of the problem 4 ensures that the modifications we are applying are small enough, according to the value of ε. The most used norms are the $\|\cdot\|_1$, $\|\cdot\|_2$ and $\|\cdot\|_\infty$. In this work, we consider only the last one.

1.3. Untargeted and targeted attacks

As said before, the adversarial attacks' goal is to make the classifier's output wrong. We can decide whether we want to have control of the wrong output or we just care that the classifier mistakes. For these purposes, there are two types of adversarial methods: *untargeted* ones and *targeted* ones.

For the untargeted problems, we just refer to 4. It does not matter which is the output label of the adversarial example, as long as the classifier is wrong and the confidence of the output is high enough. We are satisfied only to maximize its loss function, according to the restriction on the adversarial image.

For the targeted algorithms, the problem is a little different. It is not sufficient to have a high loss value, we want to make the classifier predict a specific target y_{target}. So we want to maximize $\ell(y, y_{\text{true}})$ but also to minimize $\ell(y, y_{\text{target}})$. It has now become a min-max problem and we combine 4 with the following minimization problem:

$$\begin{aligned} &\min \ell(y, y_{\text{target}}) \\ &\text{subject to } \|x - x_{\text{ori}}\| \leq \varepsilon \,, \end{aligned}$$

in order to define a new constrained optimization problem for targeted adversarial attacks:

$$\begin{aligned} &\max \ell(y(x), y_{\text{true}}(x)) - \ell(y(x), y_{\text{target}}(x)) \\ &\text{subject to } \|x - x_{\text{ori}}\| \leq \varepsilon \,. \end{aligned} \tag{5}$$

2. Theoretical background

Adversarial studies are relatively recent, since the first study has been published in 2013. In general the definition of an attack depends on how much information about the model an adversary can access to and they can be divided into two categories: *white-box attacks* and *black-box attacks*.

In this project we focus on the first type of attacks where the adversary have full access to the target model, and thus can compute efficiently the gradient. The optimization-based methods proposed for this setting are several and, as stated before, we will analyze and test 4 different methods: *FGSM, PGD, MI-FGSM, FW-White*.

The *FGSM* method works by linearizing the network loss function and ends in just one step. *PGD* and *MI-FGSM* are iterative methods that achieve better results than simple *FGSM* but they both generate adversarial examples near the boundary of the perturbation set. The more recent *FW-white* achieve both high success rates and good adversarial examples.

2.1. FGSM

Differently from the other three algorithms, the *Fast Gradient Sign* method is a one-step gradient-based approach. It is not iterative and updates only once the input value x_{ori}.

This method linearizes the cost function around the input value. It must to be applied a perturbation η to the original point x_{ori} and, in Goodfellow [4], they suggest to use:

$$\eta = \varepsilon \, \text{sign}(\nabla_x \ell(x_{\text{ori}}))\,.$$

Since in this work we perform both untargeted and targeted attacks, we define the update rule for each of these problem.

$$\begin{aligned} x &= x_{\text{ori}} + \varepsilon \, \text{sign}(\nabla_x \ell(x_{\text{ori}})) \quad \text{(untargeted)}\,; \\ x &= x_{\text{ori}} - \varepsilon \, \text{sign}(\nabla_x \ell(x_{\text{ori}})) \quad \text{(targeted)}\,. \end{aligned}$$

It can be noticed that what changes is only the sign before the perturbation. This is due to the fact that:

1. the function $\ell(x_{\text{ori}})$ stays the same for the two problems;

2. the untargeted attack is a maximization problem, thus we have to move along the direction of the gradient in order to find the maximum. For this reason there is the $+$ sign before the perturbation;

3. the targeted attack is a minimization problem, thus we move in the opposite direction of the gradient in order to find the minimum. For this reason there is the $-$ sign before the perturbation.

The constraint of our problem, $\|x - x_{\text{ori}}\|_\infty < \varepsilon$ is certainly satisfied and it is easy to prove. Each element of the tensor corresponding to the input image x_{ori} is increased or decreased by exactly ε. This is due to the fact that the expression $\text{sign}(\nabla_x \ell(x_{\text{ori}}))$ can take values only in $\{-1, 1\}$.

We can deduce that the adversarial images will be on the boundary of the convex set, since we always apply the largest possible perturbation.

2.2. PGD

The *Projected Gradient* method is iterative and gradient-based approach. It is based on the fact that, considering only the direction of the gradient, without imposing any constraint, we might find a new value for x_{k+1} that is outside the convex set C:

$$x_{k+1} = x_k - \gamma_k \, \nabla f(x_k) \text{ might be that: } x_{k+1} \notin C\ .$$

In order to overcome this issue, the *PGD* method projects back the new point to the nearest point belonging to the set C. The procedure of the algorithm is shown in the table **Algorithm 1**.

Algorithm 1 Projected gradient general

```
1: for $k = 1 \ldots$ do
2:     Set $\bar{x}_k = \varrho_C(x_k + s_k \nabla f(x_k))$    ▷ if untargeted attack, with $s_k > 0$
3:     Set $\bar{x}_k = \varrho_C(x_k - s_k \nabla f(x_k))$    ▷ if targeted attack, with $s_k > 0$
4:     If $\bar{x}_k$ satisfies some specific condition, then STOP
5:     Set $x_{k+1} = x_k + \gamma_k(\bar{x}_k - x_k)$    ▷ with $\gamma_k \in (0, 1]$
6: end for
```

For consistency in this work, we consider the projection function $\varrho_C(\cdot)$ based on the infinite norm $\| \cdot \|_\infty$. In line 2 and 3 of the algorithm 1, the projection $\bar{x}_k$ over C is the solution of the following minimization problem:

$$\min_{x \in C} \|x - (x_k \pm s_k \nabla f(x_k))\|_\infty\,.$$

We define the argument of the projection $\varrho_C(\cdot)$ according to the type of the attack we are considering and, as done for the *FGSM*, we move along or in the opposite direction of the gradient, adding or subtracting the perturbation to x_k.

The projection function $\varrho_C(\cdot)$ ensures that the final output of the algorithm satisfies the constraint. At the end of each iteration, the new point x_{k+1} is certainly in the convex set. The line 5 can be re-written as:

$$x_{k+1} = x_k + \gamma_k(\bar{x}_k - x_k) = \gamma_k \bar{x}_k + (1 - \gamma_k) x_k\,.$$

Since γ_k is contained in $(0, 1]$ and both x_k and $\bar{x}_k$ are contained in the convex set C, the above combination is convex and this ensures that $x_{k+1} \in C$.

2.3. MI-FGSM

Here we present the *Momentum Iterative Fast Sign* method that is a iterative version of the simple *FGSM* combined with momentum. The momentum method is, in general, an acceleration technique for gradient based algorithms. It consists in memorizing the past gradient direction in order to avoid narrow valleys, poor local minima or maxima and other issues. Since the *FGSM* ends in one step under the assumption of linearity of the decision boundary [4], it may easily be stacked into poor local areas, thus it is a good idea to combine it with the momentum method.

One additional advantage of momentum in this setting is that it gives a good trade-off between attack ability and the transferability of the adversarial example.

Algorithm 2 MI-FGSM

```
Input: A real example $x$, ground-truth label $y$
Output: An adversarial example $x^*$
Require: Size of perturbation $\varepsilon$, number of iterations $T$,
    step size $\gamma$, decay factor $\beta$
1: Fix $g_0 = 0$ and $x_0^*$
2: for $t = 0$ to $T - 1$ do
3:     Input $x_t$ and obtain the gradient $\nabla_x J(x_t, y)$
4:     Accumulate velocity
           $g_{t+1} = \beta \cdot g_t + \frac{\nabla_x J(x_t, y)}{\|\nabla_x J(x_t, y)\|_1}$
5:     Update
           $x_{t+1} = x_t + \gamma \cdot \text{sign}(g_{t+1})$   if untargeted attack
           $x_{t+1} = x_t - \gamma \cdot \text{sign}(g_{t+1})$   if targeted attack
6: end for
```

The algorithm scheme is summarized in the **Algorithm 2**.

In line 1 we initialize the gradient to zero and fix the starting point. The momentum term can be found in line 4 with a decay factor β and the current gradient is normalized with its L_1 norm in order to correct the gradients scale. As done in the previous methods, in line 5 we choose the update rule according to the type of attack we are considering. An adversarial example x_t^* is perturbed in the direction of g_t (or its opposite) with a step size of γ.

Observation 2.1. If $\beta = 0$ the *MI-FSGM* reduces to *FSGM*.

2.4. FW-white

The *Frank Wolf* method is an iterative first-order optimization algorithm. It is widely used in data science since it is a projection-free algorithm and has a smaller cost per iteration with respect to projection methods.

In this section, we present the Frank Wolf method for *white-box* attacks proposed in Chen [2]. The general idea of this method is to call a *Linear Minimization Oracle* (LMO). Starting from a feasible solution, at each iteration we define a descent direction as the solution of:

$$\min_{x \in C} \nabla f(x_k)^T (x - x_k)\,.$$

The previous is equivalent to the linear approximation of $f(\cdot)$ in x_k:

$$\min_{x \in C} f(x_k) + \nabla f(x_k)^T (x - x_k)\,.$$

By the Weierstrass theorem, we have that a solution of such problem exists. In the context of *white-box attacks* the algorithms introduce an additional momentum $\mathbf{m}_t$ term. The scheme is summarized in the **Algorithm 3**.

Algorithm 3 FW-White

```
Input: Number of iterations T, step size $\gamma$
Output: An adversarial example $x_T$
1: Set $x_0 = x_{\text{ori}}$, $m_{-1} = \nabla_x f(x_0)$
2: for $t = 0$ to $T - 1$ do
3:     $m_t = \beta \cdot m_{t-1} - (1 - \beta) \cdot \nabla f(x_t)$   ▷ if untargeted
4:     $m_t = \beta \cdot m_{t-1} + (1 - \beta) \cdot \nabla f(x_t)$   ▷ if targeted
5:     $v_t = \text{argmin}_{x \in C} \langle x, m_t \rangle = -\varepsilon \cdot \text{sign}(m_t) + x_{\text{ori}}$
6:     $d_t = v_t - x_t$
7:     $x_{t+1} = x_t + \gamma d_t$
8: end for
```

The difference between untargeted and targeted attacks in this case is a little different from the previous methods we have presented. This is because we want not to modify the closed-form solution of the LMO, defined in line 5. For this reason, instead of using two different update rules, we define the momentum term m_t in different ways.

In line 4, we propose what is done in Chen [2]; in line 3, since we have to move along the opposite direction of the movement for targeted attacks, we consider $-\nabla f(x_t)$ as the gradient of the cost function.

Thanks to the momentum term m_t, the LMO direction is stabilized and the convergence of the algorithm is found to be empirically accelerated.

In line 5, we exploit the closed-form solution for the LMO for the L_∞ norm case. The proof of this formulation is in the following section 2.4.1 and here we just write the full update rule of line 7:

$$x_{t+1} = x_t - \gamma\, \varepsilon \cdot \text{sign}(m_t) - \gamma(x_t - x_{\text{ori}})\,.$$

The last term of the equation above enforces x_t to be close to x_{ori} for all of the iterations. We can suppose that the final adversarial image will have smaller distortion with respect to the other algorithms and this is the main advantage of the *FW* method with momentum.

2.4.1 LMO closed-form solution

Proposition 2.4.1. *The LMO defined in line 4 has a closed-form solution both for L_2 and L_∞.*

Proof. Let $\mathbf{h} = (x - x_{ori})/\varepsilon$. Then considering line 4 we have:

$$\begin{aligned} \underset{\|x - x_{ori}\|_p \le \varepsilon}{\text{argmin}} \;(x, m_t) &= \underset{\|h\|_p \le 1}{\text{argmin}}\, \varepsilon \cdot \langle h, m_t \rangle \\ &= \underset{\|h\|_p \le 1}{\text{argmax}}\, \varepsilon \cdot \langle h, -m_t \rangle \end{aligned}$$

By Hölder's inequality (element wise), the maximum value is reached when:

$$|h_i| = c \cdot |(m_t)_i|^{\frac{1}{p-1}}$$

Since $\|h\|_p \leq 1$ we have:

$$h_i = -\frac{\text{sign}((m_t)_i) \cdot |(m_t)_i|^{\frac{1}{p-1}}}{(\sum_{i=1}^{d} |(m_t)_i|^{\frac{1}{p-1}})^{\frac{1}{p}}}$$

And,

$$x_i = \varepsilon \cdot -\frac{\text{sign}((m_t)_i) \cdot |(m_t)_i|^{\frac{1}{p-1}}}{(\sum_{i=1}^{d} |(m_t)_i|^{\frac{1}{p-1}})^{\frac{1}{p}}} + (x_{ori})_i$$

For $p = 2$ we have

$$v_t = -\frac{\varepsilon \cdot m_t}{\|m_t\|_2} + x_{ori}$$

For $p = \infty$ we have

$$v_t = -\varepsilon \cdot \text{sign}(m_t) + x_{ori}$$

□

Observation 2.2. For $T = 1$ Algorithm 3 reduces to the *FGSM*.

2.4.2 Convergence Analysis for FW-White

We are in a constrained optimization setting and in general the loss function for common DNN models are non-convex, thus the gradient norm of f is no longer a proper convergence criterion. We then adapt as a criterion the Frank-Wolfe gap:

$$g(x_t) = \max_{x \in C} \langle x - x_t, -\nabla f(x_t) \rangle \tag{6}$$

There are 2 essential assumption to make in order to provide the convergence guarantee of the algorithm.

Assumption 2.4.1. Function $f(\cdot)$ is L-smooth with respect to x, i.e. for any x, x' it holds that:

$$f(x') \leq f(x) + \nabla f(x)^T (x' - x) + \frac{L}{2}\|x' - x\|_2^2$$

Assumption 2.4.2. Set C is bounded with diameter D, i.e. $\|x' - x\|_2 \leq D$ for all $x, x' \in C$

Lemma 2.4.1. *Under assumptions 2.4.2 and 2.4.1, for m_t in Algorithm 3, it holds that*

$$\|\nabla f(x_t) - m_t\|_2 \leq \frac{\gamma L D}{1 - \beta}$$

Proof. By definition of m_t in line 3 we have that the fist term is:

$$\begin{aligned}
&\|\nabla f(x_t) - m_t\|_2 = \\
&\quad = \|\nabla f(x_t) - \beta m_{t-1} - (1 - \beta)\nabla f(x_t)\|_2 \\
&\quad = \beta \cdot \|\nabla f(x_t) - m_{t-1}\|_2 \\
&\quad = \beta \cdot \|\nabla f(x_t) - \nabla f(x_{t_1}) + \nabla f(x_{t-1}) - m_{t-1}\|_2 \\
&\quad \leq \beta \cdot \|\nabla f(x_t) - \nabla f(x_{t_1})\|_2 + \beta \cdot \|\nabla f(x_{t_1}) - m_{t-1}\|_2 \\
&\quad \leq \beta L\|x_t - x_{t-1}\|_2 + \beta \cdot \|\nabla f(x_{t_1} - m_{t-1}\|_2
\end{aligned}$$

This holds exploiting the triangle inequality and assumption 2.4.1. Looking at line 6 and using assumption 2.4.2 we also have:

$$\begin{aligned}
\|x_t - x_{t-1}\|_2 &= \gamma\|d_{t-1}\|_2 \\
&= \gamma\|v_{t-1} - x_{t-1}\|_2 \leq \gamma D
\end{aligned}$$

Combining both these estimates yields:

$$\begin{aligned}
&\|\nabla f(x_t) - m_t\|_2 \leq \\
&\quad \leq \gamma(\beta L D + \beta^2 L D + \cdots + \beta^t \|\nabla f(x_0) - m_0\|_2) \\
&\quad = \gamma(\beta L D + \beta^2 L D + \cdots + \beta^{t-1} L D) \\
&\quad \leq \frac{\gamma L D}{1 - \beta}
\end{aligned}$$

□

Theorem 2.4.1. *Under assumptions 2.4.2 and 2.4.1, let $\gamma_t = \gamma = \sqrt{2(f(x_0) - f(x^*))/(C_\beta L D^2 T)}$, the output of Algorithm 3 satisfies*

$$\tilde{g}_T \leq \sqrt{\frac{2C_\beta L D^2(f(x_0) - f(x^*))}{T}}$$

Therefore the Frank Wolf white-box attack algorithm achieves a $O(1/\sqrt{T})$ convergence rate.

Proof. By assumption 2.4.1, we have:

$$\begin{aligned}
f(x_{t+1}) &\leq f(x_t)\nabla f(x_t)^T (x_{t+1} - x_t) + \frac{L}{2}\|x_{t+1} - x_t\|_2^2 \\
&= f(x_t)\gamma\nabla f(x_t)^T (v_t - x_t) + \frac{L\gamma^2}{2}\|v_t - x_t\|_2^2 \\
&\leq f(x_t)\gamma\nabla f(x_t)^T (v_t - x_t) + \frac{L D^2 \gamma^2}{2} \\
&= f(x_t) + \gamma m_t^T (v_t - x_t) + \\
&+ \gamma(\nabla f(x_t) - m_t)^T (v_t - x_t) + \frac{L D^2 \gamma^2}{2}
\end{aligned}$$

Now we use the auxiliary quantity:

$$\tilde{v}_t = \underset{x \in C}{\text{argmin}} \langle v, \nabla f(x_t) \rangle$$

The Frank Wolf gap (6) implies:

$$g(x_t) = -\langle \tilde{v}_t - x_t, \nabla f(x_t) \rangle$$

And on the other hand in line 5 we find:

$$v_t = \underset{v \in C}{\operatorname{argmin}} \langle v, m_t \rangle \implies \langle v_t, m_t \rangle \leq \langle \tilde{v}_t, m_t \rangle$$

Combining both the previous equations:

$$\begin{aligned} f(x_{t+1}) &\leq f(x_t) + \gamma m_t^T(\tilde{v}_t - x_t) + \\ &+ \gamma(\nabla f(x_t) - m_t)^T(v_t - x_t) + \frac{LD^2\gamma^2}{2} \\ &= f(x_t) + \gamma \nabla f(x_t)^T(\tilde{v}_t - x_t) + \\ &+ \gamma(\nabla f(x_t) - m_t)^T(v_t - \tilde{v}_t) + \frac{LD^2\gamma^2}{2} \\ &= f(x_t) - \gamma g(x_t) + \gamma(\nabla f(x_t) - m_t)^T(v_t - \tilde{v}_t) + \frac{LD^2\gamma^2}{2} \\ &\leq f(x_t) - \gamma g(x_t) + \gamma D \|\nabla f(x_t) - m_t\|_2 + \frac{LD^2\gamma^2}{2} \end{aligned}$$

The last inequality holds thanks to Cauchy-Schwarz inequality. Recalling the previous lemma we have:

$$\|\nabla f(x_t) - m_t\|_2 \leq \frac{\gamma LD}{1-\beta}$$

And substituting we obtain:

$$f(x_{t+1} \leq f(x_t) - \gamma g(x_t) + \frac{LD^2\gamma^2}{1-\beta} + \frac{LD^2\gamma^2}{2}$$

Considering all the iterations from $t = 0, \dots, T-1$:

$$f(x_T) \leq f(x_0) - \sum_{t=0}^{T-1} \gamma g(x_t) + \frac{TLD^2\gamma^2}{1-\beta} + \frac{TLD^2\gamma^2}{2}$$

Isolating $g(\cdot)$:

$$\begin{aligned} \frac{1}{T} \sum_{t=0}^{T-1} g(x_t) &\leq \frac{f(x_0) - f(x_T)}{T\gamma} + \gamma\Big(\frac{LD^2\gamma}{1-\beta} + \frac{LD^2\gamma}{2}\Big) \\ &\leq \frac{f(x_0) - f(x^*)}{T\gamma} + \gamma\Big(\frac{LD^2\gamma}{1-\beta} + \frac{LD^2\gamma}{2}\Big) \end{aligned}$$

Where we consider that $f(x_T) \geq f(x^*)$
In conclusion:

$$\tilde{g}_T = \min_{1 \leq k \leq T} g(x_k) \leq \frac{f(x_0) - f(x^*)}{T\gamma} + \gamma\Big(\frac{LD^2\gamma}{1-\beta} + \frac{LD^2\gamma}{2}\Big)$$

If we let $\gamma = \sqrt{2(f(x_0) - f(x^*))/(C_\beta LD^2 T)}$ where $C_\beta = (3-\beta)/(1-\beta)$ we have:

$$\tilde{g}_T \leq \sqrt{\frac{2C_\beta LD^2(f(x_0) - f(x^*))}{T}}$$

□

3. Test on ImageNet

Our goal is to compare how the 4 algorithms presented before behave in adversarial attacks. In order to do so, we exploit the **ImageNet** dataset, in particular we use 99 different images belonging to 10 different classes of dataset. We choose to run our **Python** code in the *Colab* platform with *runtime* set to *GPU*.

Moreover we decide to attack three different models:

1. **Inception V3**: this model is reported to have a 77.9% top-1 accuracy and a 93.7% top-5 accuracy [1].

2. **Inception ResNet-V2**: this model is reported to have a 80.3% top-1 accuracy and a 95.3% top-5 accuracy [1].

3. **MobileNet V2**: this model is reported to have a 71.3% top-1 accuracy and a 90.1% top-5 accuracy [1].

3.1. Hyper-parameter selection

In order to be consistent with the related literature, we choose to set the hyper-parameters following the baseline given in [2], i.e. we choose to set maximum distortion ε, step-size γ and proportion in momentum β as:

Parameters	FGSM	PGD	MI-FGSM	FW
ε	0.05	0.05	0.05	0.05
γ	-	0.03	0.03	0.1
β	-	-	0.9	0.9

We fixed the maximum number of iterations at 20 but we have also set two different stopping criteria : for untargeted attacks we stopped our algorithms when the label assigned to the distorted image from the network become different from the original one, while for targeted attacks we stopped our methods when the label assigned from the network to the distorted image is identical to the one wanted.

*** stopping criteria ***
*** how we did backprop ? ***

4. Experiments

We divide our experiment in two sub sections, one for untargeted attack and one for targeted attack. As stated before, depending on the type of attacks, the algorithms reported above need to be slightly modified in order to perform correctly.

For untargeted attacks we want to maximize the loss with respect to the current image and the correct label. Doing so, the model will hopefully misclassify the image.

For targeted attacks we need to minimize the objective function and we are forcing the model to return a given label y_{target}.

As a mean to compare the results, we use the *success rate* (%) computed as the precentage of images misclassified by the models with any amount of confidence, the *average number of iterations* computed as a the average among the minimum number of iteration needed to output a misclassification by the model for each image and the *distortion* of the adversarial example x^*, computed as $\|x_{\text{ori}} - x^*\|_\infty$.

4.1. Untargeted attacks

In this section, we have three tables reporting the results for untargeted attacks.

As we notice before, the *FGSM* has no average iteration, since it is a one-step method. It also has the highest distortion in all three models, that coincides with $\varepsilon = 0.05$.

The *PGD* method turns out to need more iterations than the other two iterative algorithms.

Instead, the *MI-FGSM* has the lowest number of iteration, but has higher distortion that the other two iterative ones.

Finally, as written in Chen [2], the *FW-white* is a good trade-off between an acceptable average iterations and a very low distortion of the adversarial image.

*** to expand the comments ***

4.1.1 Inception V3

Attack	Success rate	AvgIt	Distortion
FGSM	70.71 %	-	0.05
PGD	80.81 %	8.39	0.0166
MI-FGSM	95.96 %	2.01	0.0344
FW-white	92.03 %	2.68	0.0065

4.1.2 Inception ResNet V2

Attack	Success rate	AvgIt	Distortion
FGSM	49.49 %	-	0.05
PGD	40.40 %	14.47	0.0162
MI-FGSM	96.97 %	2.48	0.0400
FW-white	92.93 %	4.16	0.0121

4.1.3 Mobile Net V2

Attack	Success rate	AvgIt	Distortion
FGSM	90.91 %	-	0.05
PGD	90.91 %	5.07	0.0200
MI-FGSM	97.98 %	1.46	0.0314
FW-white	95.96 %	1.98	0.0060

4.2. Targeted attacks

Here there are other three tables with the results for targeted attacks. As target class, we use the same for all the models and algorithms. We choose the *sea lion* class with index 150 of *ImageNet* database.

The *FGSM* is very unsuccessful, it does not fool any proposed model. In this case, we cannot compute any distortion value, since it is computed only for the adversarial images that actually works. Since it is not an iterative method it is very difficult in only one step to retrieve an image with the wanted label.

The *PGD* method attacks well in general, except for the *ResNetV2* model. However, it has the lowest success rate and the highest number of iteration with respect the other two iterative algorithms. The average distortion is really high due to the aggressive approach of the method : new points produced by PGD are often not in the convex set and the projection operator gives as output points that are very near the boundary of the convex set.

The only one to complete successfully every attack is the *MI-FGSM*. In our experiments, it has 100 % success rate, but the image distortion is the highest, coinciding with the maximum possible distortion $\varepsilon = 0.05$. On average it is also the method having the lowest number of iterations thank to the momentum technique accelerating the convergence of the algorithm.

The *FW-white* is very successful as well against the three models. It is still the result of a good balance between medium average iteration and a low value for the adversarial image distortion.

*** to expand the comments ***

4.2.1 Inception V3

Attack	Success rate	AvgIt	Distortion
FGSM	0.00 %	-	-
PGD	87.88 %	7.76	0.0378
MI-FGSM	100.00 %	5.42	0.0500
FW-white	100.00 %	5.86	0.0215

4.2.2 Inception ResNet V2

Attack	Success rate	AvgIt	Distortion
FGSM	0.00 %	-	-
PGD	57.58 %	13.40	0.0411
MI-FGSM	100.00 %	8.66	0.0500
FW-white	98.99 %	8.00	0.0256

4.2.3 Mobile Net V2

Attack	Success rate	AvgIt	Distortion
FGSM	0.00 %	-	-
PGD	98.99 %	4.25	0.0380
MI-FGSM	100.00 %	3.24	0.0500
FW-white	100.00 %	3.90	0.0159

5. Comparison between the algorithms

We now want to compare the four algorithms when changing the distortion ε and the maximum number of iterations. In particular, we are interested in how the success and the distortion rates change.

The values of ε we choose to analyze are: $0.010, 0.042, 0.074, 0.107, 0.139, 0.171, 0.203, 0.236,$ $0.267, 0.300$. The tested number of iterations are: $2, 4, 6, 8, 10, 12, 14, 16, 18, 20$.

The figures from 1 to 6 are organized in four different graphs. The ones in the first row have the range of ε as x-axis, and the ones in the second row have the range of maximum iterations. In the left column, there is the success rate as y-axis and in the right one there is the distortion rate.

For all the graphs, we keep the same colour legend. The blue line is for the *FGSM*, the red one is for the *PGD* method, the green line is for the *MI-FGSM* and the violet one is for the *FW-white*.

6. Conclusion

We can conclude that the expectation for *Inception V3* are met since our results are comparable with [2].

We can see empirically that the cost per iteration drops with the Frank Wolfe based algorithm as well as the amount of distortion.

The methods works best in *Inception V3* and also reach good results in both *Inception ResNet V2* and *MobileNet*.

References

[1] Keras applications.

[2] Jinghui Chen, Dongruo Zhou, Jinfeng Yi, and Quanquan Gu, editors. *A Frank-Wolfe Framework for Efficient and Effective Adversarial Attacks*.

[3] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li, editors. *Boosting Adversarial Attacks with Momentum*.

[4] Ian J. Goodfellow, Jonathon Shelns, and Christian Szegedy, editors. *Explaining and Harnessing Adversarial Examples*.

[5] Francesco Rinaldi. *Optimization for Data Science Notes*. 2021.

A. Graphs for different ε and number of iteration

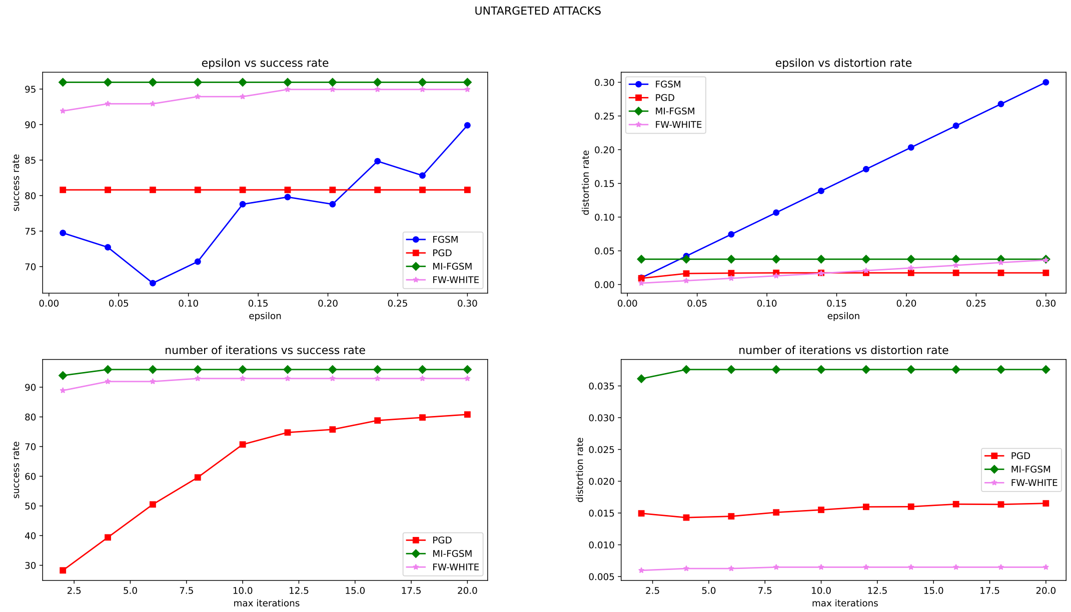

Figure 1. InceptionV3 - untargeted attack

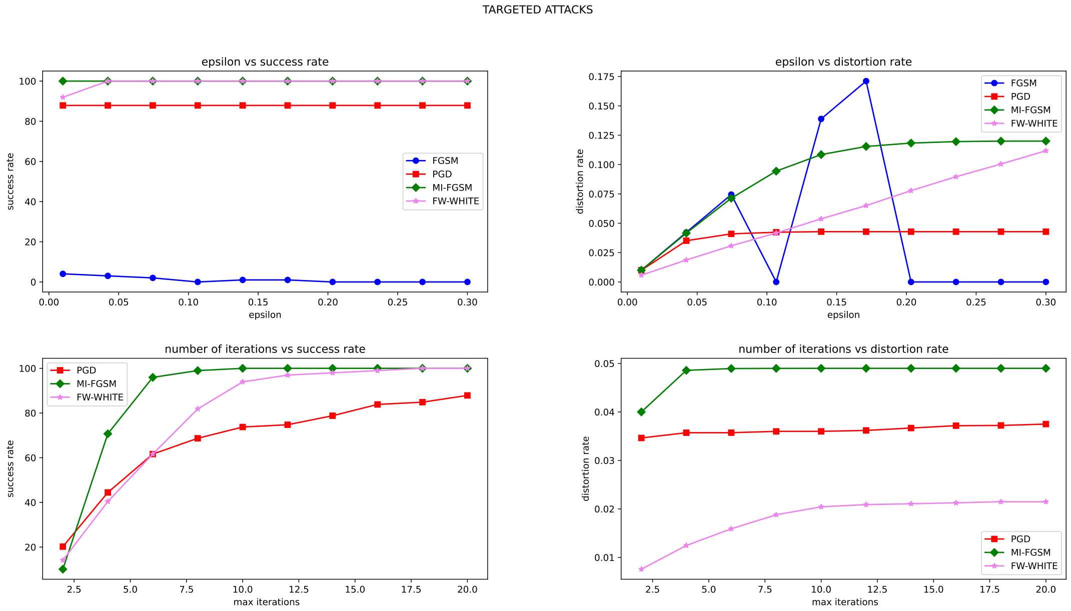

Figure 2. InceptionV3 - targeted attack

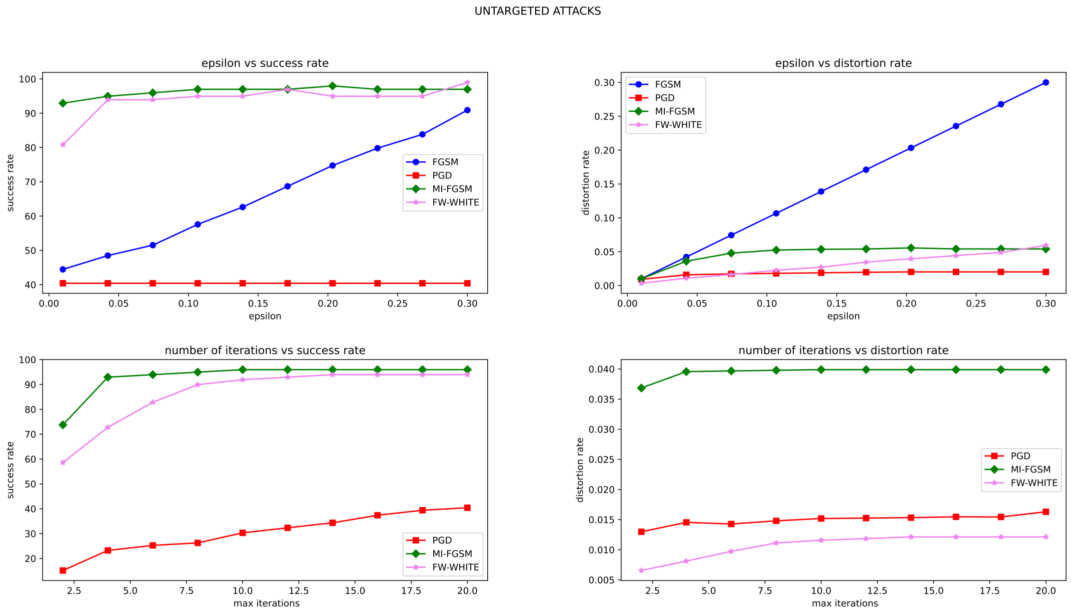

Figure 3. ResNet - untargeted attack

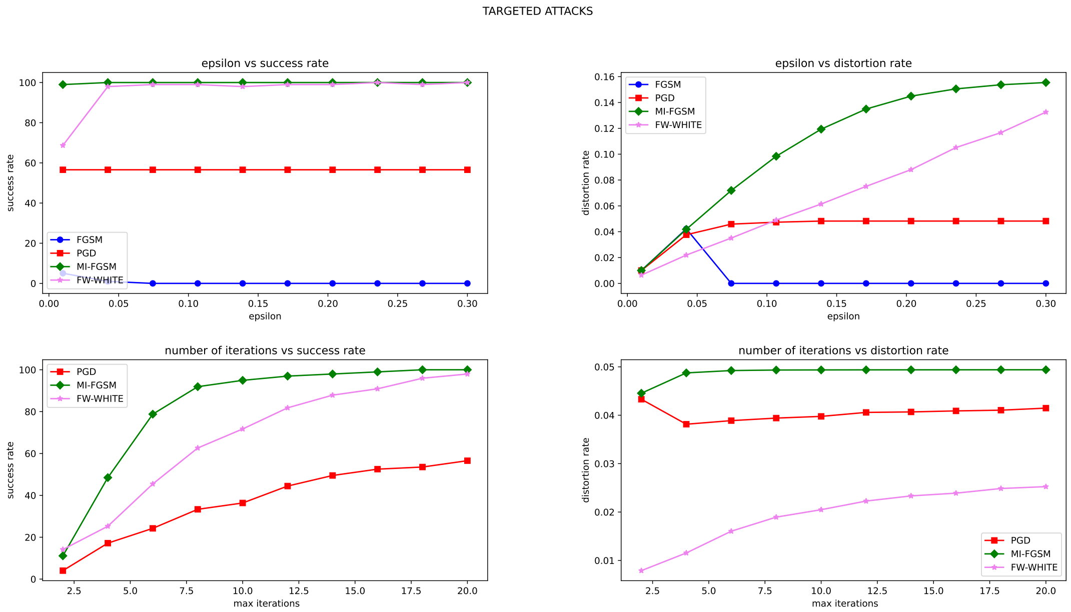

Figure 4. ResNet - targeted attack

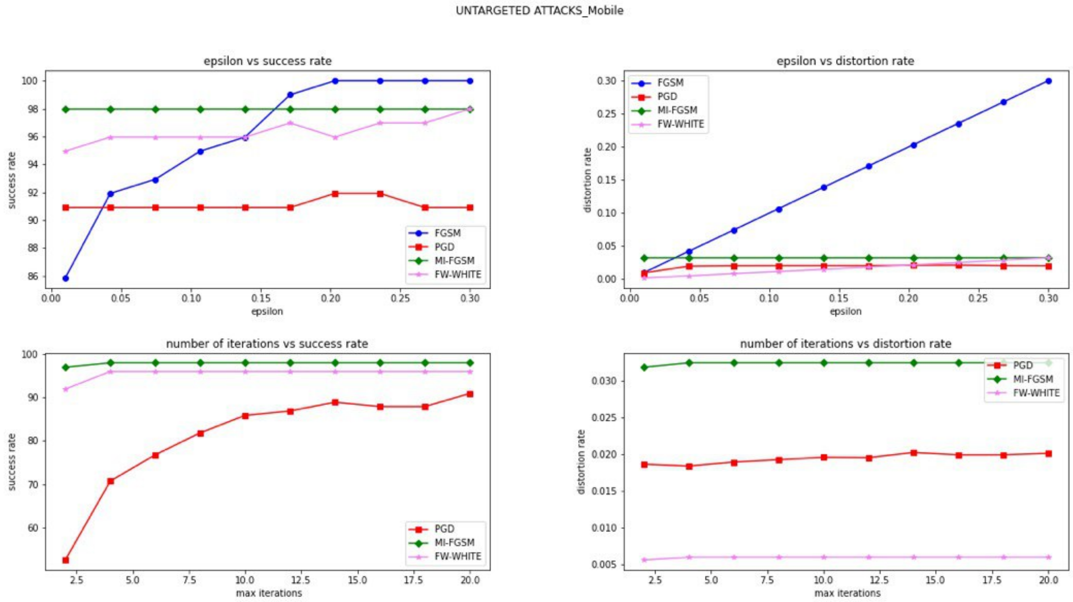

Figure 5. MobileNet - untargeted attack

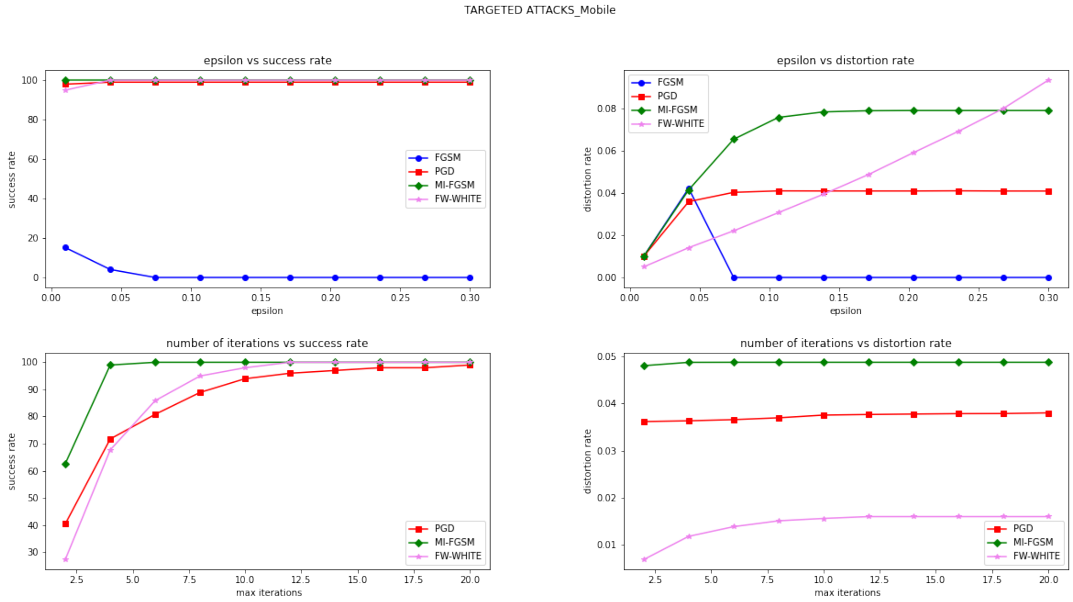

Figure 6. MobileNet - targeted attack