Optimization for Data Science - Final project
Study of first order Frank-Wolfe algorithms to solve constrained non-convex problems in the context of white-box adversarial attacks.

Luis Marcos Lopez Casines
luismarcos.lopezcasines@studenti.unipd.it

Laia Porcar Guillamon
laia.porcarguillamon@studenti.unipd.it

Philippe Robert
philippe.robert@studenti.unipd.it

Stefanija Galevska
stefanija.galevska@studenti.unipd.it

1. Introduction and motivation

Machine and Deep learning has been a fast rising practice in the last years. Thanks to the advancement of some technologies, we are now able to achieve tasks that were unthinkable for humans and software in the past. Convolutional neural networks used in computer vision tasks are a good example. Indeed, it is now considered the state-of-the-art technique to identify images. It attains about the same score or even higher than human beings for most of the tasks today. Moreover, the development of convolutional neural networks makes them so effective nowadays that mistakes are considered exceptions rather than the normality. Considering that, it could be easy to get fooled by the fallacy that they are robust to errors, thus once the model has been implemented there is no need to worry about improper behaviour. However, a new technique has arisen in the past years whose purpose is to take advantage of machine learning models by adding carefully crafted "noise" to create malicious attacks, commonly called Adversarial Attacks.

To prevent these attacks it is necessary to first know how they work and how effective and efficient they can be. For this reason, in this project, an attack on a model of the famous MNIST dataset is designed. To achieve small image perturbations while keeping high success rates, different variants of the Frank-Wolfe (FW) algorithm are used.

Frank-Wolfe's algorithm [1] needs little introduction. Its advantages are well known for dealing with constrained optimization problems, as an efficient alternative to projection dependent methods. This work is based on the problem posed by [2] and uses the following algorithms: Stochastic Frank-Wolfe (SFW) [3], Variance Reduction Non-convex Conditional Gradient Sliding (NCGS-VR) [4] and Faster First-Order Conditional Gradient Sliding Method (FCGS) [5]. The results obtained for each algorithm are shown in Table 2.

2. Adversarial attacks

An adversarial attack, or adversarial training, is an example that has been carefully computed to be misclassified by the model. In case of images, the noise introduced makes the image change in a degree that most humans can not discern. If we represent visually the precise direction to move the pixels of the original images to cause a misclassification, it looks like noise. However, it is a defined structure of a computed function of the parameters of the network. The effect of multiplying this structure by a small number and adding it to the original image is what causes the neural network to change its prediction to something else. In some cases, the model has even more confidence regarding its incorrect prediction than its correct prediction.

Adversarial attacks are a threat since they cause a malfunction to the model while humans might not be conscious of what has occurred. These attacks are not unique to image classification. They can also impact other areas like speech recognition and machine translation. However, in this work we approach uniquely adversarial attacks against image classification on deep neural networks.

Adversarial attacks based on access to the parameters/gradient. Adversarial attacks can be classified in two classes based on the information that can be accessed: white-box and black-box attacks. In this work we implement variants of the FW algorithm on a white-box attack, but we give a brief summary of these two.

When the adversary has full access to the information of the target model, the attack is known as white-box attack. By having access to the parameters of the model, first order methods can be used to back propagate the gradient descent to the inputs.

In contrast, in black-box attacks there is no access to the target model. It has only access to its inputs and outputs. Thus, given the inaccessibility of the architecture and

parameters of the targeted model, a substitute white-box model is made to represent the target model. This surrogate model trains adversarial examples that will be transferred to the target model to launch the attack on the black-box settings. Hence, by using a transfer-based strategy on a black-box one, even without any access to the model, it can be broken to implement the attack.

Adversarial attacks based on target. There are two types of adversarial attacks: untargeted and targeted. The difference between them consists on how we want the model to misclassify the true class. More precisely, with targeted attacks, we want the model to mistake the true class with another specific class, while with untargeted attacks we want it to mistake the true class with any other class.

In this work, we will only focus on targeted attacks given they present a slightly higher complexity than their counterparts. Nevertheless, it's important to keep in mind that both the procedure followed and the algorithms used can be unfolded for untargeted attacks. To achieve a targeted adversarial attack, we aim at minimizing the classification loss function $h\left(x, y_{\text{tar}}\right)$ with an input $x \in \mathbb{R}^d$ and the target class y_{tar}.

3. Frank-Wolfe method and non-convex applications

The Frank-Wolfe algorithm, also known as conditional gradient method, is a projection free method that at each iteration calls a first order oracle (IFO) and a linear minimization oracle (LO). The linear oracle dictates the direction of movement by minimizing a linear approximation of the objective function. As a consequence to this, there is no projection of the gradient onto the constrained domain the function is defined on, instead the problem is solved directly over such domain. It has been widely studied for convex smooth functions. However its applications on non-convex functions have begun to be studied only a few years ago. The importance of non-convex functions in machine and deep learning models and the need to incorporate non trivial constraints in such models makes the development of projection free methods imperative to approach these types of problems.

Work [3] presents a first approach to the stochastic projection free non-convex setting and also develop two variance reduced algorithms for the non-convex finite sum setting. For convex optimization some methods have been studied that outperform the classical FW. Among them, Conditional Gradient Sliding (CGS) stands out [6]. It integrates Nesterov's accelerated gradient and Frank-Wolfe methods. In [4] Chao Qu et al. study the CGS method for non-convex functions, calling it Non-convex Gradient Sliding (NCGS). In the same paper, they pose a variant of NCGS (NCGS-VR) based on the idea of variance reduction, which is stated to obtain faster convergence rates than NCGS in the finite sum setting and is hence used in this work. Authors of [5], find that the iteration complexity of NCGS-VR is suboptimal and propose a faster first-order conditional gradient descent method (FCGS) to further improve it.

4. Preliminaries

In this work we face a constrained optimization problem of the form of Eq. (1) focusing on two different problem settings:

$$\min_{x\in\Omega} F(x) := \begin{cases} \mathbb{E}_z[f(x,z)], & \text{(stochastic)} \\ \frac{1}{n}\sum_{i=}^{n} f_i(x). & \text{(finite-sum)} \end{cases} \tag{1}$$

In the stochastic setting $F(x) = \mathbb{E}_z[f(x,z)]$ where z is a random variable whose distribution $\mathcal{P}$ is supported on $\Xi \subset \mathbb{R}^p$. In the finite-sum setting $F(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$.

Assumptions. The algorithms we will work with assume F is non-convex and L-smooth, i.e. its gradient is Lipschitz continuous with constant L:

$$\|\nabla F(x) - \nabla F(y)\| \leq L\|x-y\|, \quad \forall x, y \in \Omega. \tag{2}$$

For the finite-sum setting we also assume that the individual functions $f_i (i \in [n])$ are L-smooth i.e.:

$$\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x-y\|, \quad \forall x, y \in \Omega. \tag{3}$$

Convergence criterion. The gradient norm is usually adopted as convergence criterion for non-convex unconstrained optimization problems. For the case of constrained non-convex problems like the one at hand, different authors have different methodologies. In [2] and [3] the Frank-Wolfe gap or duality gap is used as the convergence criterion:

$$\mathcal{G}(x) = \max_{u\in\Omega} \langle u - x, -\nabla F(x)\rangle. \tag{4}$$

As stated in [2] we always have $\mathcal{G}\left(x\right) \geq 0$ and x is a stationary point for the constrained optimization problem if and only if $\mathcal{G}\left(x\right) = 0$, which makes $\mathcal{G}\left(x\right)$ a perfect convergence criterion for Frank-Wolfe based algorithms.

Other authors, like [4] and [5] use a gradient mapping approach, which is defined as:

$$\mathcal{G}(x, \nabla F(x), \gamma) = \frac{1}{\gamma}(x - \psi(x, \nabla F(x), \gamma)), \tag{5}$$

where $\psi(x, \nabla F(x), \gamma)$ denotes a prox-mapping function which is defined as follows [5]:

$$\psi(x, \nabla F(x), \gamma) = \arg\min_{y\in\Omega}(\nabla F(x), y\rangle + \frac{1}{2\gamma}\|y-x\|^2. \tag{6}$$

Eq. (5) is a natural extension of gradient, since it reduces to it when there are no constraints [4]. Therefore, the criterion used in these cases is $\|\mathcal{G}(\theta, \nabla F(\theta), \gamma)\|^2 \leq \epsilon$.

Oracles. The following set of oracles are used to compare the convergence rate of the algorithms:

- Stochastic First Order Oracle (SFO): Given a function $F(\theta) = \mathbb{E}_\xi f(\theta, \xi)$ where $\xi \sim P$, a SFO takes a point θ and returns the stochastic gradient descent $G(\theta_k, \xi_k) = \nabla_\theta f(\theta_k, \xi_k)$, where ξ_k is an example drawn i.i.d. from P in the *k*-th call.
- Incremental First Order Oracle (IFO): For a problem setting $F(\theta) = \frac{1}{n}\sum_{i=1}^{n} f_i(\theta)$, an IFO selects $i \in [n]$ and returns the gradient $\nabla_\theta f_i(\theta)$.
- Linear Oracle (LO): For a set Ω the LO solves the following linear programming problem $\underset{x\in\Omega}{\operatorname{argmin}} \langle x, \nabla f(x_t)\rangle$. It can be seen as the minimization of the first-order Taylor expansion of $f(\cdot)$ at point x_t [2] :

$$\min_{x\in\mathcal{X}} f(x_t) + \langle x - x_t, \nabla f(x_t)\rangle . \tag{7}$$

We summarize the iteration complexity result to achieve an ϵ-solution for each of the algorithms in Table 1.

Algorithm	SFO/IFO complexity	LO complexity
FW	$\mathcal{O}\left(n/\epsilon^2\right)$	$\mathcal{O}\left(1/\epsilon^2\right)$
M. FW	$\mathcal{O}\left(n/\epsilon^{1/2}\right)$	$\mathcal{O}\left(1/\epsilon^{1/2}\right)$
SFW	$\mathcal{O}\left(1/\epsilon^4\right)$	$\mathcal{O}\left(1/\epsilon^2\right)$
FCGS	$\mathcal{O}\left(n + \frac{n^{1/2}}{\epsilon}\right)$	$\mathcal{O}\left(1/\epsilon^2\right)$
NCGS-VR	$\mathcal{O}\left(n + \frac{n^{2/3}}{\epsilon}\right)$	$\mathcal{O}\left(1/\epsilon^2\right)$

Table 1: Complexity comparison of algorithms discussed in the paper based on the number of oracles calls needed to achieve ϵ-solution [4], [5].

5. Algorithms

As previously mentioned our aim is to tackle a white-box attack problem using three variants of the FW algorithm. In consideration of the fact that [2] is our main reference paper, we implement the FW with momentum algorithm that the authors use for their white-box attack. Algorithm 1 is built upon the classic FW method adding a momentum term (m_t) that, according to [2], helps stabilizing the LO direction and leads to empirically accelerated convergence.

The LO oracle that Algorithm 1 calls, together with NCGS-VR and FCGS as we will see in this section, is in general expensive to compute. But for the constraint of our problem (refer to Section 4) it has a closed form solution that was derived in [2] and we present in Eq. (8):

$$v_t = -\epsilon \cdot \operatorname{sign}(m_t) + x_{\text{ori}} . \tag{8}$$

Eq. (8) is used in the *condg* procedure (Algorithm 2), which is in turned used in NCGS-VR and FCGS. Algorithm 2 is basically updating the input by performing the classic FW method [6].

Algorithm 1 Frank-Wolfe with momentum

Input: number of iterations T, step size $\{\gamma_t\}$
1: $x_0 = x_{ori}$, $m_{-1} = \nabla f(x_0)$
2: **for** $t = 0, ...T-1$ **do**
3: $\quad m_t = \beta \cdot m_{t-1} + (1-\beta)\cdot \nabla f(x_t)$
4: $\quad v_t = \operatorname{argmin}_{x\in\mathcal{X}} \langle x, m_t\rangle$
5: $\quad d_t = v_t - x_t$
6: $\quad x_{t+1} = x_t - \gamma_t d_t$
7: **end for**
Output: x_T

Algorithm 2 condg(g, u, γ, η)

1: $u_1 = u$, $t = 1$
2: v_t be an optimal solution for
$V_{g,u,\gamma}(u_t) = \max_{x\in\Omega}\langle g + \frac{1}{\gamma}(u_t - u), u_t - x\rangle$
3: If $V_{g,u,\gamma}(u_t) \leq \eta$, return $u^+ = u_t$.
4: Set $u_{t+1} = (1-\alpha_t)u_t + \alpha_t v_t$ where
$\alpha_t = \min\left\{1, \frac{\langle \frac{1}{\gamma}(u-u_t) - g, v_t - u_t\rangle}{\frac{1}{\gamma}\|v_t - u_t\|^2}\right\}$
5: Set $t \leftarrow t+1$ and go to step 2.

5.1. Stochastic Frank-Wolfe (SFW)

For a non-convex problem in a stochastic setting we implement the SFW method [3]. With this we acquire an estimate of the gradient, as opposed to obtaining the full gradient which is usually not possible in this type of setting. SFW can handle non-convex optimization problems of the form $F(x) = \mathbb{E}_z[f(x, z)]$.

As seen in Table 1 , the SFW has an SFO of $\mathcal{O}\left(1/\epsilon^4\right)$. Compared to the classic FW and all other algorithms stated in the same table, it is notable that the SFW has the worst convergence rate. Also being slightly worse that its convex counterpart [7].

Key parameters for this algorithm are the step size γ_t and the minibatch size b_t. Choosing them correctly is crucial to the convergence of the algorithm. As we will see in Section 6.2 we choose a fixed step size which later will be tuned for the purpose of our experiment.

Algorithm 3 Stochastic Frank-Wolfe (SFW)

Input : $x_0 \in \Omega$, number of iterations T, $\{\gamma_i\}_{i=0}^{T-1}$ where $\gamma_i \in$ [0,1] for all $i \in \{0, ..., T-1\}$, minibatch size $\{b_i\}_{i=0}^{T-1}$

1: **for** $t = 0, ..., T-1$ **do**
2: Uniformly randomly pick i.i.d. samples $\{z_1^t, ..., z_{b_t}^t\}$ according to the distribution $\mathcal{P}$
3: Compute $v_t = \text{argmax}_{v \in \Omega} \langle v, -\frac{1}{b_t} \sum_{i=1}^{b_t} \nabla f(x_t, z_i) \rangle$
4: Compute update direction $d_t = v_t - x_t$
5: $x_{t+1} = x + \gamma_t d_t$
6: **end for**

Output: Iterate x_α chosen uniformly random from $\{x_\alpha\}_{t=0}^{T-1}$

5.2. Variance Reduction Non-convex Conditional Gradient Slicing (NCGS-VR)

The variance reduction non-convex conditional gradient sliding algorithm proposed is based upon minimizing the finite-sum problem setting described in Eq.(1), where the number of training examples n is large as reported in [4]. The goal is to achieve a fast convergence rate by reducing the number of IFO calls, thus by reducing the number of gradient computation. With regard to the constraint given in Eq. (5), the NCGS-VR attains an IFO complexity $\mathcal{O}\left(n + \frac{n^{2/3}}{\epsilon}\right)$ and a LO complexity $\mathcal{O}\left(1/\epsilon^2\right)$ to find an optimal solution, as previously shown in Table 1.

Algorithm 4 Variance Reduction Non-convex Conditional Gradient Sliding (NCGS-VR)

Input : $\tilde{x}_0 = x_m^0 = x_0 \in \mathbb{R}^d$, epoch length m, stepsize λ_t, tolerance η, minibatch size b, iteration limit T, $S = \frac{T}{m}$

1: **for** s = 0,..., $S-1$ **do**
2: $x_0^{s+1} = x_m^s$
3: $g^{s+1} = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\tilde{x}^s)$
4: **for** t = 0, ..., $m-1$ **do**
5: Pick I_t uniformly from $\{1, ..., n\}$ with replacement such that $|I_t| = b$
6: $v_t^{s+1} = \frac{1}{b} \sum_{i \in I_t} (\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s)) + g^{s+1}$
7: $x_{t+1}^{s+1} = \text{condg}\left(v_t^{s+1}, x_t^{s+1}, \lambda_t, \eta\right)$
8: **end for**
9: $\tilde{x}^{s+1} = x_m^{s+1}$
10: **end for**

Output: x_α is chosen uniformly at random from $\{\{x_t^{s+1}\}_{t=0}^{m-1}\}_{s=0}^{S-1}$

The idea behind the technique of variance reduction is to lower the impact of the number of examples on the IFO complexity. As shown in line 3 of Algorithm 4, we compute the full gradient. It is then used in line 6 of the inner loop to reduce the stochastic gradient's variance.

To achieve a fast convergence rate, we set the minibatch gradient at $b = n^{2/3}$ and iteration length at $m = n^{1/3}$, as suggested in [4].

5.3. Faster First-Order Conditional Gradient Sliding Method (FCGS)

In order to improve the NCGS-VR algorithm, which has an IFO of $\mathcal{O}\left(n + \frac{n^{2/3}}{\epsilon}\right)$, a new stochastic first order conditional gradient sliding method, FCGS, with an IFO of $\mathcal{O}\left(n + \frac{n^{1/2}}{\epsilon}\right)$ was proposed by Gao and Huang [5].

The FCGS method described in Algorithm 5, is based on the constrained finite-sum minimization problem from Eq.(1). Each component function f_i of the sum belongs to S_2, which is a set containing $\sqrt{n}$ functions that have been randomly chosen, and n represents the total number of component functions [5].

FCGS makes use of the following variance reduction technique to estimate the full gradient:

$$v_k = \frac{1}{|S_2|} \sum_{i \in S_2} \left[\nabla f_i\left(x_k\right) - \nabla f_i\left(x_{k-1}\right) + v_{k-1}\right], \quad (9)$$

in order to reduce the variance introduced by the randomly selected component functions.

As it can be seen, Algorithm 5 updates the input sample x by using the conditional gradient method [4], which is defined in Algorithm 2. This method is useful since it accelerates the convergence speed and the algorithm ends when the Frank-Wolfe gap is smaller than the tolerance η.

Algorithm 5 Faster First-Order Conditional Gradient Sliding Method (FCGS)

Input : x_0, $q > 0$, $K > 0$, $\eta > 0$, $\gamma > 0$, n

1: **for** $k = 0, ..., K-1$ **do**
2: **if** mod$(k, q) = 0$ **then**
3: Compute the full gradient $v_k = \nabla F\left(x_k\right)$
4: **else**
5: Sample S_2 to compute $v_k = \frac{1}{|S_2|} \sum_{i \in S_2} \left[\nabla f_i\left(x_k\right) - \nabla f_i\left(x_{k-1}\right) + v_{k-1}\right]$
6: **end if**
7: $x_{k+1} = \text{condg}\left(v_k, x_k, \gamma_k, \eta_k\right)$
8: **end for**
9: Randomly choose x_α from x_k and return it

6. Experiments

6.1. Dataset

All algorithms are tested on an attack to the MNIST dataset and the results are compared. The MNIST dataset stands for the Modified National Institute of Standards and Technology dataset. It is made up of 60,000 28x28 pixel gray scale images of digits that were handwritten in the range of 0 to 9. Additionally, it counts with another 10,000 images that are used for testing purposes. For some years it has become a useful benchmark for testing and comparing models in machine and deep learning.

6.2. Experimental setting

In our experiment we focus on the specific form of Eq.(1) given by:

$$\min_{\|x-x_{\text{ori}}\|_p \leq \epsilon} h(x, y_{\text{tar}}), \tag{10}$$

where x_{ori} is the original image, $h(x, y_{\text{tar}})$ is the cross entropy loss function between the image and the target class and ϵ is a constraint in the maximum distortion, measured with the L_∞ norm [1]. The cross entropy loss function can be generally expressed as:

$$h = -\sum_{i=1}^{n} t_i \log(p_i), \tag{11}$$

where n is the number of classes, t_i is the truth label and p_i is the softmax probability for the i^{th} class.

This is therefore a non-convex constrained optimization problem since the non-linear activation functions in the hidden layers have the potential of introducing local minima in the optimization of the cross entropy loss function. Furthermore, it satisfies the conditions stated in Section 4.

In this work we are also interested in the derivative of the cross entropy loss function with respect to an input image:

$$\frac{\partial h}{\partial x} = -\sum_{i=1}^{n} t_i \frac{\partial}{\partial x}(\log(p_i)) = -\sum_{i=1}^{n} \frac{t_i}{p_i} \frac{\partial p_i}{\partial x}. \tag{12}$$

The written code for the attack runs on Python 3.10 with TensorFlow 2.0 [8]. A PC with Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz was used along with a RAM of 16.0 GB.

The model we attack is trained by ourselves trying to be as faithful as possible to the structure described in [2]. We use two convolutional layers with 32 units, followed by Maxpooling and another two convolutional layers with 64 units and Maxpooling. Finally two dense layers with 200 units and the output with 10 units. All activation functions are *relu* except for the output layer for which *softmax* is used.

[1] The L_p norm of a vector $x \in \mathbb{R}^d$ is defined as $\|x\|_p = \left(\sum_{i=1}^{d} x_i^p\right)^{1/p}$. The L_∞ norm of x is $\|x\|_\infty = \max_{i=1}^{d} |\theta_i|$.

The achieved accuracy on the test set is always around 99%. From the test samples that the model classifies correctly we take 1000 to use for the attack and randomly generate a target class for each of them.

The first attack is performed with the FW with momentum algorithm from [2] with the aim of reproducing their results and make sure the experimental setup is well established. The hyperparameters of the model are chosen according to the information in [2], where a grid-search is performed. A grid-search is always a good approach in these kind of problems, but at a high computational cost. Due to this fact and other time constraints grid-search was not performed in any of the algorithms we present. More comments on this in Section 6.3.

The execution of the SFW algorithm was carried out following the pseudocode of Algorithm 3. A size one batch for our problem is a pixel of the image. The pixels with respect to which we calculate the gradient at each iteration are uniformly randomly chosen from the image. We run into some difficulties when implementing the gradient of the loss function with respect to the minibatch using the TensorFlow library. A workaround was to directly calculate the whole gradient and take the minibatch from it. This can be done given the properties of the gradient of the loss function with respect to each of the images. In a real application this would, of course, not be a viable approach because the gradient of the loss function could be costly to compute and in the case it could be done, there would be no point on taking the minibatch afterwards. This work is done however with educational purposes and this sidestep doesn't influence the final objective of it.

Algorithms 4 and 5 are implemented according to what is shown in their corresponding pseudocodes. The step size, minibatch and tolerance hyperparameters of all algorithms are fixed for simplicity. The optimal values are found by tuning them.

In order to compare the performance of the different methods, in terms of both efficiency and effectiveness some information is extracted from their execution. Effectiveness is measured as the ratio of successful attacks to total attacks. This takes into consideration that we define a maximum image distortion. This constraint takes the value of 0.3, which is the same [2] employs. Distortion measures the L_∞ norm of the difference of the altered image and the original one. Oracle calls, such as LO, SFO and IFO are counted. For both distortion and oracle calls averages over the whole sample of target images are given as results. Finally the total time to run the attack on 1000 images is provided. Time is not an ideal variable for comparing algorithms since it depends on the optimization of the code and the platform where it is executed. However, it can serve as a reference to know which algorithms perform faster in general terms.

6.3. Results and discussion

Obtained results are shown in Table 2 and the tuned hyperparameters can be found in Table 3.

The FW with momentum algorithm is clearly the winner in terms of efficiency and efficacy. Not only is its average success rate much higher than for the other methods, but it also needs a lower number of calls to both IFO and LO oracles. This said, we could not reach the exact results reported in [2], even though they are very close.

Algorithm	M. FW	SFW	NCGS-VR	FCGS
ASR(%)	99.7	57.0	73.8	66.8
IFO/SFO	47.8	75544.4	601.5	170.9
LO	4.8	186.361	416.7	91.6
Distortion	0.24	0.28	0.29	0.29
Time (s)	200.0	17978.0	7543.0	2478.0

Table 2: Results of the attack for the four algorithms employed. ASR stands for Attack Success Rate.

As for the SFW algorithm which has the worse performance, it should be noted that it highly depends on the step-size chosen. Indeed, for higher batch sizes the calculation of the gradient is closer to that of the full gradient. However there is no point on taking very big batch sizes, since the usefulness of this method lies on the fact that it approximates the calculation of the full gradient when this is impossible to compute at once. The batch size used guarantees a performance as similar as possible to that of the other methods. By increasing the number of iterations used (300) even more, better results could potentially be achieved. Nevertheless, this would require unmanageable execution times, as can be seen already in Table 2.

Algorithm	M. FW	SFW	NCGS-VR	FCGS
Step size	0.45	0.4	0.08	0.03
Batch Size		400		
Momentum	0.5			
Tolerance			1.1	18

Table 3: Tuned hyperparameters for the four methods used.

The FCGS method is found to perform faster than NCGS-VR, as stated in [5], but is less effective for our problem. Notice that, the results of the IFO/SFO calls for each method follow the predicted order in regard to the convergence rate stated in Table 1. The FW with momentum is the algorithm with the lowest number of calls while SFW is the one with the highest. The number of LO calls is of the same order for SFW, NCGS-VR, FCGS; while for FW with momentum is considerably lower. This is also the expected behaviour if we consider the results in Table 1.

The distortion for SFW, NCGS-VR and FCGS is near our constraint of 0.3. Less distortion would have meant less successful attacks, so we decided to push it till the limit to get the best possible success rate. The FW with momentum method achieves almost perfect success rates with around 20% less distortion than the other methods, meaning it minimizes the objective function more effectively.

7. Conclusions and future scope

In this work we have implemented four variants of the FW algorithm to perform a white-box attack on a model we explicitly trained on the MNIST dataset. Our results show that the best algorithm for this specific problem is the FW with momentum from [2]. It was also found that FCGS is indeed faster than NCGS-VR, at the cost of somewhat less effectiveness. Therefore it could be well suited for tasks where efficiency is valued over effectiveness.

Nevertheless, the results could be improved by the implementing grid search and a non-fixed step size. It is also important to say that the code could be vastly optimized if used for non-academic purposes.

The results obtained lead us to think that the three methods studied, apart from the FW with momentum, are not the most suitable for the minimization of the loss function that concerns us. This is because these methods are intended for functions that benefit from a stochastic or finite-sum approach. The stochastic approach does not provide any advantage to problem solving since the gradient of the loss function with respect to the images is relatively fast to compute, provided that they are 28x28 pixels, as is the case with this dataset. Although it is true that the cross entropy loss function can be expressed as a finite sum, only one of the functions that compose it provides meaningful information, that corresponding to the target class, while the other component functions will be multiplied by zero. So by calling IFO oracles on these component functions we are not providing much feedback to the update of the descent direction.

In conclusion, the use of SFW, FCGS and NCGS-VR would probably outperform, in terms of efficiency, the FW with momentum algorithm in the case of a more complex loss function, that could be divided into a sum of more meaningful functions. Also, when having very large images, taking the whole gradient at once would be computationally very expensive. This would come at the cost of less effectiveness, as has been said throughout this work, but more effective methods may not provide feasible approaches under the mentioned circumstances.

References

[1] Marguerite Frank and Philip Wolfe. "An algorithm for quadratic programming". In: *Naval research logistics quarterly* 3.1-2 (1956), pp. 95–110.

[2] Jinghui Chen et al. "A Frank-Wolfe framework for efficient and effective adversarial attacks". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 04. 2020, pp. 3486–3494.

[3] Sashank J Reddi et al. "Stochastic frank-wolfe methods for nonconvex optimization". In: *2016 54th annual Allerton conference on communication, control, and computing (Allerton)*. IEEE. 2016, pp. 1244–1251.

[4] Chao Qu, Yan Li, and Huan Xu. "Non-convex conditional gradient sliding". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4208–4217.

[5] Hongchang Gao and Heng Huang. "Can stochastic zeroth-order Frank-Wolfe method converge faster for non-convex problems?" In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3377–3386.

[6] Guanghui Lan and Yi Zhou. "Conditional Gradient Sliding for Convex Optimization". In: *SIAM Journal on Optimization* 26.2 (2016), pp. 1379–1409. DOI: 10.1137/140992382.

[7] Elad Hazan and Haipeng Luo. "Variance-reduced and projection-free stochastic optimization". In: *International Conference on Machine Learning*. PMLR. 2016, pp. 1263–1271.

[8] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.