Zeroth-Order Methods for Adversarial Machine Learning

Optimization for Data Science

Federico Zanotti[†], Lorenzo Corrado[††]

University of Padua

September 16, 2021

1 Introduction

Deep Neural Networks (DNNs) have made many breakthroughs in different areas of AI such as image classification, object detection and speech recognition. However, recent studies have shown that DNNs are vulnerable to adversarial examples [1]. Adversarial examples represent slightly modified data, often images, to which a tiny perturbation is applied. This perturbation is almost imperceptible for a human but that could deceive even a well-trained DNN, pushing it to misclassify the label of the data. Depending on how much information is available to an attacker, it is possible to classify adversarial attacks into two classes: white box attack, where the adversary has full access to the target model, and black-box attack where the attacker does not have full access to the target model. In particular, in the white-box setting the attacker knows the model architecture, the parameters, inputs and outputs and it can query the model and calculate the gradient. In the other case, more challenging, the attacker can only access the inputs and outputs of the target model but not its internal configurations. In this project we will focus our attention on the black-box setting. In general, we can formalize the generation of adversarial examples as a constrained finite-sum minimization problem [2] as follows:

$$\min_{\mathbf{x}\in\Omega} F(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x}) \tag{1}$$

where $\Omega \in \mathbb{R}^d$ denotes a closed convex feasible set, each component function f_i is smooth and non-convex and n represents the number of component functions. Our task is to implement and compare three different algorithms to solve a problem related to the one shown in Eq.1, using only zeroth-order information:

- Zeroth-Order Stochastic Conditional Gradient Method (ZSCG);
- Faster Zeroth-Order Frank-Wolfe Method (FZFW);
- Faster Zeroth-Order Conditional Gradient Sliding Method (FZCGS).

This report is organized as follows: in the next section we will introduce the key ideas of the Frank-Wolfe algorithm, which forms the foundation for the construction of the algorithms used in this work, and next we will focus on variants that exploit only zero-order information. Then we will analyze each algorithm to be implemented also describing its pseudo-code. Finally, we will describe the actual problem we had to face, its implementation and discuss the results obtained to see if the theoretical insights are also respected in practice.

2 Frank-Wolfe Method

In general, a simple algorithm to optimize the problem in Eq.1 is the projected gradient descent method. This method takes a step along the gradient direction and then performs a projection to satisfy the constraint. One problem of this approach is that it is very computational expensive, especially the projection step, and it may not be feasible in the huge-scale framework. The Frank-Wolfe algorithm (also called conditional gradient method) is a much more efficient method when we deal with constrained

[†]federico.zanotti@studenti.unipd.it (ID: 2007716)
[††]lorenzo.corrado@studenti.unipd.it (ID: 2020623)

problems. This algorithm was first proposed in 1956 but has regained considerable attention in recent years due to its nice properties. Two of the main advantages of this method are: it is relatively easy to implement and it does not require the projection step into the constrained set to find a feasible direction of descent. Instead of taking the projection step, this algorithm solve, at each iteration, a linear subproblem to make the solution lie in the feasible set Ω. This subproblem is the so called linear minimization oracle (LMO).

$$\mathbf{v}_t \in \underset{\mathbf{v}\in\Omega}{\operatorname{argmin}} \langle \nabla f(\mathbf{x}_t), \mathbf{v} \rangle \tag{2}$$

There have been many different improvements and many different variants of this algorithm [2] but they are all based on the availability of the gradient ∇f_i, so based on first-order information. Given the fact that we are in the black-box setting, only zeroth-order information can be used. The basic idea of zeroth-order methods is to approximate the gradient using an estimator, the approximation is generally obtained by focusing on the difference of function values $f(x)$ under small disturbance on x. Generally, this class of methods can be useful not only when the gradient is not available but also when its computation is too expensive.

2.1 Zeroth-Order Gradient Estimators

Following the notation in [3], we will consider three different gradient estimators:

- Two-point gaussian random gradient estimator:

$$\hat{\nabla} f_i(\mathbf{x}) = (d/\nu)[f_i(\mathbf{x} + \nu \mathbf{u}_i) - f_i(\mathbf{x})]\mathbf{u}_i, \qquad (\text{for } i \in [n]) \tag{3}$$

  Where $\nu > 0$ is the smoothing parameter, $\mathbf{u}_i \sim N(\mathbf{0}, \mathbf{I})$ and d is the number of optimization variables.

- Averaged gaussian random gradient estimator:

$$\hat{\nabla} f_i(\mathbf{x}) = (d/(\nu q)) \sum_{j=1}^{q} [f_i(\mathbf{x} + \nu \mathbf{u}_{i,j}) - f_i(\mathbf{x})]\mathbf{u}_{i,j} \qquad (\text{for } i \in [n]) \tag{4}$$

  Where $\nu > 0$ is the smoothing parameter, $\{\mathbf{u}_{i,j}\}_{j=1}^{q}$ are q i.i.d. samples drawn from $N(\mathbf{0}, \mathbf{I})$ and d is the number of optimization variables.

- Coordinate-wise gradient estimator:

$$\hat{\nabla} f_i(\mathbf{x}) = (1/(2\mu)) \sum_{j=1}^{d} [f_i(\mathbf{x} + \mu \mathbf{e}_j) - f_i(\mathbf{x} - \mu \mathbf{e}_j)]\mathbf{e}_j \qquad (\text{for } i \in [n]) \tag{5}$$

  Where $\mu > 0$ is the smoothing parameter, $\mathbf{e}_j \in \mathbb{R}^d$ denotes the basis vector and d is the number of optimization variables.

3 Algorithms

As we have already said, the methods covered in the report exploit only zero-order information with the estimators discussed above. The goal of the authors of the three algorithms analyzed in this work was to highlight that zero-order methods can share almost the same iteration complexity as their first-order counterparts, as well as some constants with respect to the input dimension and that it is also further improvements can be made.

3.1 Preliminaries

We will list some basic assumptions and some important definitions for the convergence analysis.

Assumption 1. The component function f_i, with $i \in [n]$, is L-smooth as follows:

$$||\nabla f_i(\mathbf{x}) - \nabla f_i(\mathbf{y})|| \leq L||\mathbf{x} - \mathbf{y}||, \qquad \forall \mathbf{x}, \mathbf{y} \in \Omega \tag{6}$$

We know that if a function $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth, then:

$$f(\mathbf{x}) \leq f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle + \frac{L}{2}||\mathbf{x} - \mathbf{y}||^2 \tag{7}$$

for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, this is a classic bound on the value of functions with Lipschitz continuous gradients.

Assumption 2. The diameter of the feasible set Ω is bounded such that:

$$\max_{\mathbf{x},\mathbf{y}} ||\mathbf{y} - \mathbf{x}|| \leq D_\Omega \tag{8}$$

for some $D_\Omega > 0$.

Assumption 3. Assume that the variance of the stochastic gradient $\nabla f_i(\mathbf{x})$ is bounded as follows:

$$\frac{1}{n}\sum_{i=1}^{n} ||\nabla f_i(\mathbf{x}) - \nabla F(\mathbf{x})||^2 \leq \sigma^2 \tag{9}$$

Where $\sigma > 0$.

Convergence criterion. The convergence criterion used for the standard Frank-Wolfe method is the Frank-Wolfe gap, defined as follows:

$$\mathcal{G} = \max_{\mathbf{u} \in \Omega} \langle \mathbf{u} - \mathbf{x}, -\nabla F(\mathbf{x}) \rangle. \tag{10}$$

For the conditional gradient sliding method which incorporates an acceleration technique the following gradient mapping is used as the convergence criterion:

$$\mathcal{G}(\mathbf{x}, \nabla F(\mathbf{x}), \gamma) = \frac{1}{\gamma}(\mathbf{x} - \psi(\mathbf{x}, \nabla F(\mathbf{x}), \gamma)), \tag{11}$$

where $\psi(\mathbf{x}, \nabla F(\mathbf{x}), \gamma)$ denotes a prox-mapping function which is defined as follows:

$$\psi(\mathbf{x}, \nabla F(\mathbf{x}), \gamma) = \operatorname*{argmin}_{\mathbf{y} \in \Omega} \langle \nabla F(\mathbf{x}), \mathbf{y} \rangle + \frac{1}{2\gamma}||\mathbf{y} - \mathbf{x}||^2 \rangle \tag{12}$$

where $\gamma > 0$ is an hyper-parameter.

Oracle Model. To compare the iteration complexity of different algorithms is used the following oracle models:

- Function Query Oracle (FQO): FQO samples a component function and returns its function value $f_i(\mathbf{x})$.
- Linear Oracle (LO): LO solves a linear programming problem and returns $\operatorname*{argmax}_{\mathbf{u} \in \Omega} \langle \mathbf{u}, \mathbf{v} \rangle$.

3.2 Zeroth-Order Stochastic Conditional Gradient Method (ZSCG)

The pseudo-code of the first algorithm, proposed in [4], is the following:

Algorithm 1 Zeroth-Order Stochastic Conditional Gradient Method (ZSCG)

Require: $z_0 \in \Omega$, smoothing parameter $\nu > 0$, non-negative sequence $\alpha_k > 0$, positive integer sequence m_k, iteration limit $N \geq 1$

for $k = 1, \dots, N$ **do**

1. Generate $u_k = [u_{k,1}, \dots, u_{k,m_k}]$, where $u_{k,j} \sim N(0, I_d)$, call the stochastic oracle to compute m_k stochastic gradient $G_\nu^{k,j}$ and take their average:

$$\hat{G}_\nu^k \equiv \hat{G}_\nu^k(z_{k-1}, \xi_k, u_k) = \frac{1}{m_k} \sum_{j=1}^{m_k} \frac{F(z_{k-1} + \nu u_{k,j}, \xi_{k,j}) - F(z_{k-1}, \xi_{k,j})}{\nu} u_{k,j} \tag{13}$$

2. Compute

$$x_k = \underset{u \in \Omega}{\operatorname{argmin}} \langle \hat{G}_\nu^k, u \rangle \tag{14}$$

$$z_k = (1 - \alpha_k) z_{k-1} + \alpha_k x_k \tag{15}$$

end for

return z_k

As we can see from the pseudo-code, this algorithm is different from the classic CG algorithms in that it estimates the gradient using only zeroth-order information and a probability distribution on the number of iterations $\{1, \dots, N\}$ to provide the output. In addition, the estimator used by the authors is the one represented in Eq.4 and constitutes a variance reduction technique for the gradient estimate. Indeed, when the gradient is not available and it is replaced with an estimate, an error appears inside it, the variance reduction techniques try to make this error converge to 0.

Theorem 1 *Let $\{z_k\}_{k \geq 0}$ be generated by Algorithm 1 and Assumptions 1, 2 and 3 hold. Let f be nonconvex, bounded from below by f^*, and let the parameters of the algorithm be set as:*

$$\nu = \sqrt{\frac{2B_{L\sigma}}{N(d+3)^3}}, \; \alpha_k = \frac{1}{\sqrt{N}}, \; m_k = 2B_{L\sigma}(d+5)N, \; \forall k \geq 1 \tag{16}$$

for some constant $B_{L\sigma} \geq \max\{\sqrt{B^2 + \sigma^2}/L, 1\}$ and a given iteration bound $N \geq 1$. Then we have:

$$\mathbb{E}[g_\Omega^R] \leq \frac{f(z_0) - f^* + LD_\Omega^2 + 2\sqrt{B^2 + \sigma^2}}{\sqrt{N}} \tag{17}$$

where R is uniformly distributed over $\{1, \dots, N\}$ and g_k is the Frank-Wolfe gap.

Corollary 1.1 *With the same setting of Theorem 1, the total number of calls to the zeroth-order stochastic oracle and linear subproblems required to be solved to find an ξ-stationary point are, respectively bounded by:*

$$O\Big(\frac{d}{\epsilon^4}\Big) \quad O\Big(\frac{1}{\epsilon^2}\Big) \tag{18}$$

It is important to say that the hyper-parameters proposed by the authors within the convergence analysis are useful for the proofs but, from a practical point of view, they may not be feasible. In particular, the value of m_k that represent the number of different estimates of the gradient, which plays a key role in the convergence analysis of the algorithm, generally assumes too large values which increased the computational cost of the algorithm. For this reason, within our implementations, we choose the value of the hyper-parameters trying to optimize the trade-off between theory and practice.

3.3 Faster Zeroth-Order Frank-Wolfe Method (FZFW)

The pseudo-code of the second algorithm, proposed in [2], is the following:

Algorithm 2 Faster Zeroth-Order Frank-Wolfe Method (FZFW)

Require: $\mathbf{x}_0$, $q > 0$, $\mu > 0$, $K > 0$, n
  for $k = 0, \dots, K-1$ **do**
    if $mod(k, q) = 0$ **then**
      Sample S_1 without replacement to compute $\hat{\mathbf{v}}_k = \hat{\nabla} f_{S_1}(\mathbf{x}_k)$
    else
      Sample S_2 with replacement to compute $\hat{\mathbf{v}}_k = \frac{1}{|S_2|} \sum_{i \in S_2} [\hat{\nabla} f_i(\mathbf{x}_k) - \hat{\nabla} f_i(\mathbf{x}_{k-1}) + \hat{\mathbf{v}}_{k-1}]$
    end if
    $\mathbf{x}_k = \underset{\mathbf{u} \in \mathcal{X}}{\operatorname{argmax}} \langle \mathbf{u}, -\hat{\mathbf{v}}_k \rangle$
    $\mathbf{d}_k = \mathbf{u}_k - \mathbf{x}_k$,
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \gamma_k \mathbf{d}_k$
  end for
  return $\mathbf{x}_k$

As we can see from the pseudo-code, one of the major differences with respect to Algorithm 1 is that at each q iteration the gradient is estimated through:

$$\hat{\nabla} f_{S_1}(\mathbf{x}_k) = \sum_{j=1}^{d} \frac{f_{S_1}(\mathbf{x}_k + \mu_j e_j) - f_{S_1}(\mathbf{x}_k - \mu_j e_j)}{2\mu_j} e_j \tag{19}$$

while in the other iterations:

$$\hat{\mathbf{v}}_k = \frac{1}{|S_2|} \sum_{i \in S_2} [\hat{\nabla} f_i(\mathbf{x}_k) - \hat{\nabla} f_i(\mathbf{x}_{k-1}) + \mathbf{v}_{k-1}] \tag{20}$$

where S_1 and S_2 represent randomly selected samples. The fact that the coordinate-wise gradient estimator is used brings the number of function queries to $O(d)$, compared to the estimator used in the previous algorithm. Also it is important to note that this estimator is deterministic while the previous one is stochastic. Furthermore, this algorithm implements a different variance reduction technique for the gradient estimation; this technique is called Stochastic Path Integrated Differential Estimator (SPIDER) and it allows a much more accurate gradient estimation.

Theorem 2 *Under Assumption 1, if the parameters are chosen as* $|S_1| = n$, $q = |S_2| = \sqrt{n}$, $\gamma_k = \gamma = \frac{1}{D\sqrt{K}}$ *and* $\mu = \frac{1}{\sqrt{dK}}$, *then Algorithm 2 satisfies:*

$$\mathbb{E}[\mathcal{G}(\boldsymbol{x}_\alpha)] \leq \frac{D(F(\boldsymbol{x}_0) - F(\boldsymbol{x}_*) + 11L)}{\sqrt{K}} \tag{21}$$

where $\mathcal{G}(\boldsymbol{x}_\alpha)$ *is the Frank-Wolfe gap.*

Corollary 2.1 *With the same setting of Theorem 2, the total number of calls to the zeroth-order oracle and linear subproblems required to be solved to find an stationary point are, respectively bounded by:*

$$O\Big(\frac{n^{1/2}d}{\epsilon^2}\Big) \quad \Big(\frac{1}{\epsilon^2}\Big) \tag{22}$$

3.4 Faster Zeroth-Order Conditional Gradient Sliding Method (FZCGS)

The pseudo-code of the last algorithm, proposed by [2], is the following:

Algorithm 3 Faster Zeroth-Order Conditional Gradient Sliding Method (FZCGS)

Require: $\mathbf{x}_0$, $q > 0$, $\mu > 0$, $K > 0$, $\eta > 0$, $\gamma > 0$, n
  for $k = 0, \dots, K-1$ **do**
    if $mod(k, q) = 0$ **then**
      Sample S_1 without replacement to compute $\hat{\mathbf{v}}_k = \hat{\nabla} f_{S_1}(\mathbf{x}_k)$
    else
      Sample S_2 with replacement to compute $\hat{\mathbf{v}}_k = \frac{1}{|S_2|}\sum_{i \in S_2}[\hat{\nabla} f_i(\mathbf{x}_k) - \hat{\nabla} f_i(\mathbf{x}_{k-1}) + \hat{\mathbf{v}}_{k-1}]$
    end if
    $\mathbf{x}_{k+1} = condg(\hat{\mathbf{v}}_k, \mathbf{x}_k, \gamma_k, \eta_k)$
  end for
  return x_k

Algorithm 4 $u^+ = condg(\mathbf{g}, \mathbf{u}, \gamma, \eta)$

  $\mathbf{u}_1 = \mathbf{u}$, $t = 1$
  $\mathbf{v}_t$ be an optimal solution for $V_{\mathbf{g},\mathbf{u},\gamma}(\mathbf{u}_t) = \max_{\mathbf{x} \in \Omega}\langle \mathbf{g} + \frac{1}{\gamma}(\mathbf{u}_t - \mathbf{u}), \mathbf{u}_t - \mathbf{x}\rangle$
  while $V_{\mathbf{g},\mathbf{u},\gamma}(\mathbf{u}_t) > \eta$ **do**
    $\mathbf{u}_{t+1} = (1 - \alpha_t)\mathbf{u}_t + \alpha_t \mathbf{v}_t$, where $\alpha_t = \min\{1, \frac{\langle \frac{1}{\gamma}(\mathbf{u} - \mathbf{u}_t) - \mathbf{g}, \mathbf{v}_t - \mathbf{u}_t\rangle}{\frac{1}{\gamma}||\mathbf{v}_t - \mathbf{u}_t||^2}\}$
    $t = t + 1$
  end while
  return $\mathbf{u}^+ = \mathbf{u}_t$

As we can see, the gradient estimation in this algorithm uses the same technique seen in the previous one. The main difference that is the way in which $\mathbf{x}_k$ is updated. For the update this algorithm uses a sub-routine called conditional gradient sliding, the pseudo-code is shown in Algorithm 4. Basically, the procedure iteratively updates $\mathbf{x}_k$ until the Frank-Wolfe gap is less than a certain tolerance η.

Theorem 3 *Under Assumption 1, if the parameters are chosen as* $|S_1| = n$, $q = |S_2| = \sqrt{n}$, $\mu = \frac{1}{\sqrt{dK}}$, $\gamma_k = \gamma = \frac{1}{3L}$ *and* $\eta_k = \eta = \frac{1}{K}$ *then Algorithm 3 has the following convergence rate:*

$$\mathbb{E}[||\mathcal{G}(\mathbf{x}_\alpha, \nabla F(x_\alpha, \gamma))||^2] \leq \frac{\Big(3(F(\mathbf{x}_0) - F(\mathbf{x}_*) + 1) + 7L)6L\Big)}{K} \tag{23}$$

Corollary 3.1 *With the same setting of Theorem 3, the total number of calls to the zeroth-order oracle and linear subproblems required to be solved to find a stationary point are, respectively bounded by:*

$$O\Big(\frac{n^{1/2}d}{\epsilon}\Big) \quad O\Big(\frac{1}{\epsilon^2}\Big) \tag{24}$$

4 Experiments

The aim of this project is to verify the performance of the three aforementioned algorithms on the task of generation of adversarial attacks on black-box DNNs. In particular, given a black-box DNN $f : \mathbb{R}^d \to \mathbb{R}^c$ and a dataset $\{(\mathbf{x}_i, y_i : \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{0, 1, \dots, c\}\}_{i=1}^n$, we want to optimize the following objective function:

$$\min_{||\delta||_\infty \leq s} \frac{1}{n}\sum_{i=1}^{n} \max\{f_{y_i}(\mathbf{x}_i + \delta) - \max_{j \neq y_i} f_j(\mathbf{x}_i + \delta), 0\} \tag{25}$$

where $f(\mathbf{x}) = [f_1(\mathbf{x}), \dots, f_c(\mathbf{x})]$ denotes the output of the last layer before conducting the softmax operation and $\mathbf{x}_i$, in this case, represents an image. Intuitively, we want to minimize the difference between the score estimated by the DNN in the true label and the maximum score estimated in a label corresponding to a different class, therefore the type of attack to be carried out against our network is

untargeted. In the end, the task is to find the universal adversarial perturbation $\delta \in \mathbb{R}^d$ such that the network ends up misclassifying the example assigning it to a whatever different class, without specifying which one it is. The constraint of the problem represents the order of magnitude of the distortion that can be applied to the images, in our case $s = 0.1$.

4.1 Dataset and Setup

In this project we used the MNIST dataset, a popular benchmark dataset for learning, classification and computer vision systems. This dataset contains 55,000 images in the training set, 5,000 images in the validation set and 10,000 images in the test set; each image represents an handwritten digits (0-9) with dimension 28x28 pixels in gray-scale. We normalized the dataset by dividing it by 255, so each pixel is in a range between [0,1] and then flattened each image. Given that this optimization problem is quite computational demanding both from the memory allocation and computational complexity points of view, to run our code we used Google Colab Pro that provides Intel Xeon E5-2699v4 CPU with Nvidia Tesla T4 GPU. To build the DNN we used Keras, an open source Python library specialized for deep learning applications. For the implementation of the three algorithms we used the libraries included in one of the most used Python distribution, Anaconda.

4.2 Model

Following the experiment in [2], we used the same pretrained DNN for MNIST dataset as the black-box model. To ensure maximum comparability we only defined the architecture of the DNN and imported the weights of the pre-trained net. The original architecture is:

Layer Type	Size
Convolution + ReLU	3x3x32
Convolution + ReLU	3x3x32
Max Pooling	2x2
Convolution + ReLU	3x3x64
Convolution + ReLU	3x3x64
Max Pooling	2x2
Fully Connected + ReLU	200
Fully Connected + ReLU	200
Softmax	10

The original DNN was trained for 50 epochs with an initial learning rate of 0.1, a momentum of 0.9 and a batch-size of 128. It is important to say that we slightly changed the architecture adding a reshape layer to pass in input to the DNN the row images. While this does not affect the performance of the model, on the other hand it allows us to manage vectors instead of matrices, which has made implementation much easier and more intuitive.

4.3 Implementation Details

Our implementation of the three algorithms does not differ much from the one proposed in the pseudo-codes. A particularity that is important to underline is how we can obtain the LMO. Indeed, as shown in [5] the derivation of the LMO in this case has an explicit formula true for $p \geq 1$, in our case the L-infinity norm ($p = \infty$). The problem can be rewritten as:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & ||\mathbf{x} - \mathbf{x}_{ori}||_p \leq s \end{aligned} \tag{26}$$

Where $\mathbf{x}_{ori}$ represent the original images. The constraint set $\Omega = \{\mathbf{x} \text{ s.t. } ||\mathbf{x} - \mathbf{x}_{ori}||_p \leq s\}$ is a bounded convex set when $p \geq 1$. Given that, the closed-form solution for the LMO in our case is:

$$\mathbf{v}_t = \underset{u \in \Omega}{\operatorname{argmin}} \langle \mathbf{g}_t, \mathbf{u} \rangle = -s \cdot sign(\mathbf{g}_t) + \mathbf{x}_{ori} \tag{27}$$

Where $\mathbf{g}_t$ denotes the gradient of the objective function. Within our algorithms we have also added a code string that reproject, at each iteration, the pixels of the new perturbed images in the range between [0,1]; this procedure is called clipping. When the clip is used, the objective function does not achieve as low values as when it is not used. For research purposes we produced plots relating to both the case in which the clip is applied and the case in which the clip is not applied. However, it is important to state that if we wanted to conduct a real adversarial attack it would be necessary to apply it. Otherwise for the defense against an attack it would be sufficient to constrain the network to receive only images with pixels in a specific range, in this case between [0,1]. In our implementation we also added a stopping criterion based on the predictions of the DNN. In practice, at the end of each iteration we use the DNN to make predictions using the new perturbed images as input, if the DNN misclassifies the labels over a certain percentage (generally set at 100%) we terminate the algorithms prematurely. As regards the implementation of the averaged gaussian random gradient estimator we followed part of the experiments present in [3] in which, to have more accurate estimates, the authors generate q random directions for the gradient estimation for each component function f_i. In our opinion this is the only solution to obtain decent performance for the ZSCG algorithm when a small batch of examples is used. Indeed, in the figure below it is possible to see that the performance of this algorithm improves only considering large batches of examples if we set $q = 1$, but this increases the computational time in a way that makes the algorithm practically unusable.

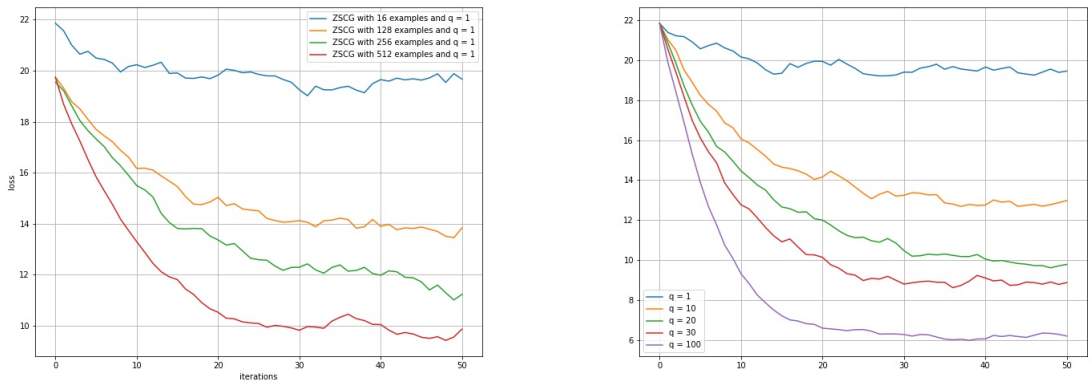

Figure 1. *ZSCG with different sample sizes (left) and different q with $n = 16$ (right).*

Finally, as said before, some of the hyper-parameters defined by the authors were not actually usable for the implementation, so we choose to define the hyper-parameters by doing grid-search and running the algorithms for 50 iterations on $n = 36$ examples coming from the same class, for computational reasons. The results are shown in the following table.

	Hyper-parameters				
Algorithm	α	ν	γ	μ	η
ZSCG	10^{-1}	10^{-5}	-	-	-
FZFW	-	-	10^{-1}	10^{-3}	-
FZCGS	-	-	10^{-1}	10^{-1} or 10^{-3}	10^{-5} or 10^{-7}

4.4 Results

To compare the performance of the algorithms we considered the speed of convergence. Furthermore, to have a fair comparison between the algorithms and to have a more general idea of their behavior, we decided to attack both sets of images taken randomly from the same class and randomly taken from different classes, in both cases the images come from the test set and therefore have not been used to train the DNN. First, let's see what is the behavior of the three algorithms in the generation of adversarial attacks on small batches of images and we will see the differences in performance when the perturbation is clipped and when it is not.

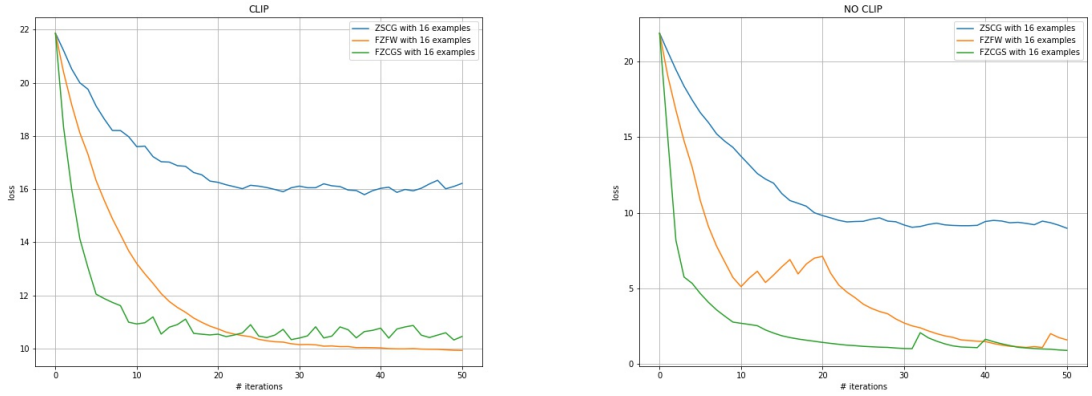

Figure 2. *Comparison with $n = 16$ from class 4.*

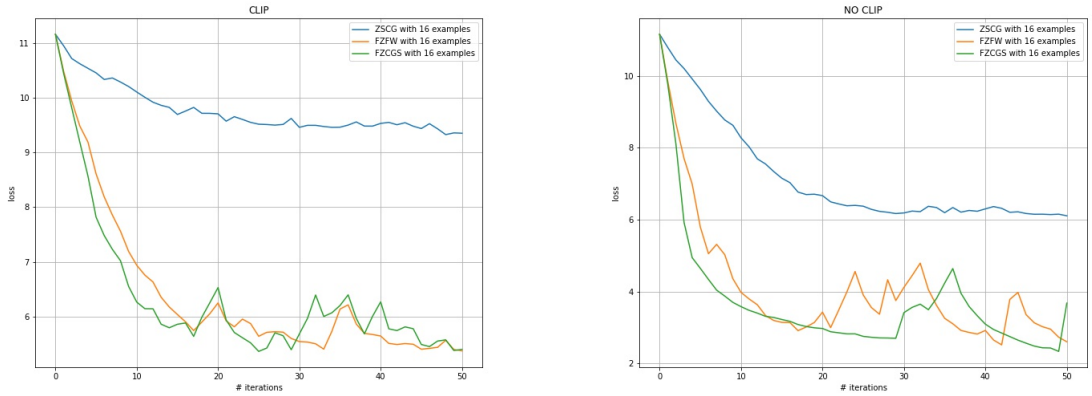

Figure 3. *Comparison with $n = 16$ from different classes.*

The convergence results of the three zeroth-order methods after 50 iterations are shown in Fig.1 and Fig.2. As claimed in [2], we confirm that the algorithm FZFW and FZCGS seem to have a better speed of convergence with respect to classic ZSCG. This result doesn't change significantly when we consider examples of the same or different classes. However, it is important to note that the performance of ZSCG compared to the other algorithms is worse given the fact that the gradient estimate performed only on $n = 16$ is too noisy, even with $q = 30$. As we will see, increasing the number of examples to be attacked is positive in terms of performance for the ZSCG algorithm as in this way it is able to have a more accurate estimate of the gradient but this significantly increases the convergence time. A visible thing is that the fact of clipping the perturbations somehow regularizes the progress of the minimization of the objective function, but at the same time does not allow to reach sufficiently low values to generate a significant number of adversarial examples. Now we can see the behavior of the algorithms when attacked with a larger and more representative batch of examples and without clip.

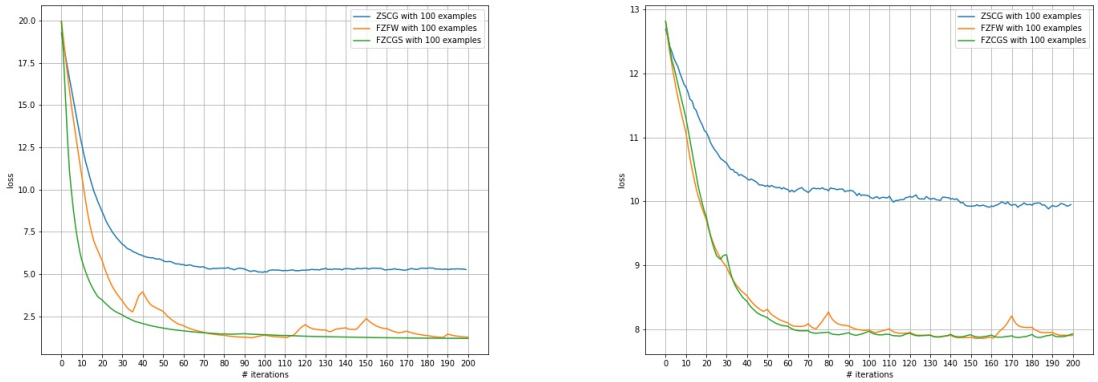

Figure 4. *Comparison with $n = 100$ of class 4 (left) and different classes (right).*

As we can see, the theoretical results quite reflect the empirical ones. It is important to note how much more difficult the attack is when we use batches of examples from different classes where, among other things, FZSCG does not perform significantly better than FZFW. Finally, as a last example we have attacked a batch with an even larger size, the examples inside it are from the same class and also in this case the clip has not been applied.

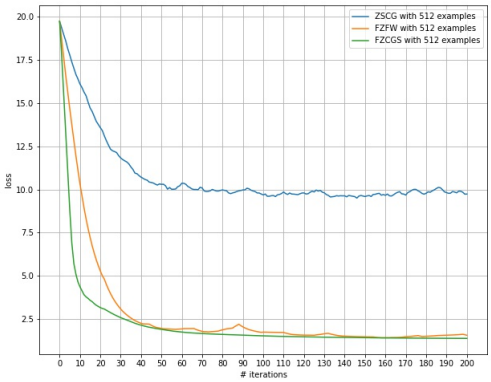

Figure 5. *Comparison with $n = 512$ of class 4.*

We specify that in this case we used $q = 1$ in the implementation of ZSCG so the gradient estimator we used is the averaged gaussian random gradient estimator as specified in [2]. As we can see, the performance of ZSCG, although not at the level of the other two algorithms, is not bad. However, it must be remembered that in order to obtain accurate estimates of the gradient, it requires a rather large number of examples. Now we can observe some adversarial examples generated by the three algorithms.

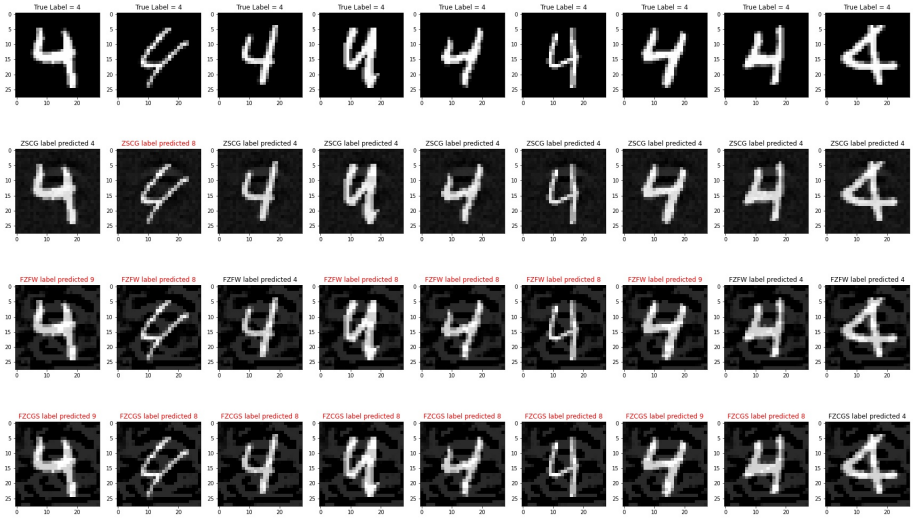

Figure 6. *Generated adversarial examples from the same class.*

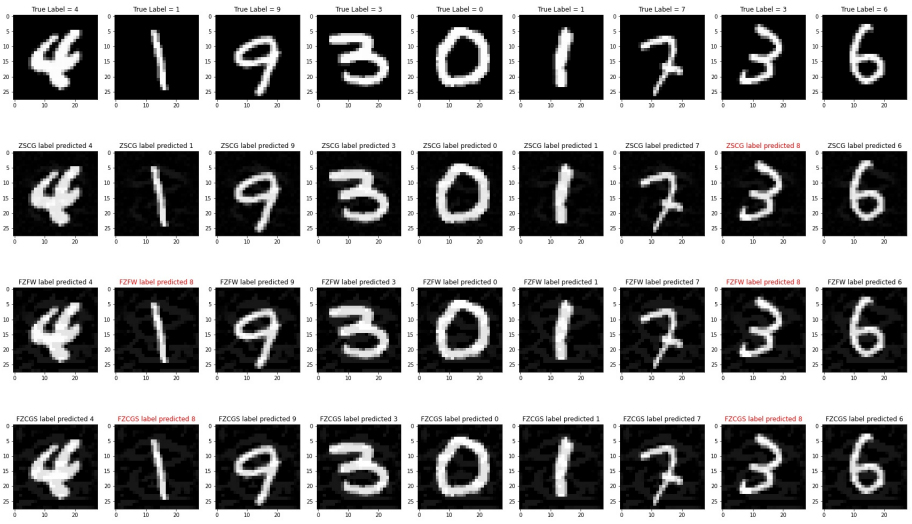

Figure 7. *Generated adversarial examples from different classes.*

5 Conclusions

The aim of this project was to analyze the problem related to the generation of adversarial examples on black-box DNNs. We presented the problem of adversarial attacks and cited their main characteristics. Then we analyzed a general formalization of the constrained problem to which the generation of adversarial examples can be traced. So we looked at the Frank-Wolfe algorithm, commonly used to tackle constrained optimization problems, and analyzed the theoretical features of the three algorithms that use only zeroth-order information. The goal was to verify whether the results demonstrated in the theory are also observable in practice. Given the results of our tests we can confirm what the authors demonstrated in [2]: the ZSCG algorithm is the slower algorithm and instead the FZSCG algorithm turns out to be the fastest as regards the convergence speed. Furthermore, while the FZFW and FZCGS algorithms manage to perform well even when the batch of images to be attacked is small, the ZSCG algorithm turns out to have poor performance. This also demonstrates a certain superiority of the coordinate-wise estimator over the averaged random gradient estimator.

References

1. Carlini, Nicholas and Wagner, David. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pp. 39–57, (2017).

2. Gao, Hongchang and Huang, Heng. Can Stochastic Zeroth-Order Frank-Wolfe Method Converge Faster for Non-Convex Problems?. In *International Conference on Machine Learning*, pp. 3377–3386, (2020).

3. Liu, Sijia and Kailkhura, Bhavya and Chen, Pin-Yu and Ting, Paishun and Chang, Shiyu and Amini, Lisa. Zeroth-order stochastic variance reduction for nonconvex optimization. In *arXiv preprint arXiv:1805.10367*, (2018).

4. Balasubramanian, Krishnakumar and Ghadimi, Saeed. Zeroth-order (non)-convex stochastic optimization via conditional gradient and gradient updates. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 3459–3468, (2018).

5. Chen, Jinghui and Zhou, Dongruo and Yi, Jinfeng and Gu, Quanquan. A Frank-Wolfe framework for efficient and effective adversarial attacks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp.3486–3494, (2020).

6. Kurakin, Alexey and Goodfellow, Ian and Bengio, Samy. Adversarial machine learning at scale. In *arXiv preprint arXiv:1611.01236*, (2016).

7. Huang, Feihu and Tao, Lue and Chen, Songcan. Accelerated stochastic gradient-free and projection-free methods. In *International Conference on Machine Learning*, pp. 4519–4530, (2020).

8. Ji, Kaiyi and Wang, Zhe and Zhou, Yi and Liang, Yingbin. Improved zeroth-order variance reduced algorithms and analysis for nonconvex optimization. In *International conference on machine learning*, pp. 3100–3109, (2019).