Zeroth-Order Methods for Adversarial Attack

Optimization for Data Science

Mincato Emanuele Caregari Alberto

Abstract

Adverse machine learning is a learning method that aims to fool neural network models by providing misleading input. Thus, an adversarial attack is an input to a machine learning model that is purposely designed to cause the model to make an error in its predictions, despite looking like a valid input to a human being. In particular, in this project we focus on the generation of adversary attacks for the MNIST dataset, so we are dealing with an image classification problem. To generate the adversarial attack we implemented and tested three different Zeroth-Order Frank-Wolfe algorithms.

1. Introduction

Nowadays, the correctness of the neural networks used for image recognition is sometimes superior to the human brain. This holds until we add some specific noise to the image; a human being can easily recognise this as noise, but it is not as easy for a machine. The aim of this project is to implement and test three different variants of the vanilla Frank-Wolfe (FW) method in order to generate adversary attacks. The three algorithms are:

- Zeroth-order Stochastic Conditional Gradient Method (ZSCG)
- Faster Zeroth-Order Frank-Wolfe Methods (FZFW)
- Faster Zeroth-Order Conditional Gradient Method (FZCGS)

In data science problems it often happens that we have to minimize a function and just as often happens that we have access neither to the gradient of the function nor the correct function, but we can only we can only estimate the approximation of the function function altered by a certain noise, i.e.

$$\min_{x\in\Omega} f(x), \quad \text{where} \quad f(x) = \mathbb{E}[F(x,\xi)]$$

Where $\Omega \in \mathbb{R}^n$ is a closed convex set and f is the true non necessarily convex loss function. The scenario previously described, were only the output of the method that we want to attack can be observed is called *black-box* attack. A *white-box* attack instead is a scenario where the attacker has complete access to the target model, including the model's architecture and parameters. Furthermore, an attack can have different objectives: a *targeted* attack tends to force the algorithm to misclassify the prediction in favour of a specific class, an *untargeted* attack on the other hand forces the algorithm to misclassify each prediction regardless the class. In this work we will focus on the *untargeted* approach.

2. Frank-Wolfe

The classic FW algorithm is based on approximating the objective by the first-order Taylor approximation. In our studies we work with three

different variation of the classical FW algorithm, basically exploring zeroth-order alternatives for the gradient approximation. The main ideas behind the functioning of these methods are:

- Choose a point $x_k \in \Omega$, the feasible set.
- Find $\hat{x}_k = \arg\max_{x\in\Omega}\langle\nabla f(x_k), (x - x_k)\rangle$ solving the linear maximization problem.
- Defining next point as $x_{k+1} = x_k + \alpha_k(\hat{x_k} - x_k)$, where α_k a chosen step.

In this case, the next point is chosen at a position between the current point and the final point, updated by the formula.

Frank-Wolfe is projection-free algorithm as it solve, at each iteration, a Linear Minimization problem over the constraint feasible set Ω. In contrast, methods based on the projection of the gradients are usually more computational expensive than the projection-free ones, and thus they usually could not be used for huge data problems. To solve the $\arg\max$ query is used a method described in Chen et al. (2018); thanks to some geometrical properties of the problem, we can find:

$$\hat{x}_k = -\epsilon \cdot sign(\nabla f(x_k)) + x_k\,.$$

The Linear Oracle (LO) could also be used if the first-grade information is not available (as we have done in this work), in fact we have sobstituted the correct evaluation of the gradient with three different approximations.

2.1. Oracle model

In order to compare the iteration complexity between the algorithms we need to define the following "*oracle*" functions:

- Function Query Oracle (FQO): samples a component function and returns its function value $f_i(\mathbf{x})$.
- Incremental First-Order Oracle (IFO): samples a component function and returns its gradient $\nabla f_i(\mathbf{x})$.
- Linear Oracle (LO): solves a linear programming problem and returns $\arg\max_{\mathbf{u}\in\Omega}\langle\mathbf{u}, \mathbf{v}\rangle$ or $\arg\min_{\mathbf{u}\in\Omega}\langle\mathbf{u}, \mathbf{v}\rangle$ depending on the algorithm.

2.2. Gradient Estimators

Since the gradient of the function f is not available we must estimate it. The idea is to approximate the gradient by taking the difference between the function values under small modification on x. The most used technique are the two-point Gaussian random gradient estimator (Nesterov and Spokoiny (2017)) and the coordinate wise gradient estimator (Lian et al. (2016)). Below we reported three different methods implemented in(Lian et al. (2016)), even if in the algorithms we used the equations taken from (Liu et al. (2018)), the only difference is in the notation and some constants. It is important to note that the use of these approximations could lead to potentially divergent iteration sequences. To avoid this problem, mainly introduce by non-decay gradient (due to stochasticity of the approximation) or bias estimate, is crucial to apply variance reduction techniques in the algorithms. After having found the optimal direction of the gradient we updated the variable x (image), even in this case we updated the value considering a weighted average with the previous steps.

2.2.1 Kiefer-Wolfowitz stochastic approximation

Kiefer-Wolfowitz stochastic approximation (KWSA) is a Coordinate-wise gradient estimator. It consists in approximating the gradient by sampling the objective function along the canonical basis vectors.

$$\mathbf{g}\left(\mathbf{x}_t; \mathbf{y}\right) = \sum_{i=1}^{d} \frac{F\left(\mathbf{x}_t + c_t\mathbf{e}_i; \mathbf{y}\right) - F\left(\mathbf{x}_t; \mathbf{y}\right)}{c_t}\mathbf{e}_i \qquad (1)$$

where c_t is a time-decaying sequence and $\mathbf{e}_i$ denotes the basis vector where only the i-th element is 1 and all the others are 0. This methods requires d samples (directions) at each step, in order

to evaluate the gradient. A slightly different version is used to estimate the gradient for the FZFW and FZCGS algorithms. (Details in the next section).

2.2.2 Random directions gradient estimator

This is a two-point (Gaussian) random gradient estimator, that involves estimating the directional derivative along a randomly sampled direction from an appropriate probability distribution.

$$\mathbf{g}\left(\mathbf{x}_t;\mathbf{y},\mathbf{z}_t\right)=\frac{F\left(\mathbf{x}_t+c_t\mathbf{z}_t;\mathbf{y}\right)-F\left(\mathbf{x}_t;\mathbf{y}\right)}{c_t}\mathbf{z}_t \qquad (2)$$

where z_t is a random vector sampled from a Gaussian distribution, such that $\mathbb{E}[\mathbf{z}_t\mathbf{z}_t^T] = \mathbf{I}_d$, and c_t is a carefully chosen time-decaying sequence.

2.2.3 Improved Random directions gradient estimator

The main difference, of this improved version, is that now are sampled $m < d$ directional derivatives. Basically is simply an averaged version of the previous technique.

$$\mathbf{g}\left(\mathbf{x}_t;\mathbf{y},\mathbf{z}_t\right)=\frac{1}{m}\sum_{i=1}^{m}\frac{F\left(\mathbf{x}_t+c_t\mathbf{z}_t;\mathbf{y}\right)-F\left(\mathbf{x}_t;\mathbf{y}\right)}{c_t}\mathbf{z}_t \qquad (3)$$

This method is used for the implementation of the ZSCG alghorithm.

3. Algorithms

3.1. Assumptions

First of all, it is necessary to state some basic assumptions that are applied in every algorithm and theorem in this work. These assumptions are quite common in the stochastic optimization context, refer to the behaviour of the estimation function, the boundness of the feasible set and the correctness of our approximation function. More specifically they are:

first assumption *The feasible set Ω is bounded and its diameter is $D \in \mathbb{R}^d$.*

second assumption *function F has a Lipschitz continuous gradient with constant L, almost surely for any ξ, i.e.*

$$\|\nabla F(y,\xi)-\nabla F(x,\xi)\|\le L\|y-x\|$$

third assumption *let $\|\cdot\|$ be a norm on $\mathbb{R}^d$. For any $x \in \mathbb{R}^d$, the zeroth-order oracle outputs an estimator $F(x,\xi)$ of $f(x)$ such that*

- $\mathbf{E}[F(x,\xi)] = f(x)$,
- $\mathbf{E}[\nabla F(x,\xi)] = \nabla f(x)$,
- $\mathbf{E}[\|\nabla F(x,\xi) - \nabla f(x)\|_*^2] \le \sigma^2$, *where $\|\cdot\|_*$ denotes the dual norm.*

Moreover we can consider an additional assumption, related with the type of the random vectors that we will generate:

fourth assumption the $\mathbf{z}_t$ are drown from a distribution μ such that $M(\mu) = \mathbf{E}[\|\mathbf{z}_t\|^6]$ is finite and for every vector $\mathbf{g} \in \mathbb{R}^d$ exists a function $s(d) : \mathbb{N} \to \mathbb{R}_+$ such that

$$\mathbb{E}[\|\langle \mathbf{g},\mathbf{z}_t\rangle \mathbf{z}_t\|]^2 \le s(d)\|\mathbf{g}\|^2$$

3.2. Convergence criterion

We used the Frank-Wolfe gap as convergence criterion. The FW gap is define as:

$$G=\max_{\mathbf{u}\in\Omega}(\mathrm{u}-\mathrm{x},-\nabla F(x,\xi)\rangle.$$

Since in the FZCGS we used a conditional gradient sliding method that incorporates the Nesterov's acceleration technique we implemented the following gradient mapping as convergence criterion, implemented in (Qu et al. (2018),Lian et al. (2016)):

$$G(\mathrm{x},\nabla F(\mathrm{x}),\gamma)=\frac{1}{\gamma}(\mathrm{x}-\psi(\mathrm{x},\nabla F(\mathrm{x}),\gamma)).$$

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

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

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

3.3. Zeroth-order Stochastic Conditional Gradient

The first algorithm that we have implemented is ZSCG, described in Balasubramanian and

Ghadimi (2018). This is a zero-order method in the sense that does not need the correct value of the gradient to work, but it needs only to generate some random directions (m_k in the following pseudo code) and calculate the increment over that direction. Finally the gradient is approximated by dividing the sum of those directional increments by the total number of chosen directions. The last part of the algorithms is the classical Frank-Wolfe method: calling for the LO in point 3 and finding the new point by a weighted sum in point 4.

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

Input: $z_0 \in \mathcal{X}$, smoothing parameter $\nu > 0$, non-negative sequence α_k, positive integer sequence , iteration limit $N \geq 1$ and probability distribution $P_R(\cdot)$ over $\{1, \ldots, N\}$

1: **for** k = 0,.., N **do**

2: Generate $u_k = [u_{k,1}, \ldots, u_{k,m_k}]$, where $u_{k,j} \sim N(0, I_d)$, call the stochastic oracle to compute m_k stochastic gradient $G_\nu^{k,j}$ according to (1.4) and take their average:
$\bar{G}\nu^k \equiv \bar{G}\nu(z_{k-1}, \xi_k, u_k) =$

$$\frac{1}{m_k} \sum_{j=1}^{m_k} \frac{F(z_{k-1} + \nu u_{k,j}, \xi_{k,j}) - F(z_{k-1}, \xi_{k,j})}{\nu} u_{k,j}$$

3:
$$x_k = \underset{u \in \mathcal{X}}{\operatorname{argmin}} \left\langle \bar{G}_\nu^k, u \right\rangle$$

4:
$$z_k = (1 - \alpha_k) z_{k-1} + \alpha_k x_k$$

5: **end for**

Output: Generate R according to $P_R(\cdot)$ and output z_R.

Theorem 3.1 *Under Assumption seen in 3.1, let f be non convex, bounded from below by f^*, and if the parameters are chosen as*

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

for some constant $B_{L\sigma} \geq \max\sqrt{B^2 + \sigma^2}/L, 1$ and a given iteration bound $N \geq 1$ then Algorithm 1 satisfies:

$$\mathbb{E}[g_X^R] \leq \frac{f(z_0 - f^* + LD_x^2 + 2\sqrt{B^2 + \sigma^2}}{\sqrt{N}} \tag{4}$$

where R is uniformely distribuited over $\{1, ..., N\}$ and g_k is define in . Hence, the total number of calls to the zeroth-order stochastic oracle and linear subproblems required to be solved to find an ϵ-stationary point of problem described before are, respectively bounded by:

$$\mathcal{O}\left(\frac{d}{\epsilon^4}\right), \mathcal{O}\left(\frac{d}{\epsilon^2}\right)$$

3.4. Faster Zeroth-Order Frank-Wolfe

The pseudo code for the Faster Zeroth-Order Frank-Wolfe algorithm proposed by Gao and Huang (2020) is summarized in Algorithm 2.

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

Input: x_0, $q > 0$, $\mu > 0$, $K > 0$, n

1: **for** k = 0,.., K - 1 **do**
2: **if** mod(k,q) = 0 **then**
3: Sample S_1 without replacement to compute $\hat{\mathbf{v}}_k = \hat{\nabla} f_{S_1}(\mathbf{x}_k)$
4: **else**
5: Sample S_2 with replacement to compute

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

6: **end if**
7: $\mathbf{u}_k = \arg\max_{\mathbf{u} \in \Omega} \langle \mathbf{u}, -\hat{\mathbf{v}}_k \rangle$
8: $\mathbf{d}_k = \mathbf{u}_k - \mathbf{x}_k$
9: $\mathbf{x}_{k+1} = \mathbf{x}_k + \gamma \mathbf{d}_k$
10: **end for**

Output: Randomly choose x_α from x_k and return it

The main difference with the Algorithm 1 (ZSCG) is on how the gradient is estimated. Here is used a different version of the KWSA method, described before. 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. In the algorithm the gradient is estimated every q iterations applying the equation:

$$\hat{\nabla} f_{S_1}(\mathbf{x}_k) = \sum_{j=1}^{d} \frac{f_{S_1}(\mathbf{x}_k+\mu_j \mathbf{e}_j) - f_{S_1}(\mathbf{x}_k-\mu_j \mathbf{e}_j)}{2\mu_j} \mathbf{e}_j \tag{5}$$

While the other iterations we used the following formula:

$$\hat{\mathbf{v}}_k = \frac{1}{|S_2|} \sum_{i \in S_2} \left[\hat{\nabla} f_i(\mathbf{x}_k) - \hat{\nabla} f_i(\mathbf{x}_{k-1}) + \hat{\mathbf{v}}_{k-1} \right] \tag{6}$$

Where S_1 and S_2 represent randomly selected samples. The equation 6 is a variance reducing technique used to estimate the gradient. This estimator is used to reduce the variance due to the fact that we chose a random sample to estimate the gradient.

Theorem 3.2 *Under Assumption seen in 3.1, 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:*

$$\mathbb{E}\left[\mathcal{G}(\mathrm{x}_\alpha)\right] \leq \frac{D\left(F(\mathrm{x}_0) - F(\mathrm{x}_*) + 11L\right)}{\sqrt{K}} \tag{7}$$

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

.

Corollary 3.2.1 *With the same setting as Theorem 3.2, the number of call required to solve* ϵ*-stationary point are:*

$$FQO : O\left(\frac{n^{1/2}d}{\epsilon^2}\right) \quad LO : O\left(\frac{1}{\epsilon^2}\right)$$

.

3.5. Faster Zeroth-Order Conditional Gradient Sliding

In this section we reported the pseudo code for the Faster Zeroth-Order Conditional Gradient Sliding Algorithm. This algorithm is an improved version of the previous one (FZFW). The main difference is on how x_k is updated. The FZCGS used a conditional gradient sliding algorithm (Algorithm 4), that return the new x_k when the Frank-Wolfe gap is smaller than a certain threshold, η. The exact definition on how the conditional gradient sliding algorithm is implemented and how it works can be found in (Liu et al. (2018); Qu et al. (2018)).

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

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

1: **for** k = 0,.., K - 1 **do**
2: **if** mod(k,q) = 0 **then**
3: Sample S_1 without replacement to compute $\hat{\mathbf{v}}_k = \hat{\nabla} f_{S_1}(\mathbf{x}_k)$
4: **else**
5: Sample S_2 with replacement to compute

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

6: **end if**
7: $\mathbf{x}_{k+1} = condg(\hat{\mathbf{v}}_k, \mathbf{x}_k, \gamma_k, \eta_k)$
8: **end for**

Output: Randomly choose x_α from x_k and return it

Algorithm 4 $\mathbf{u}^+ = condg(\mathbf{g}, \mathbf{u}, \gamma, \eta)$ (Qu et al., 2017)

1: $\mathbf{u}_1 = \mathbf{u}, t = 1$
2: $\mathbf{v}_t$ be an optimal solution for

$$V_{g,u,\gamma}(\mathbf{u}_t) = \max_{\mathbf{x} \in \Omega} \langle \mathbf{g} + \frac{1}{\gamma}(\mathbf{u}_t - \mathbf{u}), \mathbf{u}_t - \mathbf{x} \rangle$$

3: **if** $V_{g,u,\gamma}(\mathbf{u}_t) < \eta$ **then**
4: return $\mathbf{u}^+ = \mathbf{u}_t$
5: **end if**
6: Set $\mathbf{u}_{t+1} = (1 - \alpha_t)\mathbf{u}_t + \alpha_t \mathbf{v}_t$ where

$$\alpha_t = \min\{1, \frac{\langle \frac{(\mathbf{u}-\mathbf{u_t})}{\gamma} - \mathbf{g}, \mathbf{v_t} - \mathbf{u_t} \rangle}{\frac{\|\mathbf{v_t}-\mathbf{u_t}\|^2}{\gamma}}\}$$

7: t = t + 1, go to step 2

Compared with the previous algorithm, the FZCGS has better FQO since it employs the conditional gradient sliding algorithm to accelerate the convergence speed.

Theorem 3.3 *Under Assumption seen in 3.1, 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 2 has the following convergence rate:*

$$\mathbb{E}\left[\left\|\mathcal{G}\left(\mathrm{x}_\alpha, \nabla F\left(\mathrm{x}_\alpha\right), \gamma\right)\right\|^2\right] \leq \leq \frac{\left(3\left(F\left(\mathrm{x}_0\right) - F\left(\mathrm{x}_*\right) + 1\right) + 7L\right) 6L}{K} \quad (8)$$

.

Corollary 3.3.1 . *With the same setting as Theorem 3.3, the number of call required to solve* ϵ*-stationary point are:*

$$FQO : O\left(\frac{n^{1/2} d}{\epsilon}\right) \quad LO : O\left(\frac{1}{\epsilon^2}\right)$$

4. Experiments

The aim of this project is to test the previous algorithms for the generation of adversarial examples in the case of a Black-box attack. In particular, given a black-box DNNs $f : \mathbb{R}^d \to \mathbb{R}^c$ and a dataset $\left\{(\mathbf{x}_i, y_i) : \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{0, 1, \cdots, c\}\right\}_{i=1}^n$, the task is to find the universal adversarial perturbation $\delta \in \mathbb{R}^d$ for sample $\mathbf{x}_i$ such that the DNN model makes an incorrect prediction $\hat{y}_i \neq y_i$. To achieve this results we want to minimize the difference, of the output in the last layer, between the correct prediction and the second most likely prediction according to the networks. So, in the end, we end up optimizing the following minimization problem:

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

where $f(\mathbf{x}) = [f_1(\mathbf{x}), f_1(\mathbf{x}), \cdots, f_c(\mathbf{x})]$ denotes the output of the last layer before conducting the softmax operation. For our experiments we tested our model with s = 0.1 and s = 0.3. Of course we achieved better results with s=0.3, since the perturbation is stronger, but the problem is that if s is too high, it becomes obvious that the images are attached and even a human eye cannot classify them correctly. In the code we have defined s as the maximum difference between a perturbed pixel and the original one.

4.1. Dataset

We tested our algorithm with the MNIST dataset, containing handwritten digits. This dataset has a training set of 60,000 examples and a test set of 10,000 examples. For our experiments we didn't attack all the images in the test set (too time-consuming and outside the scope of the project) but we restricted our attack in a sample of 100 images. The dataset contains grey images, size 28x28 pixels, of handwritten digits from 0 to 9. We divided each pixel by 255, in order to restrict the pixels in the range [0,1], moreover we flattened each image so that instead of working with matrices of size 28x28, we worked with vectors of length 784.

4.2. Model

For this project we used the model proposed by Liu et al. (2018). The architecture of the DNN is reported in the table below. Since we could not find pre-trained weights for this network, we decided to train it once and import the weights every time.

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 trained network reach a test accuracy of 99.40%, which is a satisfactory result. But since the purpose of the project is to perform adversarial attacks we decided to eliminate, from the test set, all the wrongly predicted examples. This is done to make sure that all examples of misclassification are due to image attacks. For our exper-

iments we considered two main cases. The case where we consider all labels (so we extract 10 images of each label from the test dataset) and the case where we attach only one label, in our case the number 9.

4.3. Hyperparameters

Since not all hyperparameters defined in the papers were practically usable for the analysis, as they could take too long in some cases, we decided to empirically find the best hyperparameters for this problem. For this purpose, we performed a grid search with different combinations of variables. To speed up the process, we used fewer images (50) and fewer epochs (20), than in the final model. Moreover we tuned the hyperparameters in the case we are considering all the labels and only, due to time limitation, for the label nine, even if every label should have their own hyperparameters. Below we reported just one table because the hyperparameters we found for label nine are the same for all the labels.

Hyperparameters					
Algorithm	α	v	γ	μ	η
ZSCG	0.1	10^{-5}	-	-	-
FZFW	-	-	0.1	0.1	-
FZCGS	-	-	0.1	0.1	0.01

4.4. Tests

Before running the final model, we decided to run several tests to better understand how the algorithms work by changing their parameters. In order to avoid redundancies, we only reported the results obtained by attaching only the nine labels; in the code are also shows the results if all labels are considered.

4.4.1 Test time ZSCG

Here we tested some parameters regarding the number of images given as input (n) and the number of random direction (q or m_k) chosen every time we want to estimate the gradient. In this analysis two aspect were evaluated:

- The average time to complete one epoch, obtained by dividing the total time by the number of epochs.

- The loss function value, to have an idea about the goodness of the method.

The aim of the test was to find the best trade off between *Loss* and Time, these were the parameters tested and the results are shown in the table below.

	number of samples			
q	32	64	128	256
1	23.4466	23.1779	23.0070	21.9089
4	23.0355	22.9021	22.6687	21.3321
10	22.6541	22.3504	21.9131	20.7494
30	21.8477	21.1695	20.6888	18.8878

As might be expected, increasing the number of random directions also decreases the value of the loss function, a decrease can also be achieved by increasing the size of the set used as input. The decreasing values of the loss function are closely correlated with the increasing time to complete an epoch: if more accurate results want to be obtained, more computational time is required. Figure 1 shows it graphically.

After seeing this figure, may be think that increasing the number of random direction q does not affect so much the time to complete an epoch but affect only the decreasing of the loss function. Next figure (Figure2) will clarify why we choose $q = 30$ for the final model: after that value the loss function keep decreasing but the computational time start rising faster.

4.4.2 FZFW random directions vs number of samples

In Algorithm 2 (FZFW) and Algorithm 3 (FZCGS) there is a very important parameter, q.

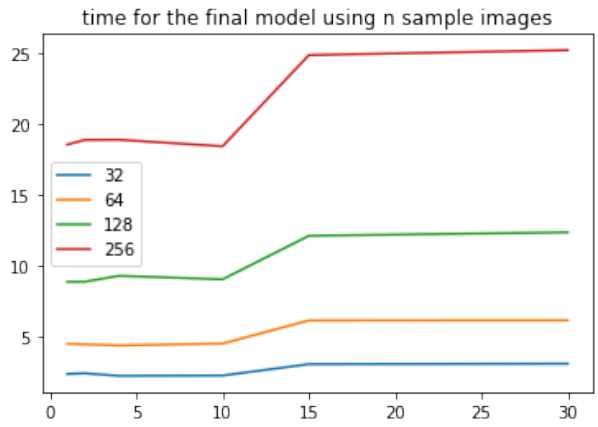

Figure 1. Average time to complete an epoch and low number of random direction used in the gradient approximation, for different sizes of input sets.

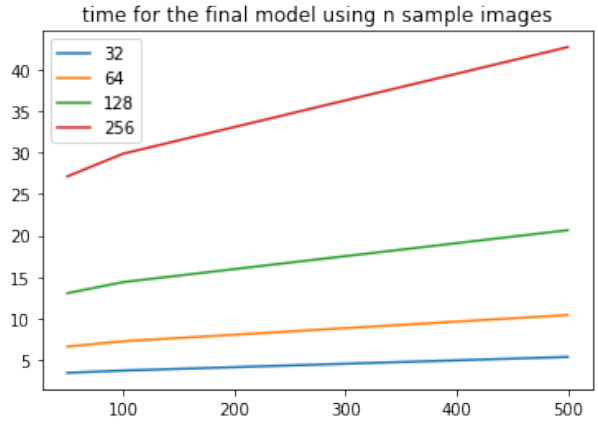

Figure 2. Average time to complete an epoch and high number of random direction used in the gradient approximation, for different sizes of input sets

This parameter regulates how many times the algorithms has to compute the approximation of the gradients for all the images. From Theorem 3.2 this parameter is set to $\sqrt{n}$. We decided to test the impact of this parameter on the performance of the algorithms, in particular for FZFW. Below we reported the time and the Loss for increasing q and different n. In the tables the time is referred to compute 20 epochs.

Of course, if the value of q becomes smaller, we obtain better results in terms of the loss function. But we can see from the tables that there is not much difference for the loss function, instead the

$n = 36$		
q	t (s)	Loss
1	147	2.57
2	81	2.83
5	61	3.08

$n = 64$		
q	t (s)	Loss
1	258	2.76
2	137	3.68
5	84	3.38

$n = 144$		
q	t (s)	Loss
1	578	3.12
2	296	3.16
5	148	3.96

time required decreases a lot if we increase the parameter. This implies that, if we want to have a good trade-off between time and performance, the value $q = 1$ and even $q = 2$ are not the best ones. Next, in the analysis of the final model, we kept the value of q suggested by the Theroem, $\sqrt{n}$.

4.4.3 Clip vs No Clip

A constrain that we can applied to our algorithms is to add a Clip function at each epochs. The scope of this function is to keep the pixels of the perturbed images between the range [0,1]. This constrain will of course reduce the performance of the attacks but could be useful in real case scenario since, as counter attack, it would be sufficient to constrain the network to classify only the images that has pixels in a specific range (in this case [0,1]). Figure 4.4.3 shows the loss function of the three algorithms with and without the use of the clip functions.

It can be seen from the graph that, when using clip function, the loss does not reach the same low values as when it is not applied. Moreover the clip function leads to worse results (reported on the code). This is quite obvious since applying the clip function imply that we are imposing more constrain on the attacks. Another main difference between the application and non-application of the clip function concerns the perturbation that the algorithms learn. In fact if the clip is used the perturbation strongly depends on the image

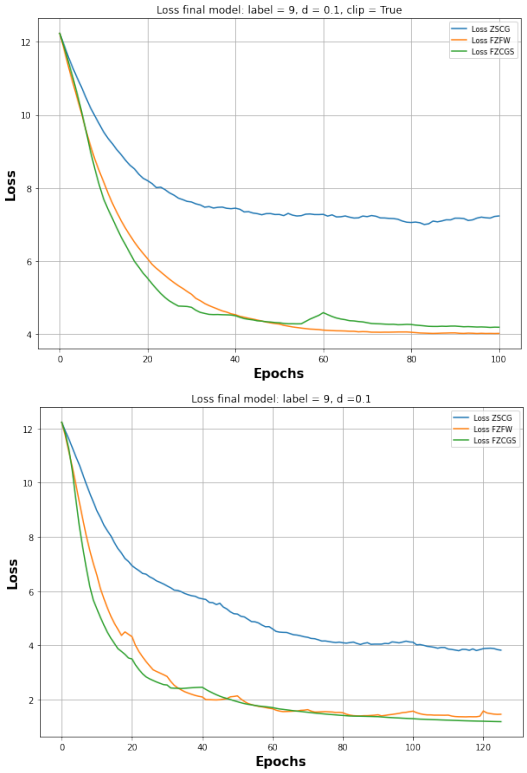

Figure 3. Loss with Clip vs loss without Clip

on which the attack is performed, as we can see below. Instead, if the clip is not applied, the algorithm learns a universal perturbation, for all images it attacks. This is not a surprising results, in fact at each iteration the clip functions act differently for each images, according to the initial values of the pixel. In the table below are reported the results, where ASR is the Attack Success Rate (basically the percentage of incorrectly classifying images by the original DNN).

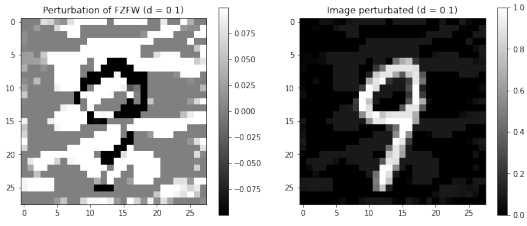

Figure 4. Perturbation using FZFW and clip = True

4.4.4 Test unseen images

The algorithms, if clip is not applied, learn an universal perturbation, that can be also be applied to completely new images. We decided to test the performances of the algorithms also to unseen images, to see if they still work well. We extracted 500 non-attacked images from the test-set to test the algorithms. We applied to them three difference perturbation found by the algorithms.

attacked images		unseen images	
Algorithm	$ASR(\%)$	Algorithm	$ASR(\%)$
ZSCG	22	ZSCG	15.8
FZFW	44	FZFW	31.8
FZCGS	47	FZCGS	35.4

We can notice that there is not a huge drop in the results in this case. This means that the perturbations learn for few example can be extended, in this case, to completely unseen images. Moreover the perturbation are learnt in a sample of 100 images, of course if we increase this number we can reach better result also against unknown images (but this is not the focus of the project).

4.5. Final model

We finally decided to test the ability of generating adversarial attack of our algorithms. To do so we run our algorithms for 125 epochs with the value of the hyperparameters that we found before. We tested our algorithms to attack 100 images both in the case they are taken from the same label and from different labels. To compare the results we checked the speed of convergence of the loss function and the attack success rate (ASR).

4.5.1 Single label d = 0.1

We first tested our algorithms considering a single label, in this case 9 with a constrain in the distortion equal to 0.1 and without applying the clipping procedure. As we can see below, from the Convergence of the Loss function 5, our empirical results are aligned with the theory. In fact the

Algorithm FZFW and FZCGS converges much faster than ZSCG algorithm.

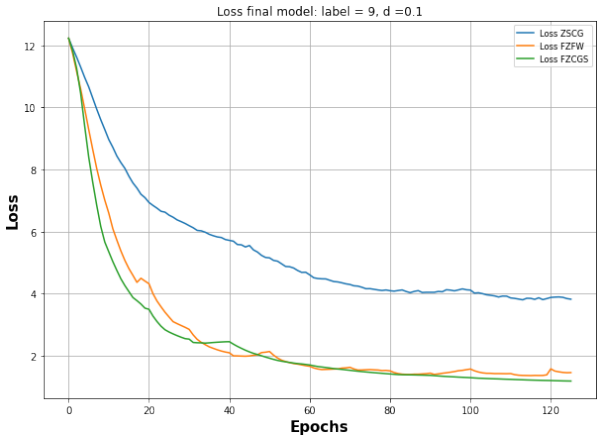

Figure 5. Comparison single label and d = 0.1

Here we reported the universal perturbations discovered by the algorithms and we applied them to the same images to see the different attacks generated by the three algorithms. Finally, to have another comparison between the algorithms, we predicted the labelling of the perturbed images using the original DNNs. As we can see from the table below the algorithm that achieve the best result is the FZCGS, being able to make the original network go wrong by 47%.

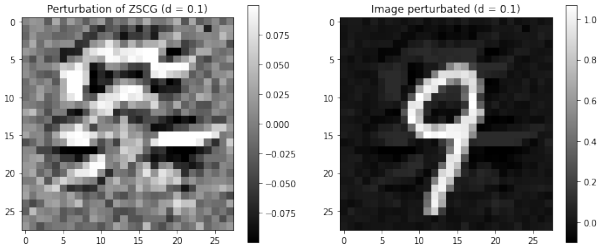

Figure 6. Perturbation using ZSCG and d = 0.1

Algorithm	ASR
ZSCG	22%
FZFW	44%
FZCGS	47%

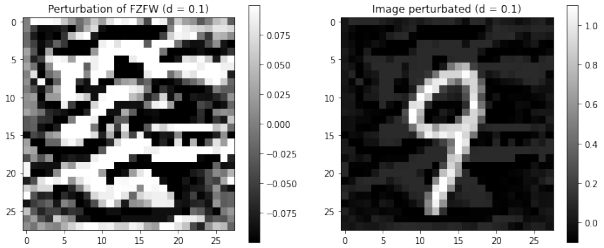

Figure 7. Perturbation using FZFW and d = 0.1

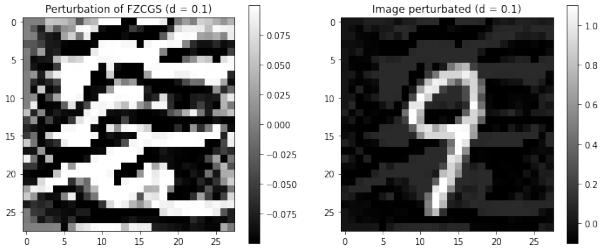

Figure 8. Perturbation using FZCGS and d = 0.1

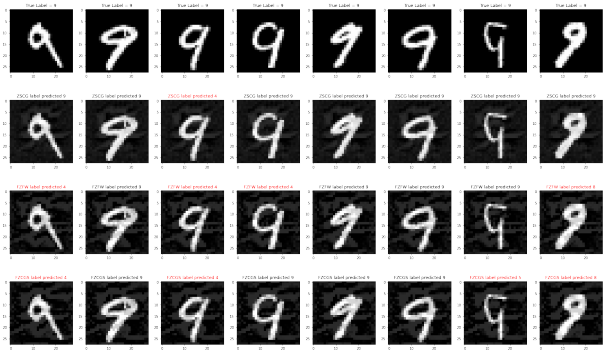

Figure 9. Generated adversarial examples from the same class (d = 0.1).

4.5.2 Single label d = 0.3

Subsequently we have repeated the same tests but we increased the constrain in the distortion to 0.3. Obviously this imply that the algorithms will reach better performance, since they are able to deeper perturb the images. We can see this also from the convergence of the Loss function, since it goes almost to zero in few epochs. In this case we have strange behaviour for the FZFW at higher epochs, but this is not a problem, as it does not affect performance. With a distortion of 0.3 all the algorithms performs very good attacks, causing the original network to misclassify more than 90% of the total images. The main problem in this

case is that the distortion is so high that is obvious that the images have been manipulated, as we can see from the perturbations reported below.

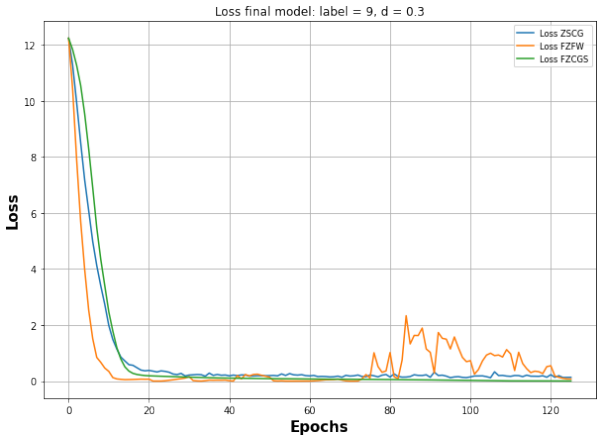

Figure 10. Comparison single label and d = 0.3

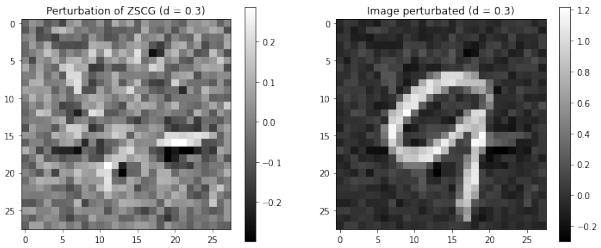

Figure 11. Perturbation using ZSCG and d = 0.3

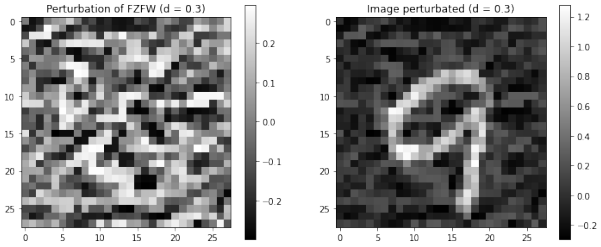

Figure 12. Perturbation using FZFW and d = 0.3

Algorithm	ASR
ZSCG	91%
FZFW	91%
FZCGS	94%

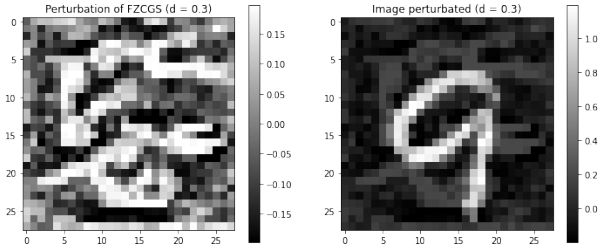

Figure 13. Perturbation using FZCGS and d = 0.3

4.5.3 All labels d = 0.1

Finally we tested our algorithms performance considering the attack to a dataset containing all labels. The first thing we can notice from the convergence of the Loss function 14, is that the algorithms are not able to reach the same level of performance. In fact the Loss are much higher comparing with the case of a single label. This is also reflected in the results, in fact the attack performed by all the three algorithms reaches very poor result, against the original DNNs. From the table below we can notice that the best algorithm is the FZCGS, but is able to force the original network to fail only 11% of the total images, very low if it is compared with the attack to a single label (47%). We think that this may due for multiple reasons: we simply train our methods for to few epochs or that 100 images, of different labels, are not enough to learn an efficient universal perturbations. Another idea is, instead of found an universal perturbation for all labels, to divide the dataset in 10 different subset where all the labels are well separated. In this case the object of the algorithms will be to find ten different universal perturbation, one for each label, and finally attack the original dataset depending on the label. This of course will be more time consuming, since it needs to find more perturbation, but we believe that it can improve the results a lot.

Algorithm	ASR
ZSCG	2%
FZFW	9%
FZCGS	11%

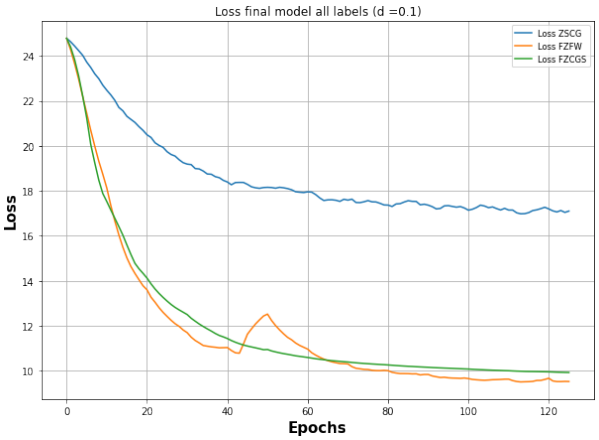

Figure 14. Comparison all labels and d = 0.1

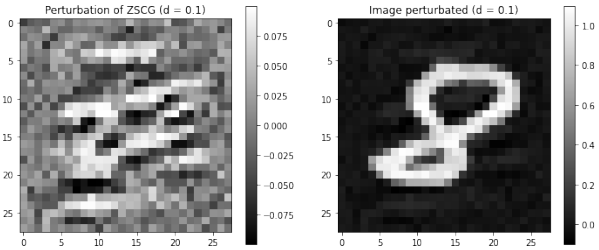

Figure 15. Perturbation using ZSCG and d = 0.1

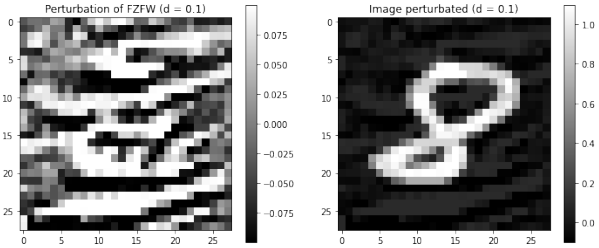

Figure 16. Perturbation using FZFW and d = 0.1

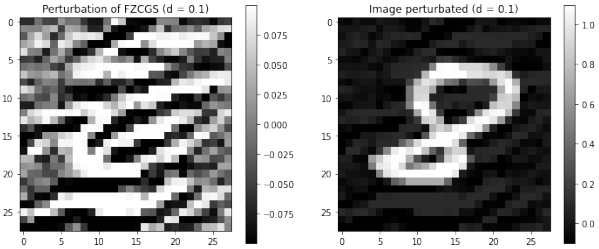

Figure 17. Perturbation using FZCGS and d = 0.1

5. Conclusion

In this project we tested the convergence and the performance of three different zeroth-order Frank-Wolfe methods on a Black-box DNNs.

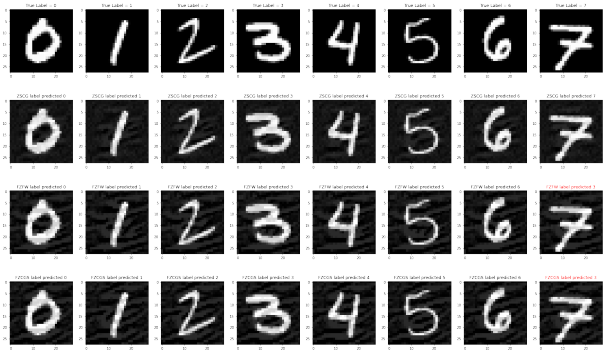

Figure 18. Generated adversarial examples from different classes (d = 0.1).

The results we found are perfectly aligned with the theory results presented in Gao and Huang (2020). In fact both FZFW and FZCGS converge much faster respect ZSCG. This suggest that the coordinate-wise gradient estimator works better than the averaged random estimator. Moreover the last algorithm (FZCGS) is the one that is able to force the original network to misclassify more images, even if it reach a loss function similar to the FZFW. This confirms that applying acceleration technique for the gradients helps to achieve better results. Future work, if more powerful resources are available, could consist in extending the proposed methods to more complex datasets and attacking more complex DNNs.

References

Krishnakumar Balasubramanian and Saeed Ghadimi. Zeroth-order (non)-convex stochastic optimization via conditional gradient and gradient updates. 31, 2018. URL https://proceedings.neurips.cc/paper/2018/file/36d7534290610d9b7e9abed244dd2f28-Paper.pdf.

Jinghui Chen, Dongruo Zhou, Jinfeng Yi, and Quanquan Gu. A frank-wolfe framework for efficient and effective adversarial attacks. 2018.

Hongchang Gao and Heng Huang. Can stochastic zeroth-order frank-Wolfe method converge faster for non-convex problems? 119:3377–3386, 13–18

Jul 2020. URL https://proceedings.mlr.press/v119/gao20b.html.

Xiangru Lian, Huan Zhang, Cho-Jui Hsieh, Yijun Huang, and Ji Liu. A comprehensive linear speedup analysis for asynchronous stochastic parallel optimization from zeroth-order to first-order. 2016.

Sijia Liu, Bhavya Kailkhura, Pin-Yu Chen, Paishun Ting, Shiyu Chang, and Lisa Amini. Zeroth-order stochastic variance reduction for nonconvex optimization. 2018.

Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2): 527–566, 2017.

Chao Qu, Yan Li, and Huan Xu. Non-convex conditional gradient sliding. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4208–4217. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/qu18a.html.