Zeroth-Order Frank-Wolfe Variants for Robust Adversarial Attacks

Riccardo Minzoni, Alfredo Petrella, Marco Francesco Sommaruga[†]

***Abstract*—In a world where Machine Learning is extensively used in an exponentially growing number of fields, knowing about Black-Box Attacks is key to increase awareness about models' security issues. Image classification, among many others, poses different doubts about the reliability of these systems when performing extremely complex tasks, such as self-driving cars and intelligent video surveillance.**

In this report, three Zeroth-Order Stochastic Frank-Wolfe methods are analyzed and implemented in order to find a *small* perturbation that, applied to an image from the famous MNIST dataset, forces the LeNet5 model to return a wrong label for it, without precluding a human observer to recognise the image's original class. The resulting outputs are then commented and compared.

***Index Terms*—Optimization, Neural Networks, Adversarial Attacks, Black-Box, Zeroth-Order Frank-Wolfe.**

I. Introduction

It is known that there exist several deep neural network algorithms able to perform image classification with a high level of accuracy. One of the challenges nowadays is to attack these highly performing algorithms in order to let them fail in the prediction of the images' class and discover their fragilities.

In this article, the objective is to study the state-of-the-art optimization techniques developed to perform an Adversarial Attack to an image dataset.

The algorithms available to perform such attacks can be distinguished by numerous characteristics.

- **Targeted VS Untargeted**
  The first important difference can be found in the type of attack: in untargeted attacks the aim is simply to cause the misclassification of an instance into a random class, while, in targeted attacks, the outcome of the misclassification is intentionally shifted to an established class.
  Beyond this explicit diversity, the main implication is that, in targeted algorithms, the key step is to minimize/maximize an objective function (loss function) with respect to a determined class, whereas in untargeted algorithms it is enough to minimize/maximize it, without considering an explicit dependence by a target label. In this work we will focus on the untargeted approach.

[†]Department of Mathematics, University of Padova, emails:
riccardo.minzoni@studenti.unipd.it
alfredo.petrella@studenti.unipd.it
marcofrancesco.sommaruga@studenti.unipd.it

- **Unconstrained VS Constrained**
  Another distinction is given by unconstrained and constrained algorithms. Both aim to optimize an objective function: the former has no limits in terms of bounds within which to find the solution, the latter is characterized by boundaries into which the optimized solution has to be searched.
  Since the foundation of Adversarial Attacks is to extract a result that is highly similar to the original image (without human-detectable distortions), the principal algorithms we may consider are those referred to as constrained: remaining inside a set C permits to reach a solution that must be close to the *non-attacked* instance.

- **Zeroth, First, Second order approximation of the gradient**
  A generic constrained optimization algorithm is posed as:
  $$\min_{\mathbf{x}\in C} \quad f(\boldsymbol{x}) \tag{1}$$
  where $C \subset \mathbb{R}^d$ is the associated constraint and $f(\boldsymbol{x})$ is the objective function to be minimized. In the context of this study, the optimization problem involves the minimization of the loss function inside a convex boundary set C.
  Most of the optimization problems assume a gradient based approach. Often it is not possible to estimate explicitly the gradient and a stochastic approach is needed. In the context of Black-Box Adversarial Attacks the explicit form of the loss function is not known, thus a stochastic method must be employed.
  A general stochastic optimization problem can be stated as:
  $$\min_{\mathbf{x}\in C} \quad f(\boldsymbol{x}) = \min_{\mathbf{x}\in C} \quad \mathbb{E}_{y\sim P}[F(\boldsymbol{x},\boldsymbol{y})] \tag{2}$$
  where
  $$\mathbb{E}_{y\sim P}[F(\boldsymbol{x},\boldsymbol{y})] \tag{3}$$
  is the expected loss function computed on the model $\mathbf{x}$ and the data $\mathbf{y}$ obtained from a distribution P.
  The stochasticity comes by the fact that the loss function $f(\mathbf{x})$ can not be queried and just its expectation with respect to the data distribution can be computed.
  Once obtained an estimate of the loss function, the next step in the optimization algorithms considered in this work is the estimation of the gradient.
  There are three main approaches to estimate the gradient:

  - *Zeroth-Order approximation of the gradient*
    This approach exploits an oracle which yields an evaluation of the loss function at a queried point.
  - *First-Order approximation of the gradient*
    In this case the oracle is able to yield a first order approximation of the loss function.
  - *Second-Order approximation of the gradient*
    It is based on the notion of the convexity of the loss function. Since the time required by the second order oracle to compute the hessian matrix, i.e. the second order derivative of the loss function, is hugely higher than that of the oracles of lower orders, this third approach is the least widespread.

- **White-Box, Black-Box, No-Box adversarial attacks**
  When attacking deep learning models, there are two different approaches that can be distinguished. White-Box attack means the entire network (input, layers, weights, output) is known and the attacker has full access to it, hence the back propagation can be computed. The attack could be seen as a gradient ascent in which the aim is to maximize the loss of the model.
  A more real situation is described by the Black-box attack. In this framework, the attacker has only access to the input and the probabilities of the output. Back-propagation cannot be computed since the nature of the network is unknown, so there is a need of zeroth-order methods that exploit estimates of the gradients.
  In an extreme case of the previous setting, called No-Box attack, only images and class labels are given and the classifier cannot be queried. This is a very restricted situation and usually it is not experienced in optimization problems.

- **Frank-Wolfe Optimization Algorithms**
  As mentioned above, once obtained or estimated the objective function, the next step is to find an optimal point that can be a minimal or a maximal one accordingly to the scope of the problem. There is a copious number of techniques studied, and in this work we will focus on three variants of the Frank-Wolfe algorithm (FW).
  The vanilla FW algorithm starts with a first-order Taylor approximation of the objective function $f(\boldsymbol{x})$ and exploits a first-order oracle to find the optimal point.
  The algorithms studied in this paper, instead, involve zeroth-order oracles.

- **Convex VS Non-convex**
  It is also important to distinguish between convex and non-convex optimization problems, named after the eventual convexity of their respective loss function. In fact, in the convex case, searching for a critical point is equivalent to searching for an optimal point, often resulting in better convergence rates and in computational resources saving.

Summarizing, we implemented and tested three zeroth-order constrained algorithms for non-convex optimization:

- Decentralized Variance-Reduced Zeroth-Order Frank-Wolfe
- Distributed Zeroth-Order Frank-Wolfe
- Zeroth-Order Frank-Wolfe with Inexact Update

In the first two cases, the goal was finding a (single) universal perturbation thanks to which every image would result misclassified with an a priori fixed probability $1 - \delta$, while in the third one the perturbation was strictly dependent on every single image.

All the algorithms have been written in Python and run on a MacBook Pro with processor 1.4 GHz Intel-Core i5 Quad-Core. The code is available in the exam GitHub repository of our group.

II. Related Work

There are two reasons why zeroth-order oracles FW algorithms have been explored in recent years.

First, when dealing with deep learning models and huge amount of data, because of the extremely high computational cost, it is unfeasible to compute the exact gradient. Hence, algorithms such as Projected Gradient Descent cannot be used. Secondly, since an attacker can usually only retrieve the input and the output of a model, Black-Box adversarial attacks are more common than White-Box ones.

Balasubramanian and Ghadimi in [1] demonstrate theoretical convergence results about a stochastic variant of the vanilla FW algorithm, introducing an inexact update of the gradient, to further accelerate the convergence.

Sahu and Kar in [2] discuss both basic and variance reduced zeroth-order stochastic FW algorithms to solve broad classes of smooth constrained optimization problems in both decentralized and distributed setups.

In a decentralized setup, the devices, or workers, are connected to a master node to read, write, and exchange information, which is typical in data-center-type environments. Whereas, in distributed setups there is not a central coordinator and information is exchanged in a peer-to-peer manner, by means of local computation and message exchanges with neighboring devices, which is typical in IoT-type environments.

In the decentralized setup a variance reduction technique is employed, called Stochastic Path-Integrated Differential Estimator (SPIDER), that can be used to reduce query complexity. This approach was implemented by the authors of [3] with the aim of tracking many deterministic quantities of interest with significantly reduced computational cost.

The aim in [2] was to generate a universal perturbation instead of a single perturbation for each input image. This noise could fool the model in misclassifying almost all the images in the dataset.

Many experimental papers such as [4] and [5] exploit this idea of the universal perturbation pursuing, as methods able to

query oracles only once do, a trade-off between the accuracy of the misclassification and the computational effort.

In our project we replicate the algorithms implemented in [1] and [2] on MNIST dataset.

III. Dataset and Model

In order to test the algorithms, we have chosen to work with the famous MNIST dataset [6]. It contains 70'000 samples, each one consisting of an handwritten digit, size-normalized and centered in a 28x28x1 image, and its corresponding true label.

As for the target model, the choice fell on the LeNet5 model [7], proposed by the dataset's creator himself, Chief AI Scientist at Facebook & Silver Professor at the Courant Institute, New York University, Yann LeCun. A graphical representation of the model can be found in Fig. 1, and it essentially solves the classification task on our dataset with more than 99% accuracy on the test set.

The most natural choice as a loss function is the Categorical Cross Entropy computed between the true one-hot-encoded labels and the probability distributions given in output by the final Softmax layer of the model. For a dataset with N samples and nClasses classes, its expression is reported in Eq. 4:

$$CCE(\mathbf{y}, \mathbf{y}^{pred}) = \sum_{j=1}^{N} \sum_{k=1}^{\text{nClasses}} \mathbf{y}_{j,k} \log \mathbf{y}_{j,k}^{pred}. \tag{4}$$

These choices were driven both by the availability of a benchmark in the literature and by the fact that, given our hardware supply, it would have been unfeasible to perform an effective attack against a more complex model or dataset without reducing their dimensionality, and thus, again, loosing the previously mentioned benchmark.

In particular, before feeding the model with our data, we extracted 1'000 images from the canonical test set following two criteria:

- we kept the classes balanced, selecting 100 images for each of the 10 classes (digits from 0 to 9);
- we filtered out the images already misclassified by the model without the need of perturbating them, in order to avoid distorting the statistics about the attacks' performance.

The selected images' range of values have then been re-scaled to the interval [0,1], dividing every entry by 255, and every image has been padded with zeros at the edges to meet the input shape of 32x32x1 requested by the model.

Moreover, in order to make the experiments reproducible, we saved the selected model's weights and imported them at every model initialization.

IV. Algorithms

The schemes used in the algorithms follow an approximation of the gradient by sampling the objective function along different directions.

Let $\mathbf{x} \in \mathbb{R}^d$ be the variable of interest and $\mathbf{y}$ the output vector associated, distributed according a density $\mathcal{P}$.

The first method, called *Kiefer-Wolfowitz Stochastic Approximation* (KWSA), consists in approximating the gradient by sampling the objective function along the canonical basis vectors. The expression with which the estimate is found is:

$$\mathbf{g}(\mathbf{x}_t; \mathbf{y}) = \sum_{i=1}^{d} \frac{F(\mathbf{x}_t + c_t \mathbf{e}_i; \mathbf{y}) - F(\mathbf{x}_t; \mathbf{y})}{c_t} \mathbf{e}_i \tag{5}$$

where d is the dimensionality of the problem, $\mathbf{e}_i$ the canonical basis vector, and c_t is a time-decaying sequence.

Since this approach involves sampling the loss function along all the canonical basis vectors, it requires exactly d samples at each step to find an estimate of the gradient. A less time and memory consuming technique has been recently proposed in [8] and [9] called *Random Directions Stochastic Approximation* (RDSA). This approach permits to sample the objective function along random directions sampled from a Gaussian distribution. The estimation of the gradient can be written as:

$$\mathbf{g}(\mathbf{x}_t; \mathbf{y}) = \frac{F(\mathbf{x}_t + c_t \mathbf{z}_t; \mathbf{y}) - F(\mathbf{x}_t; \mathbf{y})}{c_t} \mathbf{z}_t \tag{6}$$

where $\mathbf{z}_{i,t} \in \mathbb{R}^d$ is a random vector sampled from a Gaussian distribution probability. Furthermore, if those terms are sampled and averaged in *m*-directions at each time, it is also possible to reduce the variance.

$$\mathbf{g}(\mathbf{x}_t; \mathbf{y}) = \frac{1}{m} \sum_{i=1}^{m} \frac{F(\mathbf{x}_t + c_t \mathbf{z}_{i,t}; \mathbf{y}) - F(\mathbf{x}_t; \mathbf{y})}{c_t} \mathbf{z}_{i,t} \tag{7}$$

When $c_t \to 0$, both the gradient estimations in (5) and (6) turns to be unbiased estimators of the real gradient of the function $\nabla f(\mathbf{x}_t)$.

These cited method are exploited to estimate the gradient of the loss function in the three different settings that are studied in this work.

All the algorithms considered are variants of the classic FW algorithm.

In these settings, since only a zeroth-order oracle is available, the true gradient of the objective function is replaced by its stochastic counterpart, which, in turn, is estimated through one of the techniques mentioned above. Since this introduced stochasticity could make the algorithm diverge, an averaging trick is exploited that lets the expectation of the difference between the averaged term and the real gradient going towards zero asymptotically:

$$\mathbf{d}_t = (1 - \rho_t)\mathbf{d}_{t-1} + \rho_t \mathbf{g}(\mathbf{x}_t; \mathbf{y}_t) \tag{8}$$

where $\mathbf{g}(\mathbf{x}_t;\mathbf{y}_t)$ is the estimation of the gradient at step t, $\mathbf{d}_0 = 0$ and ρ_t is a time-decaying sequence.

Once estimated the gradient, the optimal solution of the optimization problem is computed as the argmin or argmax (depending on the problem) of the scalar product between the gradient and the variable itself, bounded into the constraint:

$$\mathbf{v}_t = \arg\min_{\mathbf{v} \in C} \langle \mathbf{d}, \mathbf{v} \rangle \tag{9}$$

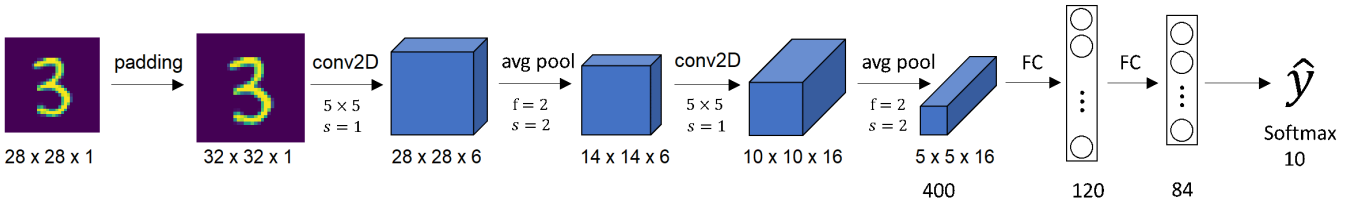

Fig. 1: Attacked LeNet 5 model structure.

After having found the optimal direction of reduction/maximization of the gradient, the variable $\mathbf{x}$ at step $t+1$ is updated with an average that considers all the previous steps performed:

$$\mathbf{x}_t = (1-\gamma_{t+1})\mathbf{x}_t + \gamma_{t+1}\mathbf{v}_t \tag{10}$$

where γ_t is a time-decaying sequence.

The main assumptions on the objective function to guarantee the convergence of the algorithms are the following.

- Consider a se C bounded with a finite diamater R.
- f^i are Lipschitz continuos with $(\mathbb{E}[\|\nabla_x(\mathbf{x};\cdot)\|^2])^{1/2} \leq L_1$ for all $\mathbf{x} \in C$.
- The function f is continuously differentiable and is L-smooth, that is, its gradient ∇f is L-Lipschitz continuous over the set C, that is, for all $x, y \in C$: $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|f(\mathbf{x}) - f(\mathbf{y})\|$
- $\mathbf{z}_t$ are drawn from a distribution μ such that $M(\mu) = \mathbb{E}[\|\mathbf{z}_t\|]^6$ is finite, and for any vector $\mathbf{g} \in \mathbb{R}^d$, there exists a function $s(d)$: $\mathbb{N} \to \mathbb{R}_+$ such that $\mathbb{E}[\|\langle \mathbf{g}, \mathbf{z}_t\rangle \mathbf{z}_t\|^2 \leq s(d)\|\mathbf{g}\|^2$
- The gradient estimates $\nabla F_i(\mathbf{x};\mathbf{y})$ of $\nabla f_i(\mathbf{x})$ are unbiased, and satisfy $(\mathbb{E}[\|\nabla F_i(\mathbf{x};\mathbf{y}) - \nabla f_i(\mathbf{x})\|^2] \leq \sigma^2$ for all i.

The algorithms analyzed are exposed in the following sections.

A. Variance-reduced Decentralized Zeroth-Order FW

The first algorithm we developed is the Variance-reduced Decentralized Zeroth-Order FW. Decentralized means that the computations are performed by M different nodes, called workers, coordinated by a central entity, called master node. Every worker is then expected to work on a decentralized part of the optimization problem in Eq. 2, over a separate subset of the whole dataset. Our problem can be rewritten, in this framework, as:

$$\begin{aligned} \min_{\mathbf{x}\in C} f(\mathbf{x}) &= \min_{\mathbf{x}\in C} \frac{1}{M}\sum_{i=1}^{M} f_i(\mathbf{x}) \\ &= \min_{\mathbf{x}\in C} \frac{1}{M}\sum_{i=1}^{M} \mathbb{E}_{\mathbf{y}\sim P_i}\left[F(\mathbf{x},\mathbf{y})\right], \end{aligned} \tag{11}$$

where f_i can be computed locally for each node, which maintains a copy of the optimizer $\mathbf{x}$.

To further illustrate stochasticity in the objective function, we assume that each function f_i is composed of n component functions, thus leading to rewrite Eq. 11 as:

$$\begin{aligned} \min_{\mathbf{x}\in C} f(\mathbf{x}) &= \min_{\mathbf{x}\in C} \frac{1}{M}\sum_{i=1}^{M} f_i(\mathbf{x}) \\ &= \min_{\mathbf{x}\in C} \sum_{i=1}^{M}\sum_{j=1}^{n} f_{i,j}(\mathbf{x}). \end{aligned} \tag{12}$$

To address this problem, the SPIDER variance reduction technique is generalized in [2], starting from the centralized version proposed in [3].

SPIDER is a tool for dynamic tracking which requires as input:

- an estimate of the initial value of the sequence we want to track,
- an unbiased estimator of the difference of two consecutive terms of the sequence,

and can be used to track the stochastic gradient, the loss function values and the zeroth-order gradient estimates. The main advantages are that it reduces queries' number and complexity, and it can be shown to reduce the variance.

Once fixed a period parameter q, the algorithm estimates the gradient using KWSA every q steps (i.e. every one period), which results in a more time consuming but more accurate estimate, while it approximates the gradient with RDSA in the other iterations, faster but less effective.

After each iteration of any type, the master node computes the next iterate using the average of the received outputs and broadcasts it among the workers.

A schematic representation of the full algorithm is given in Alg. 1.

More in detail, it is considered a starting point $\mathbf{x}_0 \in C$ where C is a convex set, a loss function $f(\mathbf{x})$ and two sample sizes S_1 and S_2. For each iteration t, starting from 0 and stopping in any case at `MaxIt`, if $\text{mod}(t,q) = 0$ a new period starts and KWSA is used to accurately approximate the gradient over $S_1' = S_1/Md$, where M is the number of workers and d is the dimensionality of the problem. Being an accurate estimation, the result replaces the previous gradient approximation.

If, instead, $\text{mod}(t,q) \neq 0$, RDSA on S_2 images is performed, where the unbiased estimator for the gradient is constructed thanks to a standard normal random sample for

Algorithm 1 Decentralized Variance-Reduced Zeroth-Order FW

Require: Input, Loss function *f(x)*, Convex Set *C*, period q, Sample Sizes S_1, S_2

Initialize: $\mathbf{x}_0 \in C$

for $t = 1, ..., \texttt{MaxIt}$ **do**

if mod(t,q)=0 **then**

Draw $S_1^{'} = S_1/Md$ samples for each dimension k at each worker i and compute its local gradient along each canonical basis vector $\mathbf{e}_k$:

$$\mathbf{e}_k^T \mathbf{g}_i(\mathbf{x}_t) = \frac{1}{n} \sum_{j=1}^{n} \frac{f_{i,j}(\mathbf{x}_t + \eta_1 \mathbf{e}_k) - f_{i,j}(\mathbf{x}_t)}{\eta_1}$$

Each worker updates $\mathbf{g}_{i,t} = \mathbf{g}_i(\mathbf{x}_t)$

else

Draw S_2 pairs of component functions and Gaussian random vectors $\{\mathbf{z}\}$ at each worker i and update

$$\mathbf{g}_i(\mathbf{x}_t) = \frac{1}{|S_2|} \sum_{j \in S_2} \frac{f_{i,j}(\mathbf{x}_t + \eta_2 \mathbf{z}) - f_{i,j}(\mathbf{x}_t)}{\eta_2} \mathbf{z} - \frac{f_{i,j}(\mathbf{x}_{t-1} + \eta_2 \mathbf{z}) - f_{i,j}(\mathbf{x}_{t-1})}{\eta_2} \mathbf{z}$$

Each worker updates $\mathbf{g}_{i,t} = \mathbf{g}_i(\mathbf{x}_t) + \mathbf{g}_{i,t-1}$

end if

Each worker pushes $\mathbf{g}_{i,t}$ to the master node.

Master node computes $\mathbf{v}_t = \arg\min_{s \in C} \langle \mathbf{s}, \mathbf{g}_t \rangle$

Master node computes $\mathbf{x}_{t+1} = (1 - \gamma_t)\mathbf{x}_t + \gamma_t \mathbf{v}_t$ and send it to all workers.

end for

Output: $\mathbf{x}_{\texttt{MaxIt}}$.

each image, used as a random perturbation direction, as suggested in Thm. 1.2 of [1]. At this point the gradient is updated taking into account the previous one in not order to lose the effectiveness of the KWSA step.

Note that, as we will also see in Section V, given the optimal parameters, each image will be drawn, on average, S_2/n times.

Once the gradient has been estimated, the new iterate is computed depending on the goal of the optimization process. This step exploits the *average trick* exposed in Eq. 10. After reaching an optimum or the maximum number of iterations MaxIt, an average of the updated $\mathbf{x}_{t+1}^i$ is returned.

It has been proven in [2] that the optimal parameters, in terms of duality gap, in the non-convex case are the ones reported below:

$$\begin{aligned} &S_1 = Mnd \implies S_1' = n, \\ &n_0 \in [1, Mn/6], \\ &S_2 = ((2d+9)Mn)/n_0, \\ &q = (n_0 Mn)/6, \\ &\gamma_t = \texttt{MaxIt}^{-3/4}, \\ &\eta_1 = (c_t)_{KWSA} = \frac{2}{d^{1/2}(t+8)^{1/3}}, \\ &\eta_2 = (c_t)_{RDSA} = \frac{2}{d^{3/2}(t+8)^{1/3}}. \end{aligned} \tag{13}$$

In this setting, a duality gap smaller than ϵ can be obtained with a number of queries which is $O(\min\{d\sqrt{n}/\epsilon^2, d/\epsilon^3\})$.

B. Distributed Zeroth-Order FW

The second optimization algorithm studied is developed in a distributed manner, meaning that the loss function is broken into M components, each one distributed to a node of a network:

$$\min_{\mathbf{x} \in C} f(\mathbf{x}) = \min_{\mathbf{x} \in C} \frac{1}{M} \sum_{i=1}^{M} f_i(\mathbf{x}) \tag{14}$$

The same is valid for the instances, that are, as well, broken into batches, each of which belonging to one of the nodes. Thus, each node will have information about just a partial instances set, but thanks to the architecture of the algorithm, it will be able to receive knowledge about the rest of the data by the other workers. This is the main characteristic that allows to find a universal optimal point.

The main feature of this approach is the presence of a network where nodes exchange information in a peer-to-peer manner. Particularly, it is considered a simple undirected graph $G = (V, E)$, with $V = [1, \ldots, M]$ and $E = \{(i, j)\}$ with $i, j \in V$. Each node has degree $d^n = |\Omega_n|$, where Ω_n denotes the neighborhood of node *n*. The graph is described by an adjacency matrix $\mathbf{A}$ with $\mathbf{A}_{ij} = 1$ if (i,j) $\in E$, $\mathbf{A}_{ij} = 0$ otherwise. Furthermore, the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$ is positive semidefinite, where $\mathbf{D} = diag(d_1 ... d_M)$.

The information exchanged between nodes regards both the updated value of the variables and of the gradient computed in each node at each step t. A matrix $\mathbf{W} = \mathbf{I}\text{-}\delta\mathbf{L}$ weights the amount of information coming from each neighbour of each considered node, in such a way that if two nodes are not connected, they don't exchange directly their information. It is important to select a graph that is reasonably connected, permitting to each node to receive as much information as possible from the others. We need to find a trade-off between a sparse graph and a connected graph: a sparse network would not permit the flow of information between nodes, while a fully connected graph would make the algorithm too slow since all the nodes would send information to all the others.

The distributed approach exploits the deterministic version of the Zeroth-Order FW to estimate the gradient of the objective function, namely KWSA.

The algorithm, described in Alg. 2 starts building the connected graph with the characteristics mentioned above. Once obtained the graph, it is considered a starting point $\mathbf{x}_1^i$ for each node *i* in the graph. Then, it starts looping until the desired target of optimization is reached or the maximum number of iterations expires. In each of these iterations t, there is a first step called *consensus* where nodes exchange in a weighted manner the information on $\mathbf{x}_t^i$ with all the connected workers.

While in the first step of the algorithm each node will receive details just about the batch of instances retained by its neighbours nodes, in the following steps, it will increasingly receive more notions also about the other nodes. This flow of data happens thanks to the architecture of the graph: indeed, each node will need just one iteration to receive information about its neighborhood, two iterations for some details from two hopes nodes and so on. The more the iterations, the more nodes will be aware about the entire graph condition.

Received these data, each worker is ready to compute a *private* estimation of the gradient $\mathbf{g}_t^i$, that, in a second moment, will be aggregated with those of the steps before and that of the other nodes. This step, called *Aggregating*, is of fundamental importance because, even if the batch of instances that each worker exploit to estimate the gradient is different, through this aggregation step all the gradient details are sent to each others, allowing to find, at the following steps, values always more similar to each others, until reaching convergence. These steps let, in our setting, finding a universal perturbation for the images.

Once estimated the gradient, it is necessary to update the variable moving towards the descent/ascent direction of the gradient, accordingly with the scope of the optimization problem. This step exploits the *average trick* exposed in Eq. 10.

Once expired the maximum number of iterations, it is finally returned an average of the updated $\mathbf{x}_{t+1}^i$, considering the mean of $\mathbf{x}_{t+1}$ belonging to each node.

The parameters cited in Alg. 2 are:

$$\begin{aligned} &M : \text{number of nodes} \\ &\gamma_t = t^{-\alpha}, \\ &\eta = (c_t)_{KWSA} = \frac{2}{d^{1/2}(t+8)^{1/3}}. \end{aligned} \tag{15}$$

with those parameters and considering a non-convex loss function, a duality gap of ϵ is obtained with a number of queries upper-bounded by $O(d/\epsilon^2)$.

C. Zeroth-Order FW with Inexact Update

The third algorithm implemented is a zeroth-order stochastic gradient method which applies the Conditional Gradient (CG) method for non-convex problems. The authors of [1] proposed a modified algorithm that allows to skip the gradient evaluation from time to time. The algorithm 4 is the zeroth-order conditional gradient method for inexactly solving the following quadratic program:

$$P_{\mathcal{X}}(x, g, \gamma) = \arg\min_{u \in \mathcal{X}} \Big\{ \langle g, u \rangle + \frac{\gamma}{2} \|u - x\|^2 \Big\}, \tag{16}$$

which is the standard sub-problem of stochastic first-order methods applied to a minimization problem, when g is an unbiased stochastic gradient of the objective function at x. In particular, it is used the gradient mapping, defined as:

$$GP_{\mathcal{X}}(x, g, \gamma) = \gamma(x - P_{\mathcal{X}}(x, g, \gamma)),$$

where $P_{\mathcal{X}}$ is the solution of (16). This quantity plays an analogues role of the gradient in constrained problems. This way of using CG methods can significantly reduce the total number of calls to the oracle.

Thm. 2.3 in [1] defines the parameters of the following algorithm, $\forall k \geq 1$:

$$\begin{aligned} &\nu = \sqrt{\frac{1}{2\texttt{MaxIt}(d+3)^3}}, \\ &\gamma_t = 2L, \\ &\mu_k = \frac{1}{4\texttt{MaxIt}}, \\ &m_k = 6(d+5)\texttt{MaxIt}, \end{aligned} \tag{17}$$

where d is the dimensionality of the problem, `MaxIt` is the number of iterations, and L is the Lipschitz constant. Hence, it is demonstrated that the total number of calls to the stochastic

Algorithm 2 Distributed Zeroth-Order FW

Input: Initial point $\mathbf{x}_1^i$ for $i = 1, ..., M$.
for $t = 1, ..., \texttt{MaxIt}$ **do**
Consensus: approximate the average iterate:

$$\bar{\mathbf{x}}_t^i \leftarrow \sum_{j=1}^{M} W_{ij} \mathbf{x}_t^j$$

Gradient Estimation: At each node *i*, employ KWSA using Eq. 5 to estimate gradient $\mathbf{g}_t^i$
Aggregating: approximate the average gradient:

$$\mathbf{G}_t^j = \bar{\mathbf{g}}_{t-1}^i + \mathbf{g}_t^i - \mathbf{g}_{t-1}^i$$
$$\bar{\mathbf{g}}_t^i \leftarrow \sum_{j=1}^{M} W_{ij} \mathbf{G}_t^j$$

Frank-Wolfe Step: update

$$\mathbf{v}_t^i = \arg\min_{\mathbf{v} \in C} \langle \bar{\mathbf{g}}_t^i, \mathbf{v} \rangle$$

$$\mathbf{x}_{t+1}^i = (1 - \gamma_t)\bar{\mathbf{x}}_t^i + \gamma_t \mathbf{v}_t^i$$

for all nodes $i \in [M]$ and $\gamma_t \in (0, 1]$ is a step size.
end for
Return: $\bar{\mathbf{x}}_{t+1}^i \forall i \in [M]$

oracle to find an ϵ-stationary point of the problem is bounded by $O(d/\epsilon^2)$.

Algorithm 3 Zeroth-Order FW with Inexact Update

Input: $x_0 \in \mathcal{X}$, smoothing parameter $\nu > 0$, positive integer sequence m_k, and sequences γ_k and μ_k and a probability distribution $P_R(\cdot)$ over $\{0, ..., \texttt{MaxIt} - 1\}$
for $k = 1, ..., N$ **do**
Generate $u_k = [u_{k,1}, ..., u_{k,m_k}]$, where $u_{k,j} \sim N(0, I_d)$, call the stochastic oracle m_k times, compute:

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

Set

$$x_k = ICG(x_{k-1}, \bar{G}_\nu^k, \gamma_k, \mu_k)$$

where $ICG(\cdot)$ is the output of Algorithm 4 with input $(x_{k-1}, \bar{G}_\nu^k, \gamma_k)$
end for
Output: Generate R according to $P_R(\cdot)$ and output x_R.

Algorithm 4 Inexact Conditional Gradient (ICG) method

Input: (x, g, γ, μ)
Set $\bar{y}_0 = x, t = 1$, and $k = 0$
while $k = 0$ **do**

$$y_t = \arg\min_{u \in \mathcal{X}} \{h_\gamma(u) := \langle g + \gamma(\bar{y}_{t-1} - x), u - \bar{y}_{t-1} \rangle\}$$

if $h_\gamma(y_t) \geq -\mu$ **then**
Set $k = 1$
else
$\bar{y}_t = \frac{t-1}{t+1} \bar{y}_{t-1} + \frac{2}{t+1} y_t$ and $t = t + 1$
end if
end while
Output: $\bar{y}_t$

V. Results

In this section there are analyzed experimental setups of the three algorithms, the parameters involved and their performance. We considered the MNIST dataset, sampling from its test set 1'000 images, 100 for each class. In the algorithms from [2], we decided to be consistent with the paper and assign $n = 100$ images with balanced classes to each of the $M = 10$ nodes.

The final goal of the first two algorithms is to find a *small* universal perturbation (in terms of the l_p norm) with the aim of fooling the considered model (LeNet5). In particular, such perturbation has to satisfies two constraints:

1) $\|\mathbf{v} - \mathbf{x}\|_p \leq \xi$
2) $\mathcal{P}_{x \sim \mu}(C_{\text{LN5}}(\mathbf{y}^{pred}) \neq C_{\text{LN5}}(\mathbf{y})) \geq 1 - \delta$

where ξ controls the magnitude of the perturbation vector $\mathbf{v} - \mathbf{x}$ and δ quantifies the desired fooling rate for all images sampled from the distribution μ.

A key difference in the work is that, in a decentralized framework, thanks to the existence of the master node, each worker has access, after each iteration, to the whole information, speaking about the dataset and the estimated gradients, while, in the distributed case, each worker can only directly access the information of its neighborhood, even if, in a connected enough graph, as the number of iterations grows, the information flows up to reaching convergence.

Naming C_{LN5} the LeNet5 classifier, the problem can be formalized as:

$$\begin{aligned} \mathbf{v} &= \arg\max_{\mathbf{x}' : \|\mathbf{x} - \mathbf{x}'\|_\infty \leq \xi} L(\mathbf{x}', \mathbf{y}) \\ &= \arg\max_{\mathbf{x}' : \|\mathbf{x} - \mathbf{x}'\|_\infty \leq \xi} CCE(\mathbf{y}, C_{\text{LN5}}(\mathbf{x}')) \end{aligned} \tag{18}$$

where CCE refers to the non-convex Categorical Cross Entropy loss function.

In particular, these two algorithms proceed iteratively to find the optimal point that sends the perturbed images $\mathbf{x}_i$ to the boundary of the regions within which they are classified with the associated labels. In order to obtain such a point within the constraint set C, it is necessary to move along the ascent direction of the gradient: the objective in here is indeed to maximize the global loss function.

In all the algorithms we have considered the l_∞ norm for projecting over the feasible set C.

In this framework, in order for the point $\mathbf{v}_t \in \mathbb{R}^d$ to satisfy the optimization step in Eq. 9, it holds:

$$\begin{aligned} \mathbf{v}_t &= \arg\min_{\mathbf{x} \in C} \langle \mathbf{g}, \mathbf{v} \rangle \\ &= \xi \ \text{sign}(\mathbf{g}) + \mathbf{x}_0 \end{aligned} \tag{19}$$

where $\mathbf{x}_0$ is the original dataset.

The adversarial attack successes in few iterations in both the versions of the algorithm. In particular, they are both able to let the model misclassify more than 70% of the images. Moreover, as mentioned above, a requirement is to implement a further stopping criterion, besides the canonical maximum number of iterations `MaxIt`: for the Decentralized setting we considered $\delta = 0.3$, stopping the algorithm as soon as it misclassifies more than 70% of the perturbed images, while for the Distributed one $\delta = 0.2$, since the last approach finds a noise able to misclassify a higher number of images (more than 80%) in few iterations.

With regard to the third algorithm, the Zeroth-Order FW with Inexact Update, the aim is to find a perturbation for each image, differently from the universal approach of the previous ones. Within such a framework, the optimal points found by the algorithm for each image allow to reach a misclassification rate of 61%.

We now give a detailed description of the parameters used and of the results obtained for each algorithm.

A. Variance-reduced Decentralized Zeroth-Order FW

Parameter	Value
M	10
n_0	5
n	100
d	1'024
MaxIt	5
q	26
xi	0.25
S1	1'024'000
S1'	1'000
S2	13'009
L norm	Inf
Loss	Non-convex
γ_t	0.299
$(c_t)_{KWSA}$	$\frac{2}{d^{\frac{1}{2}}(t+8)^{\frac{1}{3}}}$
$(c_t)_{RDSA}$	$\frac{2}{d^{\frac{3}{2}}(t+8)^{\frac{1}{3}}}$

TABLE 1: Parameters Zeroth-Order Decentralized FW algorithm.

Parameters in Tab. 1 have been selected considering the values suggested in [2]: the algorithm runs for only 5 iterations and misclassifies 734 images out of 1'000. It can be noted that both the stopping criteria are met contemporarily (the condition of reaching a misclassification rate of 1-δ is satisfied exactly at iteration `MaxIt`). It is important to highlight that the average trick in Eq. 10 has the term γ_t that explicitly depends on `MaxIt`: choosing a higher value for such parameter would decrease the weight that is given to the $(t+1)^{th}$ perturbed batch of images, letting the algorithm converge more slowly. Another peculiarity of this approach is that it alternates KWSA and RDSA estimations of the gradient. The first iteration, which exploits KWSA, since leading to the right direction of optimization, is able to misclassify an average of 15% of the images. Once obtained this optimal direction, RDSA is used for $q-1$ more steps and, despite its lower estimation accuracy, it takes advantage of the globally approximated gradient, successfully misclassifying some of the remaining images. On the other side, the advancing rate of the algorithm is decreasing, due to the following reasons:

- the stochasticity of the chosen optimization direction;
- the reduction of perturbation freedom because of the closeness to the noise constraint;
- the higher distance of the remaining images from the boundary of the corresponding labels' region.

In order to find the optimal perturbation, the algorithm has to query several times the oracle to obtain the value of the loss function; in particular, while performing the KWSA step, each image requires d queries to the oracle, whereas the RDSA an average of S_2/n.

Furthermore, as said above, since the KWSA is far more complex, in a decentralized framework, its average time per node, per iteration is ∼40 seconds, while the RDSA requires roughly 7 seconds (including all the other steps of the algorithm, with a neglectable impact on the total).

Looking at the resulting universal noise in Fig. 2, we can note a shape that resembles overlapped digits. This is pretty intuitive and due to the fact that the algorithm is capable of detecting the most significant and characterizing pixels of every image in the dataset.

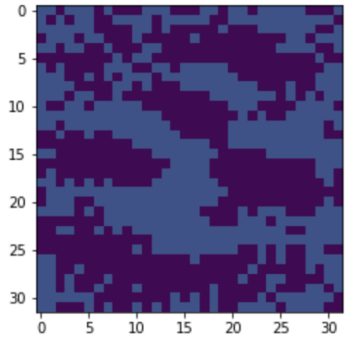

Fig. 2: Final Universal Perturbation Decentralized VR FW.

Overlapping the found noise to the original images, as we can see in Fig. 3, we obtain pictures in which the digits are clearly recognisable by the human eye but are capable of fooling the network with a misclassification rate higher than 70%.

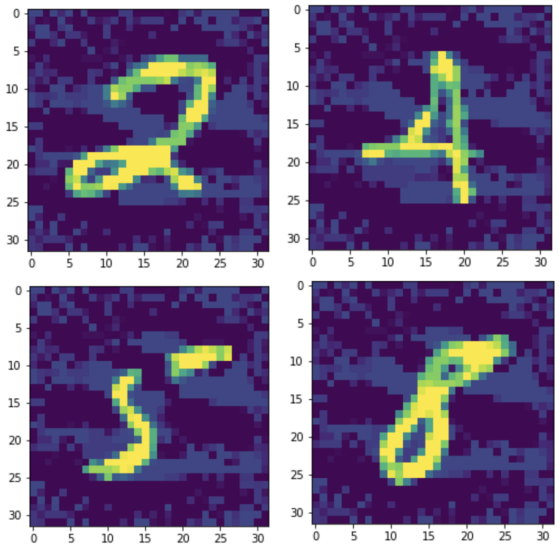

Fig. 3: Final Universal Perturbation Decentralized VR FW.

It is interesting to note that the proportion of misclassified digits is not uniform, as it can be seen in the Fig. 4 below that contains the misclassification rate per class obtained running our demo with our selected random seed.

In particular, no perturbed image containing an 8 can fool the model, probably due to the fact that, because of its shape, the 8 contains almost all the other digits, and so perturbating it up to misclassification would require a noise with a norm which is bigger than the fixed constraint. Different random

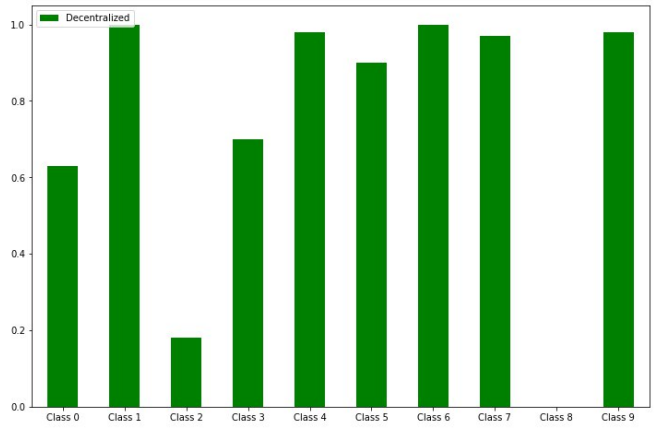

Fig. 4: % of misclassified per class in Decentralized FW.

seeds have actually been tested, and besides some small fluctuations, the results are comparable.

In order to visualize the convergence of the algorithm, the loss per iteration is shown in Fig. 5. It has been obtained as the sum of the losses computed on the whole dataset in each of the M nodes. As expected, the function is strictly increasing, as implied by the maximization step, while it is not possible to pronounce about the function convexity given the small number of iterations. Actually, we expected to see a steeper step after the first iteration due to the KWSA effectiveness, but it would probably show up averaging the loss behavior by period (every q steps); moreover, the KWSA is penalized by the fact that it is performing the very first iteration, where no previous estimate of the noise is available.

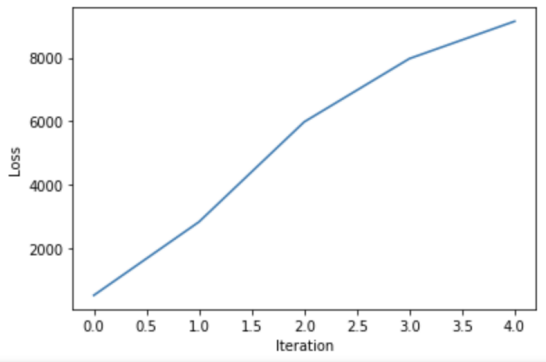

Fig. 5: Loss Function Decentralized FW vs Iterations

The algorithm's convergence is also validated by the misclassification rate trend (Fig. 6), which retraces the loss' one. It is reasonable to think that it would show a concave behavior given the noise constraint, but also in this case the small number of iterations does not allow further speculation.

B. Distributed Zeroth-Order FW

Tab. 2 contains the parameters reported in [2], which are the ones implemented in our code. Besides, after noting the importance of considering a reasonably well-connected graph, we decided to build ours reaching a connectivity parameter

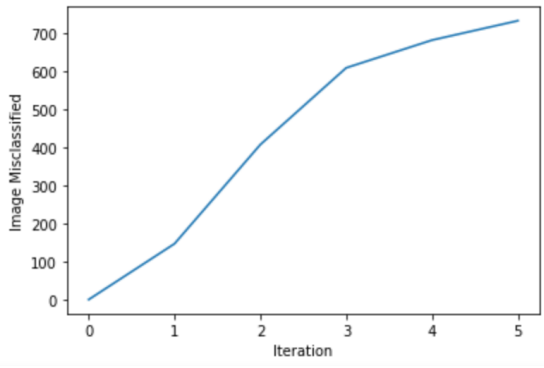

Fig. 6: Misclassification Rate Decentralized FW vs Iterations

Parameter	**Value**
M	10
n	100
d	1024
MaxIt	5
xi	0.25
alpha	0.5
L norm	Inf
Loss	Non-convex
γ_t	$\frac{1}{t^\alpha}$
c_t	$\frac{\gamma_t}{d}$
$\|\mathbf{W}-\mathbf{J}\|$	0.4289
$\mathbf{J}$	$\mathbf{1}\mathbf{1}^T/M$

TABLE 2: Parameters Zeroth-Order Distributed FW algorithm.

equal to 0.42 (Fig. 7), in accordance with [2]. With our random seed the algorithm runs for only 4 iterations, misclassifying ∼803 images out of 1'000 at the end, thanks to the KWSA optimization scheme. In particular, such a number of iterations is due to the stopping criterion on the fooling rate with a threshold fixed to 80% (i.e. $\delta = 0.2$), with no need of completing all the `MaxIt` iterations.

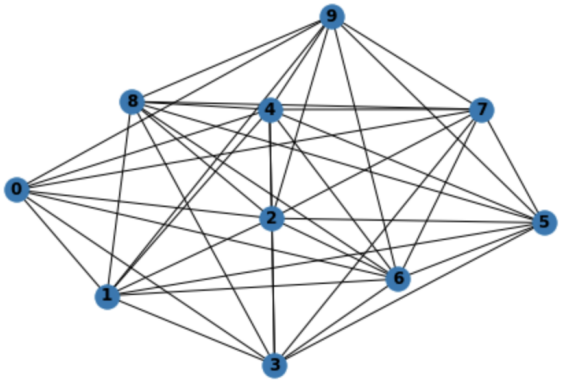

Fig. 7: Graph used in Distributed Zeroth-Order FW.

The choice of such a threshold is due to the notable concavity of the misclassification rate curve with respect to the number of iterations: in fact, it has been seen that gaining around an additional 5% of misclassified images would require around 20 iterations more.

For each image, performing the KWSA step, d queries to the oracle are required, resulting in $\sim$310 seconds per iteration, per node.

The resulting images (Fig. 8) resemble the ones obtained in the previous algorithm: even in this case, the perturbation (Fig. 9) seems a combination of all the considered digits.

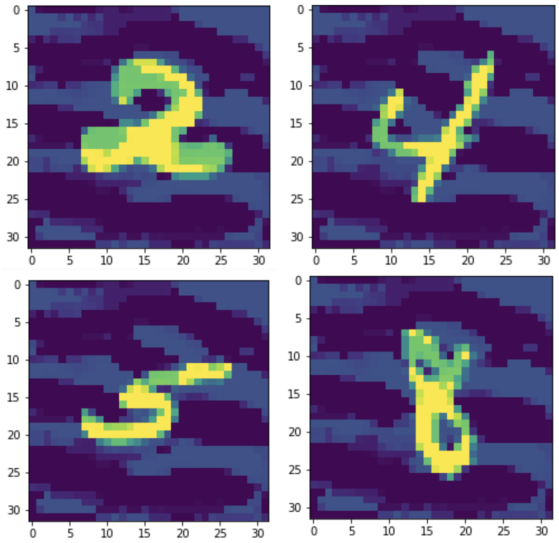

Fig. 8: Final Universal Perturbation Distributed FW.

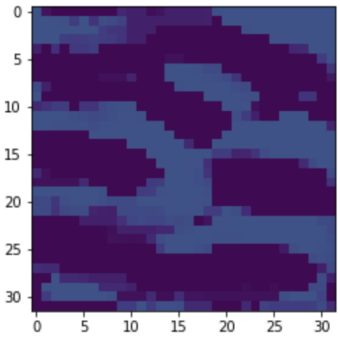

Fig. 9: Final Universal Perturbation Distributed FW.

Since we are considering a distributed setting, each node computes its own perturbation and, at the end of each iteration, it sends its information to all the neighbors. It is interesting to note how, at the beginning, all the noise masks are quite different from each others and, as the number of iterations grows, the pertubations converge, as shown in Fig. 10, 11.

Once again, the main difficulties arise in misclassifying the 8 and 2 digits samples. In particular, the misclassification rate per class obtained with our selected random seed is reported in Fig. 12.

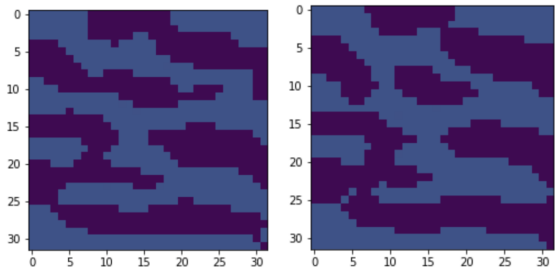

Fig. 10: Noise at iteration 1 Node 0 and Node 1

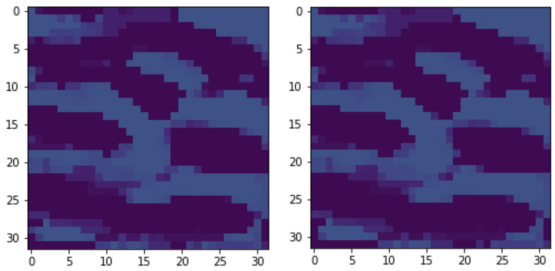

Fig. 11: Noise at iteration 4 Node 0 and Node 1

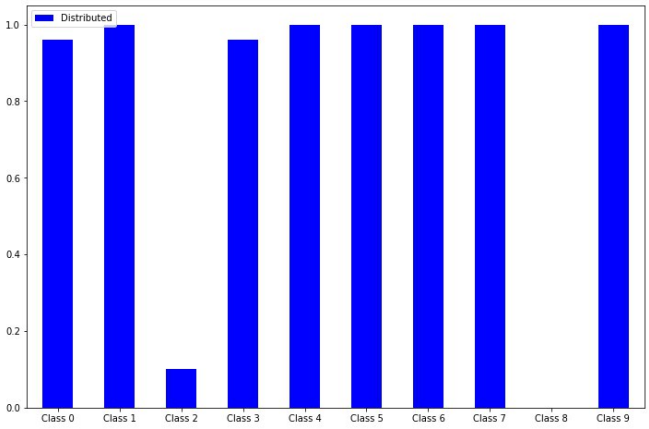

Fig. 12: % of misclassified per class in Distributed FW.

As for the loss and the misclassification rate, most of the observations reported in Subsection V-A are also valid in this case, as shown in Fig. 13 and Fig. 14. In particular, the values in Fig. 13 has been obtained as the mean of the losses computed over the images in each of the M nodes.

C. Zeroth-Order FW with Inexact Update

Our third part of the project exploits an algorithm that creates a single perturbation for each image of the dataset, even though they share some similarities due to the fact that they are generated by an average of random Gaussian vectors.

In Tab. 3 the parameters used in Alg. 3 are shown, and they are derived by a Grid-Search over an hyper-parameters space of cardinality 96. We needed it because the results obtained with the parameters defined in Eq. 17 performed poorly, without an acceptable misclassification rate.

One main problem was the dependence of the γ_t parameter from the Lipschitz constant L. A second one was spotted in the extremely small order of magnitude of the smoothing

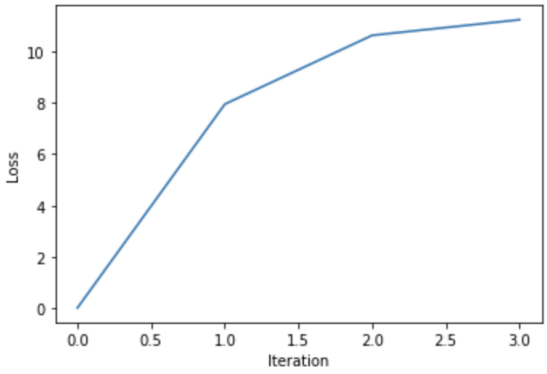

Fig. 13: Loss Function Distributed FW vs Iterations

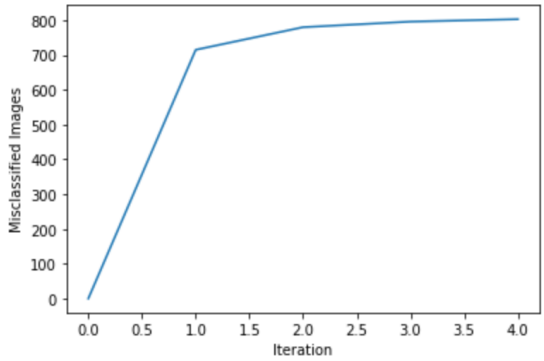

Fig. 14: Misclassification Rate Distributed FW vs Iterations

Parameter	Value
M	10
n	100
d	1024
MaxIt	100
MaxItICG	1000
xi	0.4
L norm	Inf
Loss	Non-convex
ν	1
μ	0.001
m	10240
γ_t	0.001

TABLE 3: Parameters Zeroth-Order FW with Inexact Update.

parameter ν, which would have required a huge number of iterations, unfeasible given our hardware availability. For this reason, our first intervention was to scale up ν in order to reach a better misclassification rate.

The achieved results were still not satisfying, thus, we were also forced to increase ξ, the l_∞ norm's bound of the perturbation, and, given the lack of a benchmark, we preferred to implement the previously mentioned Grid-Search to have a more general and effective overview.

Another detail to be highlighted is the choice of μ: since the parameter is used as a bound in the Alg. 4, changing its order of magnitude affects the number of iterations in the *ICG* method. Even if it could perform, at most, 1'000 iterations, we noted that, after 10 of them, there are no significant changes in the outputs; hence, we fixed $\mu = 0.001$, reaching an *ICG* average iteration number of $\sim$10.

This algorithm takes advantage of the RDSA estimation gradient technique, querying the oracle m times per each iteration, addressed in this third case to a single image of the dataset. For this reason each step requires just $\sim$2 seconds, but only provides an ad hoc perturbation step. Applying the algorithm to the whole dataset would have needed too many computational resources, so we decided to extract a 100-images balanced subset to compute the required statistics.

Our final misclassification rate is 61%, but we need to take into account that some digits, such as 2 and 5, are never misclassified, probably for the peculiar pattern of the images that the LeNet5 model recognizes easily, and other ones, such as 4 and 9, are always misclassified. It is interesting to note that digit 8, differently from the previous algorithms, is misclassified the 40% of the times. The complete overview is reported in Fig. 15.

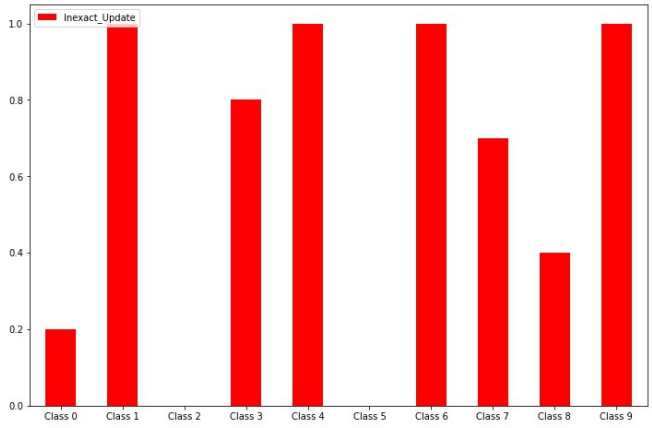

Fig. 15: % of misclassified per class in Inexact Update Algorithm

It is also important, looking at Fig. 16, to note that the noise of each image differs from the ones resulting in the universal perturbation-oriented algorithms; this is due to the fact that the RDSA step has only access to a single image oracle information, also preserving a shape that contains an higher rate of stochasticity. An example of perturbation is shown in Fig. 17.

D. Results comparison

In this last subsection we briefly compare the three algorithms' results.

A first important aspect concerns their running time. The total CPU time requested by the decentralized and the distributed ones is respectively of 775 seconds and 1'236 seconds, while, regarding the Inexact Update, it strongly depends on the number of images that the algorithm is able to misclassify, other that the absence of parallel computing and of the stopping condition over the misclassification rate, not

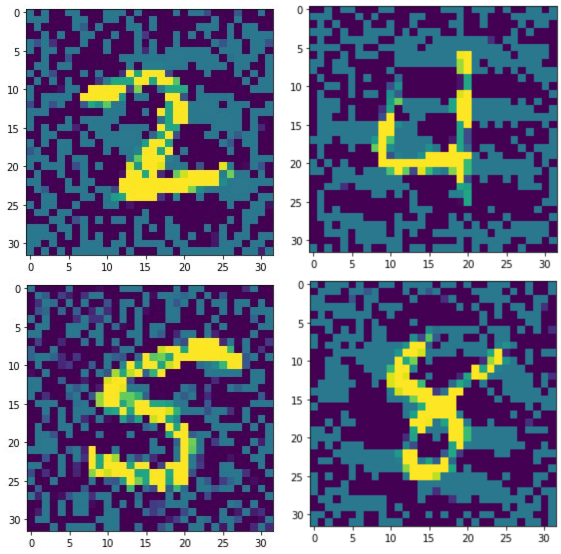

Fig. 16: Perturbated Images Zeroth-Order Inexact Update FW.

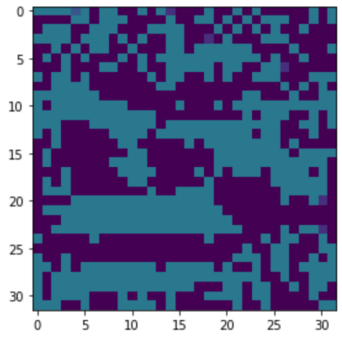

Fig. 17: Example of Noise of Zeroth-Order Inexact Update FW.

implementable in this framework; for our 100-images subset it roughly requires 4 hours.

Moreover, it can seem counterintuitive that the time required by each distributed step is $\sim$10 times the one in the corresponding decentralized iteration. This is mainly due to the fact that:

- parallelizing on a 4-core CPU requiring the existence of 10 workers implies a reduction of the computational power;
- the time needed to share the information between nodes in a peer-to-peer manner is more demanding than broadcasting it from a master node.

Furthermore, Tab. 4 shows the different bounds considered and the principal characteristics of the algorithms.

Algorithm	ξ	N. Queries/It.	Miscl. Rate
Decentralized FW	0.25	1'024/130	70%
Distributed FW	0.25	1'024	80%
Inexact Update FW	0.4	10'240	61%

TABLE 4: Comparison Zeroth-Order FW algorithms

It provides a further explanation about their time requirements. The total number of queries needed in the RDSA framework for a single stochastic step of the decentralized version over a whole batch of a single worker are 130x100=13'000, while a single stochastic step of the Inexact Update algorithm over the same 100 images requires 10'240x100=1'024'000 queries.

Essentially for the same reasons, the misclassification rate of the Inexact Update algorithm is the lowest one: despite the higher noise's norm bound ξ, the availability of a single image to estimate the ascent direction results in a noise shape which departs from the universal ones, performing more poorly at the same time.

These differences also reflect on the distribution of the misclassification rate per class. As it can be seen in Fig. 4, Fig. 12 and Fig. 15, the two universal perturbation algorithms are both not able to misclassify the 8 class, while the Inexact Update noise succeeds almost half of the times. On the other side, the Inexact Update completely fails in misclassifying the 2 and the 5 classes. The 2 challenges the first two algorithms as well, which anyway misclassify it around 20% of the attempts thanks to the global information, while they do not suffer in tackling the 5 digits.

VI. Conclusions

In this work we focused on three variants of the FW algorithm to perform Adversarial Attacks against the deep convolutional neural network LeNet5 and increment its misclassification rate perturbating the MNIST dataset.

We learnt a lot from this project, also thanks to the issues we had in elaborating it: it was deeply constructive discovering more in detail about Black-Box Adversarial algorithms, also to increase our awareness about the possible security issues our models could suffer from and to warn us to make our networks more robust. Also, spotting relevant typos in the starting papers destabilized us, and thought us to question even the most authoritative sources, but forced us to deepen our understanding of the algorithm's foundations, searching for previous and alternative work in the literature.

The implemented algorithms have different characteristics, thus also resulting in different difficulties: we learnt to parallelize code using the `multiprocessing` Python package and about the hardware (memory and CPU-related) issues it can imply.

Future work could consist in extending the proposed methods to more complex datasets, thanks to the availability of more powerful computational resources. Analyzing the convex case could also be interesting to understand the behavior of the algorithms in that framework.

REFERENCES

[1] K. Balasubramanian and S. Ghadimi, "Zeroth-order nonconvex stochastic optimization: Handling constraints, high dimensionality, and saddle points," *Foundations of Computational Mathematics*, pp. 1–42, 2021.

[2] A. K. Sahu and S. Kar, "Decentralized zeroth-order constrained stochastic optimization algorithms: Frank–wolfe and variants with applications to black-box adversarial attacks," *Proceedings of the IEEE*, vol. 108, no. 11, pp. 1890–1905, 2020.

[3] C. Fang, C. J. Li, Z. Lin, and T. Zhang, "Spider: Near-optimal non-convex optimization via stochastic path integrated differential estimator," *arXiv preprint arXiv:1807.01695*, 2018.

[4] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1765–1773, 2017.

[5] D. Willmott, A. K. Sahu, F. Sheikholeslami, F. Condessa, and Z. Kolter, "You only query once: Effective black box adversarial attacks with minimal repeated queries," *arXiv preprint arXiv:2102.00029*, 2021.

[6] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[7] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[8] J. C. Duchi, M. I. Jordan, M. J. Wainwright, and A. Wibisono, "Optimal rates for zero-order convex optimization: The power of two function evaluations," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2788–2806, 2015.

[9] Y. Nesterov and V. Spokoiny, "Random gradient-free minimization of convex functions," *Foundations of Computational Mathematics*, vol. 17, no. 2, pp. 527–566, 2017.