Universal Adversarial Perturbations starring Frank-Wolfe

Chiara Bigarella
chiara.bigarella.1@studenti.unipd.it

Silvia Poletti
silvia.poletti@studenti.unipd.it

Gurjeet Singh
gurjeet.singh@studenti.unipd.it

Francesca Zen
francesca.zen@studenti.unipd.it

Abstract

The main goal of this report is to analyze three different Stochastic Gradient Free Frank-Wolfe algorithms for producing Universal Adversarial Perturbations. These perturbations are designed to fool advanced convolutional neural networks for the classification task on the MNIST dataset. Before discussing the implementation, some key concepts about the adversarial attacks and the Frank-Wolfe framework are introduced. The last two sections instead concern the experiments and the conclusive comparison of the results.

1. Introduction

The content of this report is mainly based on the analysis of three papers: in [3] is introduced the definition of Universal Adversarial Perturbations, while in [4] and [1] are proposed different zeroth-order Frank-Wolfe algorithms that can be used to generate such perturbations. In particular, we carried out some experiments on the basis of the applications described in [4], section VIII.B. In the following subsections, we start by explaining some of the key concepts that are necessary to understand our work.

1.1. Adversarial Attacks

An adversarial attack is a machine learning technique that has the aim of fooling a classifier by providing it with some carefully designed inputs, called *adversarial examples*. An adversarial example is a datapoint that has been perturbed or distorted so that the classifier makes an erroneous prediction.

Adversarial attacks can be divided into targeted and untargeted attacks. The first ones aim to misclassify the input image to a target class, while the latter ones aim to only misclassify the input image with any other class except the true class.

A further distinction is between white-box and black-box attacks, based on the information accessible to the attacker. A *white-box* attack involves the knowledge about the classifier, the trained parameter values and the analytical form of the classifier's loss function. On the contrary, in a *black-box* setting, only a zeroth-order oracle is accessible, i.e. only the values of the loss function are available.

Black-box adversarial attacks can be further categorized into score-based and decision-based attacks. In *score-based* attacks, the attacker directly observes a loss value, class probability, or some other continuous output of the classifier on a given example, whereas in *decision-based* attacks, the attacker observes only the hard label predicted by the classifier.

In this report we will focus on untargeted, black-box, score-based adversarial attacks.

1.2. Universal Adversarial Perturbations

Universal Adversarial Perturbations are small perturbations that, when applied to the input images, are able to fool a state-of-the-art deep neural network classifier. These perturbations are quasi-imperceptible to human eyes, due to their small norm, and therefore they are difficult to detect. However, what makes Universal Adversarial Perturbations special, is their capability to generalize well both on a large dataset and across different deep neural network architectures. In fact, an important property of this kind of perturbations is that they are image-agnostic. This means that they don't depend on a single image, but rather they are able to cause label estimation change for most of the images contained in a dataset.

It is important to notice that there is no unique Universal Adversarial Perturbation: different random shuffling of the data used for the computation can lead to a diverse set of perturbations.

Furthermore, these kinds of perturbations mostly make natural images classified with specific labels, called *dominant labels*. These dominant labels are not determined a priori, but rather they are automatically found by the algorithm that computes the perturbation. In paper [3], the authors hypothesize that «these dominant labels occupy large regions in the image space, and therefore represent good candidate labels for fooling most natural images». We will discuss this axpect in the Experiment and Conclusion sections.

Finally, although some fine-tuning strategies can be adopted to make the deep neural networks more robust to Adversarial Attacks, they are not sufficient to make the classifiers totally immune. In fact, it is always possible to find a small Universal Adversarial Perturbation that can fool the network.

2. Gradient Free Stochastic Frank-Wolfe

Gradient free optimization algorithms find application in settings where the explicit closed form of the loss function is not available or the gradient evaluation is computationally prohibitive. A prime example concerns black-box adversarial attacks on neural networks, where only the model output is known, while the architecture and weights remain unknown. In fact, gradient free methods exploit just zeroth-order oracle calls (i.e. loss function evaluations) to solve an optimization problem.

In particular, this report focuses on zeroth-order Frank-Wolfe algorithms for constrained optimization problems. Unlike the Projected Gradient Descent (PGD) method that requires expensive projection operations, the Frank-Wolfe framework provides computational simplicity by making use of instances of linear minimization. Moreover, at each iteration, a naive zeroth-order analog of PGD takes a step along the estimated gradient aggressively before applying projection, and since this trajectory might not be a descent direction then the method could suffer from slow convergence.

Considering the applications of interest, that are black-box adversarial attacks, a "good enough" feasible solution is often adequate. Therefore, the use of gradient estimates obtained from zeroth-order information is suitable for performing this task.

Let's consider the constrained stochastic optimization problem defined as

$$\begin{aligned}\boldsymbol{\delta}' &= \operatorname*{arg\,max}_{\|\boldsymbol{\delta}\|_\infty \le \varepsilon} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{P}}[F(\mathbf{x}+\boldsymbol{\delta},y)] \\ &= \operatorname*{arg\,min}_{\|\boldsymbol{\delta}\|_\infty \le \varepsilon} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{P}}[-F(\mathbf{x}+\boldsymbol{\delta},y)] \qquad (1)\end{aligned}$$

where the couples $(\mathbf{x}, y)$ represent the labeled data given in input to the classifier and $F(\mathbf{x}, y)$ is the corresponding loss function. This optimization problem formalizes the procedure to generate a Universal Adversarial Perturbation that maximizes the loss function on the convex set defined by the infinity norm $\|\boldsymbol{\delta}\|_\infty = \max_i |\delta_i|$.

According to this formulation, the adversarial examples are designed to lead to misclassification and the constraint ensures a minimal visual distortion to the human eyes.

The zeroth-order optimization is carried out by applying the following biased gradient approximation schemes for $\nabla F(\mathbf{x}_t, y)$:

- KWSA:

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

- RDSA:

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

- I-RDSA:

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

where d is the dimension of the optimization problem at hand, $\{\mathbf{e}_i\}_{i=1}^d$ are the canonical basis vectors and $\{\mathbf{z}_{i,t}\}_{i=1}^m \sim \mathcal{N}(0, \mathbb{1}_d)$ are random vectors sampled from the multivariate standard normal distribution.

It's easy to see that KWSA is the most query hungry scheme but at the same time is very accurate, while RDSA and I-RDSA are potentially inaccurate but less computationally demanding, and since m is chosen to be independent of d then the query complexity doesn't scale with dimension.

Finally, c_t is a time-decaying sequence such that the aforementioned gradient estimators tend to be unbiased for $c_t \to 0$.

In the zeroth-order stochastic Frank-Wolfe framework, the gradient estimate $\mathbf{g}(\mathbf{x}_t, y)$ injects addictional variance to the stochastic counterpart $\nabla F(\mathbf{x}_t, y)$ of the objective function's true gradient $\nabla f(\mathbf{x}_t, y)$, potentially leading to divergence and exacerbating the fact that the Linear Minimization Oracle (LMO)

$$\mathbf{v}_t = \operatorname*{arg\,min}_{\mathbf{s}\in C} \langle \mathbf{g}(\mathbf{x}_t, y), \mathbf{s}\rangle \qquad (2)$$

is constrained to hold just in expectation.

To counter this issue, the following gradient smoothing scheme is applied:

$$\mathbf{d}_t = (1-\rho_t)\mathbf{d}_{t-1} + \rho_t\mathbf{g}(\mathbf{x}_t, y) \qquad (3)$$

where ρ_t is a time-decaying sequence and $\mathbb{E}[\|\mathbf{d}_t - \nabla f(\mathbf{x}_t, y)\|^2]$ can be proved to go to zero asymptotically. Therefore (2) can be rewritten as follows:

$$\mathbf{v}_t = \operatorname*{arg\,min}_{\mathbf{s}\in C} \langle \mathbf{d}_t, \mathbf{s}\rangle. \qquad (4)$$

In particular, given the definition of the convex set C as the constraint in (1), the closed-form solution of the LMO is

$$\mathbf{v}_t = \underset{\|\mathbf{s}\|_\infty \leq \varepsilon}{\arg\min} \langle \mathbf{d}_t, \mathbf{s} \rangle = -\varepsilon \; sign(\mathbf{d}_t). \tag{5}$$

The final updating rule is

$$\mathbf{x}_{t+1} = (1-\gamma_t)\mathbf{x}_t + \gamma_t \mathbf{v}_t \tag{6}$$

where $\gamma_t \in (0, 1]$.

3. Methods

Recently, decentralized and distributed settings are getting significant attention due to the need to exploit data parallelization in case of complex models trained on huge datasets, that typically require a storage that exceeds a machine capacity. In brief, in decentralized configurations the devices (or workers) exchange information with a master node, while in distributed setups they are connected in a peer-to-peer manner and therefore exchange information only with their neighbors.

The following optimization algorithms are designed to accommodate distributed data across multiple devices.

All the proposed algorithms will consider as iterates the perturbations $\boldsymbol{\delta}_t$ converging to the solution of the optimization problem (1). In particular, the iterates $\mathbf{x}_t$ in the previously defined gradient approximation schemes have to be intended as a batch of dataset images $\mathbf{x}$ to which the t-th perturbation $\boldsymbol{\delta}_t$ has been applied, i.e. $\mathbf{x}_t = \mathbf{x} + \boldsymbol{\delta}_t$ are the perturbed data.

3.1. Decentralized Stochastic Gradient Free Frank-Wolfe

In this section the goal is to solve a special case of (1), that is

$$\boldsymbol{\delta}' = \underset{\|\boldsymbol{\delta}\|_\infty \leq \varepsilon}{\arg\min} \frac{1}{M} \sum_{i=1}^{M} \mathbb{E}_{(\mathbf{x},y)\sim \mathcal{P}_i}[-F_i(\mathbf{x} + \boldsymbol{\delta}, y)]. \tag{7}$$

The decentralized architecture is composed of a central node, called master node, and other nodes, called workers, that are connected to it but not to each others. The workers are connected to the master node to read, write and exchange information.

In the Decentralized Stochastic Gradient Free Frank-Wolfe method implemented in Algorithm 1, the data are spread to M workers and the starting point $\boldsymbol{\delta}_0$ is initialized to a zeros d-vector.

Each worker i computes its local gradient using the I-RDSA scheme and considering a batch of images $\mathbf{x}_i$ and the corresponding labels $\mathbf{y}_i$ from the dataset. Then the workers apply the smoothing scheme (3).

When all the workers have finished their tasks, they send their results to the master node, which computes the average of the estimated gradients and send back the new iterate $\boldsymbol{\delta}_{t+1}$, obtained from (5) and (6) with $\gamma_t = \frac{2}{t+8}$.

When all the iterations are done, the algorithm returns the history of all the perturbations computed by the master node.

Algorithm 1 Decentralized SGF FW

```
Input Batches of images $\{\mathbf{x}_i\}_{i=1}^M$, batches of labels $\{\mathbf{y}_i\}_{i=1}^M$, loss function $F$, number of queries $T$, number of workers $M$, image dimension $d$, tolerance $\varepsilon$, number of sampled directions $m$.
Output Universal perturbation's history.
Initialize $\boldsymbol{\delta}_0 = \mathbf{0}$.
for $t = 0, \dots, T-1$ do
    Master node computes parameters required for the computation of the I-RDSA scheme:
        $(\rho_t, c_t)_{I-RDSA} = \left(\frac{4}{(1+\frac{d}{m})^{1/3}(t+8)^{2/3}}, \frac{2\sqrt{m}}{d^{3/2}(t+8)^{1/3}}\right)$
    Each worker $i$ computes the I-RDSA scheme:
    Sample $\{\mathbf{z}_{j,t}\}_{j=1}^m \sim \mathcal{N}(0, \mathbb{1}_d)$
        $\mathbf{g}_i = \frac{1}{m}\sum_{j=1}^{m} \frac{F(\mathbf{x}_i + \boldsymbol{\delta}_t + c_t\mathbf{z}_{j,t}, \mathbf{y}_i) - F(\mathbf{x}_i + \boldsymbol{\delta}_t, \mathbf{y}_i)}{c_t}\mathbf{z}_{j,t}$
    Each worker $i$ computes the gradient smoothing scheme:
        $\mathbf{g}_{i,t} = (1-\rho_t)\mathbf{g}_{i,t-1} + \rho_t\mathbf{g}_i.$
    Each worker $i$ pushes $\mathbf{g}_{i,t}$ to the master node.
    Master node computes the gradients average:
        $\mathbf{g}_t = \frac{1}{M}\sum_{i=1}^{M}\mathbf{g}_{i,t}.$
    Master node computes $\mathbf{v}_t = -\varepsilon sign(\mathbf{g}_t)$.
    Master node computes $\boldsymbol{\delta}_{t+1} = (1-\gamma_t)\boldsymbol{\delta}_t + \gamma_t\mathbf{v}_t$ and sends it to all nodes.
```

3.2. Decentralized Variance-Reduced Stochastic Gradient Free Frank-Wolfe

In this section the goal is to solve another special case of (1), that is

$$\boldsymbol{\delta}' = \underset{\|\boldsymbol{\delta}\|_\infty \leq \varepsilon}{\arg\min} \sum_{i=1}^{M}\sum_{j=1}^{n} \mathbb{E}_{(\mathbf{x},y)\sim \mathcal{P}_{i,j}}[-F_{i,j}(\mathbf{x} + \boldsymbol{\delta}, y)]. \tag{8}$$

The implementation takes into account the SPIDER variance reduction technique, which is built for dynamic tracking, while avoiding excessive querying to oracles and ultimately reducing query complexity.

In the Decentralized Variance-Reduced Stochastic Gradient Free Frank-Wolfe method implemented in Algorithm 2, at the beginning of each period, that is when $mod(t,q) = 0$ with period parameter $q \in \mathbb{N}_+$, each worker employs the KWSA scheme for the computation of its gradient estimation. In all the other cases, each worker selects a minibatch of local component functions $\{F_{i,j}\}_{j \in S_2}$ and uses the RDSA scheme to estimate and update its gradient. Then, the master node computes the average of the received estimated gradients and calculates the new perturbation using the equation (6) with $\gamma_t = \frac{2}{t+8}$.

For what concerns the data, each worker receives the same amount S_1 of images (selected to be balanced with respect to the ten classes) and the corresponding labels from the dataset. In particular, the same image is not given to different workers. For both the KWSA and the RDSA schemes, each worker samples a batch of $\frac{S_1}{M}$ images, from the received ones. For the KWSA scheme, this sampling is done d times, one for each component of the gradient.

3.3. Distributed Stochastic Gradient Free Frank-Wolfe

This section concerns the discussion of the Distributed Stochastic Grandient Free Frank-Wolfe method implemented in Algorithm 3 for the constraint optimization problem (7) in a distributed setup in which the M workers do not have a central coordinator, instead they exchange information in a peer-to-peer manner. The internode communication network used by the workers is modeled as an undirected simply connected graph $G = (V, E)$, with $V = \{1, \dots, M\}$ the set of nodes and E the set of comunication links. Each node communicates and exchanges information with its own neighbors. Given a node n, the set $\Omega_n = \{l \in V | (n,l) \in E\}$ indicates its neighborhood and $d_n = |\Omega_n|$ indicates its degree.

The $M \times M$ adjacency matrix $\mathbf{A} = [A_{ij}]$ describes the edges of the graph G: $A_{ij} = 1$ if $(i,j) \in E$ and $A_{ij} = 0$ otherwise. The diagonal matrix $\mathbf{D} = diag(d_1 \dots d_M)$ is used to compute the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$. The normalized Laplacian $\boldsymbol{\mathcal{L}} = [\mathcal{L}_{ij}]$ is defined to be the matrix

$$\mathcal{L}_{ij} = \begin{cases} 1 & \text{if } i = j \text{ and } d_i \neq 0, \\ -\frac{1}{\sqrt{d_i d_j}} & \text{if } i \text{ and } j \text{ are adjacent,} \\ 0 & \text{otherwise.} \end{cases}$$

We can write $\boldsymbol{\mathcal{L}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$. The $\boldsymbol{\mathcal{L}}$ matrix is used to compute the weghted matrix $\mathbf{W} = \mathbb{1} - \boldsymbol{\mathcal{L}}$.

Unlike the previous algorithms, the perturbation $\boldsymbol{\delta}_0$ is initialized as a zeros $M \times d$ matrix, rather than a vector: this matrix is given in input to each worker, which only considers the rows corresponding to the perturbations computed by its neighbors. This trick simplifies the distributed architecture, without changing the functioning of the algorithm.

Algorithm 2 Decentralized Variance-Reduced SGF FW

```
Input Batches of images $\{\mathbf{x}_i\}_{i=1}^M$, batches of labels $\{\mathbf{y}_i\}_{i=1}^M$, loss function $F$, number of queries $T$, number of workers $M$, image dimension $d$, tolerance $\varepsilon$, number of sampled images and labels $S_1$, number of sampled component functions $S_2$, total number of component functions $n$, period $q$.
Output Universal perturbation's history.
Initialize $\boldsymbol{\delta}_0 = \mathbf{0}$.
for $t = 0, \dots, T-1$ do
    Master node computes parameters required for the computation of the KWSA and RDSA schemes:
    $$\eta_1 = \frac{2}{d^{1/2}(t+8)^{1/3}}, \quad \eta_2 = \frac{2}{d^{3/2}(t+8)^{1/3}}$$
    Each worker $i$ computes:
    if $mod(t,q) = 0$ then
        For each component $k$ of the gradient, draw $\frac{S_1}{M}$ images $\mathbf{x}'_i$ and the corresponding labels $\mathbf{y}'_i$ from the batches $\mathbf{x}_i$ and $\mathbf{y}_i$, for a total of $\frac{S_1}{M}d$ samples. For $k = 1 \dots d$ compute
        $$\mathbf{e}_k^T \mathbf{g}_i = \frac{1}{n}\sum_{j=1}^{n} \frac{F_{i,j}(\mathbf{x}'_i + \boldsymbol{\delta}_t + \eta_1 \mathbf{e}_k, \mathbf{y}'_i) - F_{i,j}(\mathbf{x}'_i + \boldsymbol{\delta}_t, \mathbf{y}'_i)}{\eta_1}$$
        Update $\mathbf{g}_{i,t} = \mathbf{g}_i$.
    else
        Sample $\mathbf{z} \sim \mathcal{N}(0, \mathbb{1}_d)$. Draw $S_2$ component functions and $\frac{S_1}{M}$ images $\mathbf{x}'_i$ and the corresponding labels $\mathbf{y}'_i$ from the batches $\mathbf{x}_i$ and $\mathbf{y}_i$. Then compute
        $$\mathbf{g}_i = \frac{1}{|S_2|}\sum_{j \in S_2} \frac{F_{i,j}(\mathbf{x}'_i + \boldsymbol{\delta}_t + \eta_2 \mathbf{z}, \mathbf{y}'_i) - F_{i,j}(\mathbf{x}'_i + \boldsymbol{\delta}_t, \mathbf{y}'_i)}{\eta_2}\mathbf{z} - \frac{F_{i,j}(\mathbf{x}'_i + \boldsymbol{\delta}_{t-1} + \eta_2 \mathbf{z}, \mathbf{y}'_i) - F_{i,j}(\mathbf{x}'_i + \boldsymbol{\delta}_{t-1}, \mathbf{y}'_i)}{\eta_2}\mathbf{z}$$
        Update $\mathbf{g}_{i,t} = \mathbf{g}_i + \mathbf{g}_{i,t-1}$.
    Each worker $i$ pushes $\mathbf{g}_{i,t}$ to the master node.
    Master node computes the gradients average:
    $$\mathbf{g}_t = \frac{1}{M}\sum_{i=1}^{M} \mathbf{g}_{i,t}.$$
    Master node computes $\mathbf{v}_t = -\varepsilon sign(\mathbf{g}_t)$.
    Master node computes $\boldsymbol{\delta}_{t+1} = (1-\gamma_t)\boldsymbol{\delta}_t + \gamma_t \mathbf{v}_t$ and sends it to all nodes.
```

At every iteration, each worker i exchanges its current iterate with its neighbors and averages the received iterates. Then, each worker i applies the I-RDSA scheme to compute its local gradient estimation $\mathbf{g}_t^i$ and exchanges with the neighbors the vector $\mathbf{G}_t^i$, which is an averaged version of the gradient. Finally, when all the $\mathbf{G}_t^i$ have been calculated, they are involved in a weighted sum computed by each worker.

For what has been said until now, the implementation requires two round of communication: one for the iterates $\bar{\mathbf{x}}_t^i$ and one for the gradients $\bar{\mathbf{g}}_t^i$. The performance depends on how well these vectors are tracked across the network.

At the end of the t-th iteration, each worker computes the Frank-Wolfe update of the iterate $\boldsymbol{\delta}_{t+1}^i$ using the LMO closed form (5) on $\bar{\mathbf{g}}_t^i$ and the update rule (6) on $\bar{\mathbf{x}}_t^i$ with $\gamma_t = t^{-1/2}$.

Algorithm 3 Distributed SGF FW

```
Input Batches of images $\{\mathbf{x}_i\}_{i=1}^M$, batches of labels $\{\mathbf{y}_i\}_{i=1}^M$, loss function $F$, number of queries $T$, number of workers $M$, image dimension $d$, tolerance $\varepsilon$, number of sampled directions $m$, adjacency matrix $\mathbf{A}$, weight matrix $\mathbf{W}$.
Output $\bar{\boldsymbol{\delta}}_T^i \ \ \forall i \in \{1 \dots M\}$
For $i = 1, \dots, M$ initialize $\boldsymbol{\delta}_0^i = \mathbf{0}$ and $\bar{\mathbf{g}}_0^i = \mathbf{0}$
for $t = 1, \dots, T$ do
    Compute the parameter required for the computation of the I-RDSA schemes: $c_t = \frac{2\sqrt{m}}{d^{3/2}(t+8)^{1/3}}$
    In the first round of communication, each worker $i$ approximates the average iterate:
        $$\bar{\boldsymbol{\delta}}_t^i \leftarrow \sum_{j=1}^{M} W_{ij} \boldsymbol{\delta}_t^i$$
    Each worker $i$ computes the I-RDSA scheme:
    Sample $\{\mathbf{z}_{j,t}\}_{j=1}^m \sim \mathcal{N}(0, \mathbb{1}_d)$
        $$\mathbf{g}_t^i = \frac{1}{m} \sum_{j=1}^{m} \frac{F(\mathbf{x}_i + \boldsymbol{\delta}_t + c_t \mathbf{z}_{j,t}, \mathbf{y}_i) - F(\mathbf{x}_i + \delta_t, \mathbf{y}_i)}{c_t} \mathbf{z}_{j,t}$$
    Each worker $i$ computes:
        $$\mathbf{G}_t^i = \bar{\mathbf{g}}_{t-1}^i + \mathbf{g}_t^i - \mathbf{g}_{t-1}^i$$
    In the second round of communication, each worker $i$ approximates the average gradient:
        $$\bar{\mathbf{g}}_t^i \leftarrow \sum_{j=1}^{M} W_{ij} \mathbf{G}_t^j$$
    Each worker $i$ computes $\mathbf{v}_t^i = -\varepsilon sign(\bar{\mathbf{g}}_t^i)$.
    Each worker $i$ updates:
        $$\boldsymbol{\delta}_{t+1}^i \leftarrow (1 - \gamma_t) \bar{\boldsymbol{\delta}}_t^i + \gamma_t \mathbf{v}_t^i$$
```

4. Experiments

The data we used in our experiments come from the MNIST dataset, which contains 60000 train images and 10000 test images from 10 almost balanced classes (arabic numerals). Each of the following simulations generate a Universal Adversarial Perturbation starting from a portion of the MNIST test dataset. Then the perturbation is added to the test images that have been correctly classified by the pre-trained LeNet-5 convolutional neural network, aiming to maximally increase the loss function, and therefore minimize the accuracy.

In particular, the perturbation has to inject a minimal visual distortion to the original images and this is ensured by the constraint on the ℓ_∞ norm in the optimization problem (1). Therefore, the digits in the perturbed images still appear clearly distinguishable to the human eyes, but get misclassified by LeNet-5.

In the following experiments, the ℓ_∞ norm of the perturbation is chosen so as not to be higher than $\varepsilon = 0.25$.

For simplicity, all the algorithms introduced in the previous section have been implemented in a sequential fashion, rather than using a proper distributed architecture. Therefore, the workers do not effectly represent different processors in different machines, but they are just different methods called by the same machine, imitating a distributed setting. Nevertheless, all the algorithms can be easily modified to accomodate a proper distributed architecture by configuring Ray[1] or PySpark[2] libraries.

Again, for simplicity, the algorithms do not involve a normalization technique for the perturbed images and therefore normalization has been applied only after the computation of the Universal Adversarial Perturbation, that is just before the testing of LeNet-5. A possible improvement could be to consider the Box constraint described in [2], section V.B.

In addition, a key concept of the Frank-Wolfe theory is the duality gap, that is an upper bound for the primal suboptimality $F(\mathbf{x}_t) - F(\mathbf{x}^*)$ defined as:

$$\mathcal{G}(\mathbf{x}) = \max_{\mathbf{s} \in C} \langle \nabla F(\mathbf{x}), \mathbf{x} - \mathbf{s} \rangle. \tag{9}$$

Therefore, a further improvement of the implementation could be the use of the Frank-Wolfe duality gap as a stopping criterion for the studied algorithms.
However, this turned out to be difficult to develop in all the three algorithms, since we do not have a direct access to the perturbed images from the master node in the decentralized setting. Instead, in the distributed architecture, the presence

[1] Ray: https://github.com/ray-project/ray.

[2] PySpark: https://spark.apache.org/docs/latest/api/python.

of a synchronized communication among workers results in a difficult global stopping criteria.

4.1. Decentralized SGF FW Experiments

To study the performance of Algorithm 1 we considered a portion of the MNIST test set made of 1000 samples. Each of the $M = 10$ workers received 100 images, 10 for each class. We then tuned the hyperparameter m for the number of directions to be sampled in the I-RDSA scheme. The best value for which the lowest accuracy (approximatively 20%) of LeNet-5 was reached is $m = 35$, as shown in Figure 1.

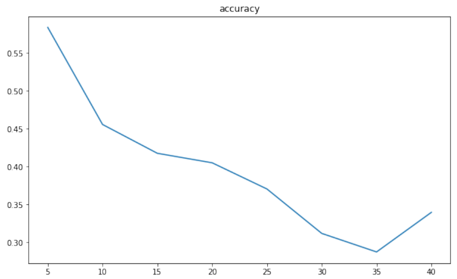
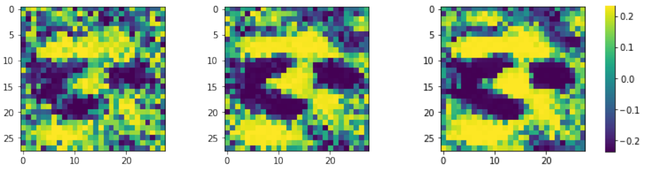


Figure 1: This plot was obtained by running Algorithm 1 multiple times, while changing the value of the parameter m (horizontal axis) from 5 to 40 by keeping all the other parameter fixed. The vertical axes indicates the values of the accuracy.

With the increasing of m, the running time becomes very large and for this reason we had to find a value for m which returns a low accuracy with respect to a reasonable running time. We ended up with $m = 15$, which is a good compromise between the previous ones, and we kept this value fixed for all the runs.

In particular, we performed three different experiments, considering $T = 20$, $T = 50$ and $T = 100$. As the number of queries increases, the pattern of the perturbation becomes clearer, and in Figure 2 we can see how the 3-shape perturbation becomes more evident for higher values of T.



(a) (b) (c)

Figure 2: Perturbations generated by the Decentralized SGF FW algorithm with different numbers of queries: (a) T=20, (b) T=50, (c) T=100.

This behaviour is coherent with the decreasing of the accuracy reported in Table 1: to a lower accuracy corresponds a sharper pattern in the perturbation.

It is interesting to notice that with $T = 20$ queries of Algorithm 1 the accuracy of LeNet-5 already decreases to 55,25%. In Figure 3 is represented the perturbation of Figure 2 (c) applied on an image from class 4. The perturbed images have been classified as 3, consistently with the 3-shape of the perturbation.

Attack	**20 queries**	**50 queries**	**100 queries**
Decentralized SGF FW	55,25%	39,46%	36,87%

Table 1: Summary of ℓ_∞ Universal Adversarial Perturbation with ε=0.25. MNIST attacks using Decentralized SGF FW. The entries of the table represent the accuracies of LeNet-5 for the three different attacks.

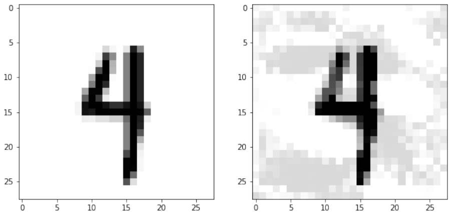


Figure 3: Image belonging to class 4 but classified as 3 by using the adversarial perturbation in Figure 2 (c) generated by the Decentralized SGF FW method.

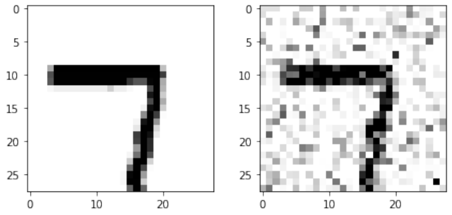


Figure 4: Image belonging to class 7 and correctly classified by the network despite the gaussian noise.

To have a yardstick, the predictions of LeNet-5 remain very precise (89.17% of accuracy) when tested on the images perturbed with random noise (Gaussian with mean 0 and standard deviation 0.3). Even though Figure 4 shows a very evident noise compared to the one of the adversarial example in the previous image, the network still classifies correctly the images perturbed with a Gaussian noise.

It is noteworthy the comparison between the attacks: the Gaussian perturbation spreads the noise randomly in the space, while the Decentralized SGF FW's adversarial perturbation is characterize by a clear pattern that covers the majority of the image space and allows to fool the network on most images.

4.2. Decentralized Variance-Reduced SGF FW Experiments

The experiments performed with Algorithm 2 were conducted considering $M = 5$ workers. Each worker was fed with $S_1 = 800$ images (80 images per class) from the portion of the MNIST test set that LeNet5 was able to classify correctly. We imposed that the same image could not be assigned to different workers. This aspect was not clearly specified in the original algorithm proposed in [4], section VI, in which the authors seem to suggest to use the same S_1 images for all the workers. However, the distributed data settings arise from the need of dividing huge datasets into different machines, for simplifying the computational complexity, and therefore we opted for an implementation choice that resulted more coherent with this idea.

The number M of workers was halved compared to the one of the previous algorithm and also the other hyperparameters was chosen to be pretty low to reduce a bit the CPU-time consumption. In particular, we set the number of component functions to $n = 5$ or $n = 10$, the number of sampled components in RDSA to $S_2 = 3$, the number of queries to $T = 20$ and the period parameter to $q = 5$ or $q = 7$ or $q = 9$. With these settings, the algorithm approximatively took between one and four hours to terminate. In fact, the algorithm combines the hungry but accurate queries of KWSA, regulated by the parameter q, with the efficient but potentially inaccurate queries of RDSA.

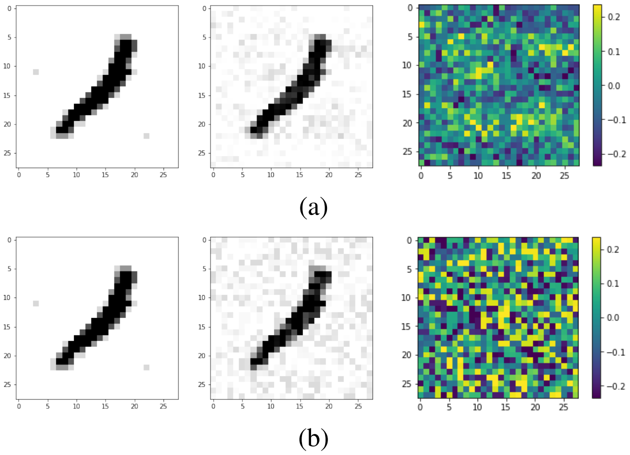
In Table 2 are reported the values of the accuracy reached by LeNet-5 with the adversarial examples obtained using the perturbation generated by Algorithm 2. The attack seems to be more effective as the value of q decreases, i.e. as the KWSA scheme is performed more frequently.

Attack		q=5	q=7	q=10
Decentralized Variance- Reduced SGF FW	n=5	92,37%	97,43%	94,18%
	n=10	84,00%	93,66%	98,99%

Table 2: Summary of ℓ_∞ Universal Adversarial Perturbation with ε=0.25. MNIST attacks using Decentralized Variance-Reduced SGF FW. The entries of the table represent the accuracies of LeNet-5 for the three different attacks. For increasing values of q the algorithm performs the KWSA scheme less and less times.

Figure 5 compares two adversarial perturbations obtained with same n but different values of q. It's clear that the perturbation computed with the lowest q is smoother than the other and less visible to the human eyes.

By looking at the colorbar, we can also observe that a higher q results in higher absolute values of the perturbation's components: bright yellow and dark blue pixels indicate values near $\pm\varepsilon$ and are much more frequent in the perturbation computed with the highest q.



(a)

(b)

Figure 5: (a) Adversarial example obtained from Algorithm 2 with $n = 10$, $q = 5$. (b) Adversarial example obtained from Algorithm 2 with $n = 10$, $q = 10$.

4.3. Distributed SGF FW Experiments

To test the performance of Algorithm 3 we used a distributed network of $M = 10$ worker nodes and we gave them 10 images per class each, for a total of 100 different images each.

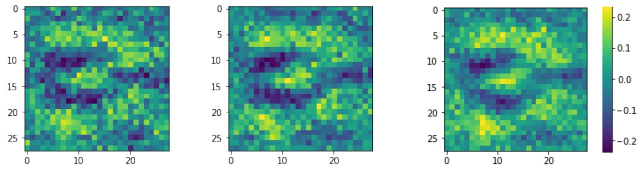
The connectivity of the graph describing the distributed setting is represented by the value $\|\mathbf{W} - \mathbf{J}\|$, where $\mathbf{J} = \mathbf{1}\mathbf{1}^T/M$ with $\mathbf{1}\mathbf{1}^T$ the matrix having all entries set to 1.
For the experiments, we created a network with a connectivity value of 0.438 given by the following adjacecy matrix:

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Since each node is connected to itself, the diagonal of the matrix A is filled with ones. Furthermore, we computed the I-RDSA approximated gradient scheme along $m = 15$ directions and we tested the algorithm for $T = 20$, $T = 50$ and $T = 100$ queries.

In Figure 6 we can see the Universal Adversarial Perturbations produced by Algorithm 3 for different values of T.

It's clear that the three perturbations are very similar to each other and all of them present a quite evident 3-shape.



(a) (b) (c)

Figure 6: Perturbations generated by the Distributed SGF FW algorithm with different numbers of queries: (a) T=20, (b) T=50, (c) T=100.

In Figure 7 is represented the perturbation of Figure 6 (c) applied on an image from class 2.

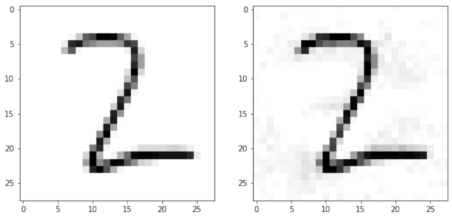


Figure 7: Image belonging to class 2 but classified as 3 by using the adversarial perturbation in Figure 6 (c) generated by the Distributed SGF FW method.

Although we can still recognize the 3-shape pattern in the perturbation, it is much smoother compared to the perturbation in Figure 2 (c) computed by the Decentralized SGF FW algorithm. Nevertheless, it is strong enough to fool the classifier and lead it to a wrong prediction.

Table 3 displays the values of accuracy reached by the LeNet-5 classifier on perturbed images, generated using the results of Algorithm 1 with different amounts of queries.

Attack	**20 queries**	**50 queries**	**100 queries**
Distributed SGF FW	73.47%	74.62%	75.84%

Table 3: Summary of ℓ_∞ Universal Adversarial Perturbation with ε=0.25. MNIST attacks using Distributed SGF FW. The entries of the table represent the accuracies of LeNet-5 for the three different attacks.

We can see that there's no significant difference among the accuracy values corresponding to different number of queries. What we can notice is that the perturbation becomes slightly less effective in fooling the classifier, as long as we increase the number of queries.

Even if the lowest accuracy reached with the perturbations obtained from Algorithm 3 is just 73.47%, this result is still better than the 89.17% of accuracy reached using Gaussian noise.

4.4. Generalization across Deep Neural Networks

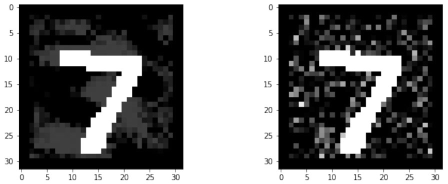
In the previous experiments, expecially for what concerns Algorithm 1 which produced the best perturbations, we showed a remarkable generalization property of the universal avdersarial attacks: perturbations that have been generated starting from just a small portion of the dataset (1000 images out of 10000) are able to fool new images with high efficacy.

Moreover, in [3] has been shown that Universal Avdersarial Perturbations are not only universal across images, but also generalize well across different deep neural network architectures. Therefore we implemented the AlexNet convolutional neural network and test it on the adversarial examples produced by considering the loss function of LeNet-5.

Since the version of AlexNet that we implemented requires in input mini-batches of 3-channel RGB normalized images of shape $32 \times 32 \times 3$, we had to apply a constant padding to the 28×28 images of MNIST in order to obtain a shape of 32×32. After normalization, we extended the dimension to $32 \times 32 \times 1$ and used the method *repeat* from TensorFlow on the third axis to get the desired shape of $32 \times 32 \times 3$.

On the original MNIST test set, AlexNet achieved an accuracy of 98,18% which abruptly dropped to 32,95% on the adversarial examples obtained by applying the 100 queries-perturbation generated by Algorithm 1.

To understand the entity of the attack, note that AlexNet was able to get a 73,73% of accuracy on the same images perturbed with random noise (Gaussian with mean 0 and standard deviation 0.3). Figure 8 shows the difference between an adversarial and a noisy example.



(a) (b)

Figure 8: (a) Adversarial example. (b) Noisy example.
Note that the dataset preprocessing needed for the correct functioning of AlexNet, transformed the images in their negative colored version.

5. Conclusions

In this report we focused on the problem of producing Universal Adversarial Perturbations by analyzing three Stochastic Gradient Free Frank-Wolfe algorithms.

First of all, we have shown that the perturbations created by Decentralized (1) and Distributed (3) SGF FW algorithms present a similar and more clear pattern compared to the Decentralized Variance-Reduced SGF FW Algorithm (2). In particular, we can clearly see that the reproduced pattern has a 3-shape, which leads the majority of handwritten digits to be misclassified as 3 or 8, which has a similar shape. This can be explained by the concept of *dominant labels*, mentioned in Section 1.2. In fact, digit 3 is a wide number, that covers most of the space in the image. Therefore, a perturbation with a 3-shape can easily lead to the misclassification of smaller numbers such as 1 and 7, which occupy less space in the image. On the contrary, the perturbations produced by the Decentralized Variance-Reduced SGF FW algorithm, don't have a clear pattern and the noise associated with them looks randomly spread.

Secondly, the algorithm that reached better results in terms of misclassification is Algorithm 1, which lowered the classifier's accuracy to 55%. In this sense, the worst algorithm was 2 since it was unable to decrease the classifier's accuracy below 84%. A possible improvement could be a better tuning of the hyperparameters; in fact in our experiments we considered quite low values of the number M of workers, the number T of queries and the number S_2 of sampled component functions and we also set the period parameter q so that the KWSA scheme was called at most 4 times.

If we compare the execution time of the three algorithms, we can observe that algorithms 1 and 3 are much faster then Algorithm 2. This is due to the fact that Algorithm 2 employs KWSA, which is very expensive in terms of CPU time.

Compared to the experiments described in paper [4], we obtained slightly higher error rates with Algorithm 1, while, with Algorithm 3 , we achieved lower error rates. The latter result can be explained by the fact that we chose to use the I-RDSA scheme with $m = 15$ instead of the KWSA scheme, to reduce the time complexity of the algorithm, although the KWSA scheme gives a more precise gradient approximation.

Furthermore, the distributed setting of Algorithm 3 naturally leads to a less precise gradient approximation than the one of Algorithm 1, due to the fact that each node has access only to the computations made by its neighbors. Therefore, the choice of a less precise method to compute the gradient, i.e. the I-RDSA scheme with a small value for m, makes the resulting perturbation less performing. Nevertheless, the attack performed with the perturbation obtained from Algorithm 3 is satisfying enough, since random noise resulted to be much less effective.

Moreover, it has to be noticed that although the perturbations of Algorithm 1 lower more the accuracy than the ones of Algorithm 3, the latter are much less visible. This can be easily seen by comparing the adversarial example in Figure 2 with the one in Figure 7.

Finally, in our last experiment we proved that the perturbation created with Algorithm 1 on LeNet-5's loss function is universal not only with respect to the MNIST dataset, but also across different deep neural network architectures, such as AlexNet.

References

[1] Manzil Zaheer Anit Kumar Sahu and Soummya Kar. Towards gradient free and projection free stochastic optimization, 2019.

[2] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks, 2017.

[3] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawz, and Pascal Frossard. Universal adversarial perturbations, 2017.

[4] Anit Kumar Sahu and Soummya Kar. Decentralized zeroth-order constrained stochastic optimization algorithms: Frank–wolfe and variants with applications to black-box adversarial attacks, 2020.