Universal Adversarial Perturbation starring Frank-Wolfe

Chiara Bigarella
Student nr. 2004248

Silvia Poletti
Student nr. 1239133

Gurjeet Singh
Student nr. 2004251

Francesca Zen
Student nr. 2010640

1. Introduction

TODO

1.1. Adversarial Attacks

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

Adversarial attacks can be divided into targeted and untargeted attacks. *Targeted* attacks have the aim to misclassify the input image to a target class, while *untargeted* attacks are aimed at only misclassifying the input image to any other class except the true class.

A further way to categorize adversarial attacks is to divide them into white-box and black-box attacks, based on the information accessible to the attacker. To perform a *white-box* attack, the classifier, the trained parameter values, and the analytical form of the classifier loss function must be all available to the attacker. On the contrary, in a *black-box* setting, only a zeroth-order oracle is accessible.

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

1.2. Universal Adversarial Perturbations

Universal 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 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 universal 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's no unique universal perturbation: different random shuffling of the data used to compute the perturbation can lead to a diverse set of universal perturbations.

Furthermore, universal 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 universal perturbation. In paper XXX, 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".

Finally, although some fine-tuning strategies can be adopted to make deep neural networks more robust to adversarial attacks, they are not sufficient to make the classifiers immune to the attacks. In fact, the fine-tuning of a classifier with universal perturbations leads to an improvement in the classification of perturbed images, however it is always possible to find a small universal 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

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

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 as 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)

$$\mathbf{v}_t = \underset{\mathbf{s}\in C}{\arg\min}\langle \mathbf{g}(\mathbf{x}_t, y), \mathbf{s}\rangle \tag{2}$$

is constrained to hold just in expectation.
To counter this issue, the following gradient smoothing scheme is applied:

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

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:

$$\mathbf{v}_t = \underset{\mathbf{s}\in C}{\arg\min}\langle \mathbf{d}_t, \mathbf{s}\rangle. \tag{4}$$

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 the need to exploit data parallelization in case of complex models trained on huge data, that typically require a storage that exceeds the machine capacity. In brief, in decentralized setups the devices (or workers) exchange information with a master node, while in distributed setups the devices 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 calls.
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

The decentralized architecture is composed of a central node, called master node, and other nodes connected to it but not to each other, called workers. The workers are connected to the master node to read, write and exchange information.
In the Decentralized Stochastic Gradient Free Frank Wolfe method, the data is spread to M workers and the starting point and the gradient are initialized to a null d-vector and a null $M \times d$ matrix, respectively.
The workers compute their local gradient using the I-RDSA scheme on the loss function and then 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 of the iterations are done, the algorithm returns the hystory of all the perturbations computed by the master node.

3.2. Decentralized Variance-Reduced Stochastic Gradient Free Frank Wolfe

This section analyzes the SPIDER variance reduction technique, which is built for dynamic tracking, while avoiding excessive querying to oracles and ultimately reducing query complexity.
At the beginning of Algorithm 2 we initialize our initial perturbation to a zero d vector $\boldsymbol{\delta_0}$ and denote $\boldsymbol{q} \in \mathbb{N}_+$ as a period parameter. At the beginning of each period, i.e. $\boldsymbol{mod(q,n) = 0}$, each worker employ KWSA for the computation of its own gradient estimation along the canonical basis vector $\boldsymbol{e_k}$, with $\boldsymbol{k = 1, \ldots, d}$, using $\boldsymbol{\eta = \frac{2}{d^{1/2}(t+8)^{1/3}}}$. In all other cases, the worker selected a mini-batch $\boldsymbol{S'}$ of local component functions and use the RDSA to estimate and update the gradients. In particular, $\boldsymbol{S_2}$ pairs of component functions are chosen for the computation of the gradient estimation

$$\mathrm{e}_{\boldsymbol{k}}^{\boldsymbol{T}} \mathrm{g}_{\boldsymbol{i}}(\boldsymbol{\delta_t}) = \frac{1}{\boldsymbol{n}} \sum_{\boldsymbol{j}=1}^{\boldsymbol{n}} \frac{\boldsymbol{F_{i,j}}(\mathrm{x}_{\boldsymbol{t}} + \boldsymbol{\eta} \mathrm{e}_{\boldsymbol{k}}) - \boldsymbol{F_{i,j}}(\mathrm{x}_{\boldsymbol{t}})}{\boldsymbol{\eta}},$$

which is then sent to the master node. While running RDSA we use $\boldsymbol{\eta = \frac{2}{d^{3/2}(t+8)^{1/3}}}$.
Then, the master node computes the average of the estimated gradients and calculates the new perturbation. Algorithm 2 returns 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 ε, number of sampled directions m.
Output Universal perturbation's history.

```
1: Initialize $\boldsymbol{\delta}_0 = \mathbf{0}$.
2: for $t = 0, \ldots, T-1$ do
3:     Master node computes parameters required for the computation of the I-RDSA scheme:
       $$(\rho_t, c_t)_{I-RDSA} = \left(\frac{4}{(1+\frac{d}{m})^{1/3}(t+8)^{2/3}}, \frac{2\sqrt{m}}{d^{3/2}(t+8)^{1/3}}\right)$$
4:     Each worker $i$ computes the I-RDSA scheme:
       Sample $\{\mathbf{z}_{j,t}\}_{j=1}^m \sim \mathcal{N}(0, \mathbb{1}_d)$
       $$\mathbf{g}_i = \frac{1}{m}\sum_{j=1}^m \frac{F(\mathbf{x}_i + \boldsymbol{\delta}_t + c_t\mathbf{z}_{j,t}, \mathbf{y}_i) - F(\mathbf{x}_i + \boldsymbol{\delta}_t, \mathbf{y}_i)}{c_t}\mathbf{z}_{j,t}$$
5:     Each worker i computes the gradient smoothing scheme:
       $$\mathbf{g}_{i,t} = (1-\rho_t)\mathbf{g}_{i,t-1} + \rho_t\mathbf{g}_i.$$
6:     Each worker i pushes $\mathbf{g}_{i,t}$ to the master node.
7:     Master node computes the gradients average:
       $$\mathbf{g}_t = \frac{1}{M}\sum_{i=1}^M \mathbf{g}_{i,t}.$$
8:     Master node computes $\mathbf{v}_t = -\varepsilon sign(\mathbf{g}_t)$.
9:     Master node computes $\boldsymbol{\delta}_{t+1} = (1-\gamma_t)\boldsymbol{\delta}_t + \gamma_t\mathbf{v}_t$ and sends it to all nodes.
```

3.3. Distributed Stochastic Gradient Free Frank Wolfe

In this section we discuss the Distributed Stochastic Grandient Free FW in a distributed setup. In this setting we have that the $\boldsymbol{M}$ workers do not have a central coordinator, instead they exchange information in a peer-to-peer manner. The internode comunication network used by the workers is modeled as an undirected simple connective graph $\boldsymbol{G = (V, E)}$, with $\boldsymbol{V = \{1, \ldots, M\}}$ the set of nodes and $\boldsymbol{E}$ the set of comunication links. Each node comunicates and exchange information with its own neighbors, and given a node $\boldsymbol{n}$ we indicate with $\boldsymbol{\Omega_n = \{l \in V | (n, l) \in E\}}$ its neighborhood. Node $\boldsymbol{n}$ has degree $\boldsymbol{d_n = |\Omega_n|}$. Also, we use the $\boldsymbol{M \times M}$ adjacency matrix $\boldsymbol{A = [A_{ij}]}$ to describe the edges of the graph $\boldsymbol{G}$: we have $\boldsymbol{A_{ij} = 1}$ if $\boldsymbol{(i,j) \in E}$, $\boldsymbol{A_{ij} = 0}$ otherwise. We define the diagonal matrix $\boldsymbol{D = (d_1 \ldots d_M)}$ in order to compute the graph Laplacian $\boldsymbol{L = D - A}$. The Normalized Laplacian of $\boldsymbol{L}$

Algorithm 2 Decentralized Variance-Reduced Stochastic Gradient Free FW

```
Input Batches of images $\{\mathrm{x}_i\}_{i=1}^M$, labels $y$, Loss Function $F(\delta; y)$, number of queries $T$, number of workers $M$, image dimension $d$, tolerance $\varepsilon$, number of images $S_1$, subset of component functions $S_2$, total number of component functions $n$, period $q$.
Output Universal perturbation's history.
 1: Initialize $\delta_0 = 0$.
 2: for $t = 0, \dots, T-1$ do
 3:     Each worker $i$ computes:
 4:     if $mod(t, q) = 0$ then
 5:         Draw $S_1' = \frac{S_1 d}{M}$ samples for each dimension at each worker $i$ and compute its local gradient $\mathrm{e}_k^T \mathrm{g}_i(\delta_t) = \frac{1}{n}\sum_{j=1}^n \frac{F_{i,j}(\mathrm{x}_t+\eta \mathrm{e}_k)-F_{i,j}(\mathrm{x}_t)}{\eta}$ along each canonical basis vector $\mathrm{e}_k$.
 6:         Each worker updates $\mathrm{g}_{i,t} = \mathrm{g}_i(x)_t$.
 7:     else
 8:         Draw $S_2$ pairs of component functions and Gaussian random vectors $\{\mathrm{z}\}$ at each worker $i$ and update
            $\mathrm{g}_i(x)_t = \frac{1}{|S_2|}\sum_{j\in S_2} \frac{F_{i,j}(\mathrm{x}_t+\eta \mathrm{e}_k)-F_{i,j}(\mathrm{x}_t)}{\eta}\mathrm{z}$ - $\frac{F_{i,j}(\mathrm{x}_{t-1}+\eta \mathrm{e}_k)-F_{i,j}(\mathrm{x}_{t-1})}{\eta}\mathrm{z}$
 9:         Each worker updates
                $$\mathrm{g}_{i,t} = \mathrm{g}_i(\mathrm{x}_t) + \mathrm{g}_{i,t-1}.$$
10:     Each worker pushes $\mathrm{g}_{i,t}$ to the master node.
11:     Master node computes
                $$\mathrm{g}_t = \frac{1}{M}\sum_{i=1}^M \mathrm{g}_{i,t}.$$
12:     Master node computes $\mathrm{v}_t = -\varepsilon sign(\mathrm{g}_t)$.
13:     Master node computes $\delta_{t+1} = (1-\gamma_t)\delta_t + \gamma_t \mathrm{v}_t$ and sends it to all nodes.
```

is define to be the matrix

$$\mathcal{L}(u,v) = \begin{cases} 1 & \text{if } u = v \text{ and } d_v \neq 0, \\ -\frac{1}{\sqrt{d_u d_v}} & \text{if } u \text{ and } v \text{ are adjacent}, \\ 0 & \text{otherwise}. \end{cases}$$

We can write $\mathcal{L} = D^{-1/2} L D^{-1/2}$, with the convention that $D^{-1}(v,v) = 0$ if $d_v = 0$. The Laplacian $\mathcal{L}$ is used to compute the weghted matrix $W = \mathbb{1} - \mathcal{L}$.
On the contrary of the previous algorithm, we initialize the perturbation δ_0 as a $\mathbb{0}_{M\times d}$ matrix, rather than a vector, because in a distributed setting the nodes exchange information with their neighbors. This matrix simplifies the connections between them. At every time instant t in Algorithm 3, an agent i exchange its current iterate with its neighbors and average the iterates as follows:

$$\bar{\delta}_t^i \leftarrow \sum_{j=1}^M W_{ij}\delta_t^i.$$

After this, each worker i use I-RDSA to compute its local gradient estimation g_t^i, for $i = 1, \dots, M$, using $c_t = \frac{2\sqrt{m}}{d^{3/2}(t+8)^{1/3}}$. When all of them are calculated, they are collected as follow:

$$\bar{\mathrm{g}}_t^i \leftarrow \sum_{j=1}^M W_{ij}\mathrm{G}_t^j.$$

After the update, the algorithm computes the Frank-Wolfe step, calculating the new iterate δ_{t+1}^i using $\gamma_t = t^{-1/2}$ as suggested from Sahu's paper for non convex function.
At the end of all the iterations, the average of the perturbations is returned.

A distributed FW algorithm requires two round of communication: one for the iterate and one for the gradient estimate.
The performance of Algorithm 3 depends on how well the averaged iterates and the averaged gradient estimates are tracked across the network.

4. Experiments and results

The data come from the MNIST dataset, containing 60000 train images and 10000 test images from 10 almost balanced classes (arabic numerals). The following simulations generate a universal adversarial perturbation starting from a portion of the MNIST train 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 minimizing the accuracy.
We now present the results of the three algorithm previously introduced: (i) Decentralized Stochastic Gradient Free Frank Wolfe, (ii) Decentralized Variance-Reduced Stochastic Gradient Free Frank Wolfe, (iii) Distributed Stochastic Gradient Free Frank Wolfe.
Our interest is to find a perturbation that is able to misclassify the predictions of LeNet-5 on MNIST digit images, by making also the noise on the images imperceptible to human eyes.
In order to generate an adversarial example we have to find x' for a given input x such that the corresponding loss function $L(x', y)$ is maximized, while minimizing the ℓ_∞ norm related to it. This minimization is needed for finding the minimal perturbation that makes the digits unchanged. We set the δ noise within $\varepsilon = 0.25$ by using the ℓ_∞ norm.

Algorithm 3 Distributed Stochastic Gradient Free FW

```
Input Batches of images $\{\mathbf{x}_i\}_{i=1}^M$, labels $y$, Loss Function $F(x; y)$, number of queries $T$, number of workers $M$, image dimension $d$, tolerance $\varepsilon$, number of directions $m$, adjacency matrix $A$.
Output Average perturbations.
1: Initialize $\delta_0^i = 0$ for $i = 1, \dots, M$.
2: for $t = 1, \dots, T$ do
3:     Approximate the average iterate:
           $$\bar{\delta}_t^i \leftarrow \sum_{j=1}^M W_{ij}\delta_t^i$$
4:     At each node $i$, employ I-RDSA to estimate the gradient $\mathbf{g}_t^i = \sum_{j=1}^d \frac{F(\bar{\mathbf{x}}_t^i + c_t \mathbf{e}_j) - F(\bar{\mathbf{x}}_t^i)}{c_t}\mathbf{e}_j$.
5:     Approximate the average gradient:
           $$\mathbf{G}_t^i = \bar{\mathbf{g}}_{t-1}^i + \mathbf{g}_t^i - \mathbf{g}_{t-1}^i$$
           $$\bar{\mathbf{g}}_t^i \leftarrow \sum_{j=1}^M W_{ij}\mathbf{G}_t^j$$
6:     Each worker computes $\mathbf{v}_t = -\varepsilon sign(\mathbf{g}_t)$.
7:     Frank-Wolfe Step: update
           $$\delta_{t+1}^i \leftarrow (1 - \gamma_t)\bar{\mathbf{x}}_t^i + \gamma_t \mathbf{v}_t^i$$
       for all worker $i = 1, \dots, M$.
```

We performed the experiments on the MNIST dataset with the aim to find an universal adversarial perturbation, and we defined a pretrained LeNet-5 model by using Keras/Tensorflow to test our algorithms and demonstrate the efficiency of the attacks.

Non so se ha senso metterlo qua, magari meglio discutere anche con Chiara se ha senso dire che comunque tutti gli algoritmi decentralize e distributed sono stati implementati in senquential mode invece che usare un vera e propria architettura distribuita. Dobbiamo far notare che comunque il codice e‘ stato implementato definendo i metodi in modo tale che sia scalabile facilmente alla modalita distribuita usando per esempio la libreria Ray

Since the discussed algorithms have a decentralized or a distributed "part", we implemented them in a sequential manner but in such way that all the algorithms could be scaled and changed easily to a distributed or decentralized architecture. Indeed, all the methods have been implemented by dividing the workers and master nodes behaviour in different separate methods, and the extension to a distributed architecture could be reached by configuring a framework/library like Ray[1] or PySpark[2] for defining a distributed computing application.

4.1. Decentralized Stochastic Gradient Free Frank Wolfe

Usiamo il passato ora, siccome e‘ un esperimento svolto
To study the performance of Algorithm 1 we used the 10000 images in the MNIST test set, after normalizing them.
We split the digit images by giving 10 samples of each class to our 10 workers. In such way, we make each worker holding a hundred images. After tuning the hyperparameter m of the number of direction, we discovered that $m = 15$ was the good compromise between the computation time and the overall results. For each image we estimated its gradient using 20, 50 and 100 queries.

Accuracy error achieved, during the training of the model, the we need to compare the result achieved by testing the model on the whole test set.
Notice that the drop of the accuracy, hence the accuracy is discovered already at the 10 epoch/iteration of the algorithms. Comparing the three algorithm is almost the faster one, the competing algorithm in terms of speed convergence of the noise is the distributed.
presence of the pattern in the noise when the accuracy is minimized, more is minimized more the noise the pattern is visible in this algorithms. INstead in the other algorithms the pattern is less visibile e secondo silvia per esempio il variance reduced non presenta un pattern appunto perche il noise e‘ piu distribuito siccome per la proprieta della varianza considerata nell'algoritmo. Un piccolo confronto con il guassian noise come fatto nel jupyter sarebbe il top. In

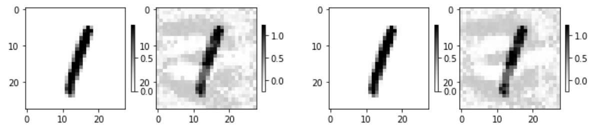

Figure 1. Decentralized FW Perturbated Images: no clue on the parameters

Figure 1 we can see an example of the perturbed images.

4.2. Decentralized Variance-Reduced Stochastic Gradient Free Frank Wolfe

For the Variance-Reduced FW algorithm we consider 5 workers and with 800 different images each, i.e. 160 images per digit. We set the number of queries to 20 and the number of component functions $S_2 = 3$. We then run the Algorithm 2 for $q = 5, 7, 9$ and $n = 5, 10$. The choice for the values of q is because of the different number of calling to the KWSA, while the n parameter identify the different number of component function.

[1]Ray: https://github.com/ray-project/ray
[2]PySpark: https://spark.apache.org/docs/latest/api/python/

To quantify the performance of Algorithm 2, it can be used the Frank-Wolfe duality gap, which is an upper bound on the primal suboptimality $f(x_t) - f(x^*)$, define as

$$\mathcal{G} = \max_{v \in C} < F(x), x - v >$$

where C is the associated constraint. In addition, it can also be used as a stopping criterion for FW algorithms. From the theory we know that the major improvement of the variance reduction scheme is in terms of the fradient tracking performance, thath is, with $S_2 = (2d + 9)\sqrt{n}/n_0$, $q = n_0\sqrt{n}/6$, where $n_0 \in [1, \sqrt{n}/6)$. For the data that we have this tecnique for measuring the performances is unfeasible.

DA SPIEGARE MEGLIO, QUESTA PARTE NON L'AVEVO CAPITA MOLTO BENE :(

4.3. Distributed Stochastic Gradient Free Frank Wolfe

To test the performance of Algorithm 3 we used 10 workers and an adjacecy matrix A given by

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

We can notice that the diagonal is of ones, this because each node is connected to itself. Our network is composed of 10 nodes and the connectivity of the graph can be know by computing $||W - J||$, where $J = 11^T/10$ and 11^T represent a matrix with all entries set to 1. In our case we have a connectivity value of 0.438. We use 15 directions and test the algorithm for 20, 50 and 100 queries.

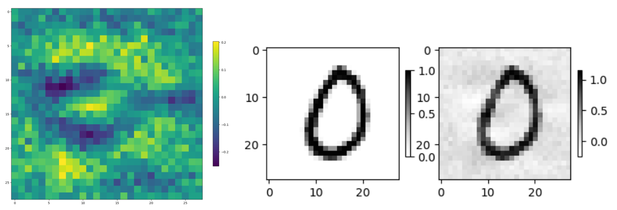

Figure 2. Perturbation and perturbed image of the distributed FW with T=100 and m=15.

In Figure 2 we can see an example of the perturbations.

4.4. Comparison between the perturbations

5. Conclusions