Zeroth-Order Methods for Adversarial Machine Learning

Optimization for Data Science Project

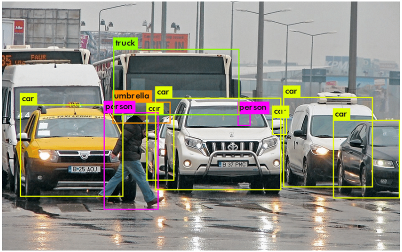
Federico Zanotti, Lorenzo Corrado

September 17, 2021

Context

Deep Neural Networks (DNNs) represents a class of models that have been the revolution in AI:

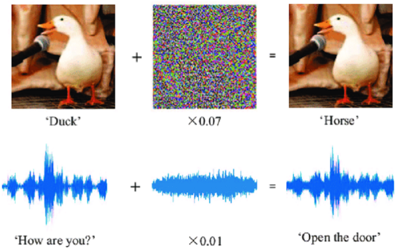
- Image recognition
- Speech recognition
- Pattern analysis



Given the wide use of this type of models, security issues have emerged.

Adversarial Attacks

- It has been shown that this class of models are vulnerable to adversarial examples
- An adversarial example represents slightly modified data that lead to incorrect classification by the model

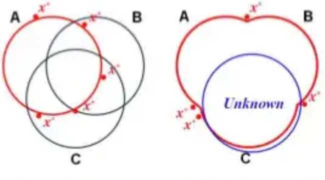


- The perturbation is carefully crafted to fool the target model and it is almost imperceptible for a human

Attacks Setting

- **White-box:** the attacker knows all the information about the target model
  - Model architecture ✓
  - Parameters ✓
  - Inputs and Outputs ✓
  - Query the target model ✓
  - Gradient ✓
- **Black-box:** the attacker does not known information about the target model
  - Inputs and Outputs ✓
  - Query the target model ✓

- In the black-box setting, an adversarial example modified over a single target model can be effective for other models



(a) A single target model (b) Multiple target models

- It is necessary to study them to understand how to build more robust models

Types of Attacks

- **Targeted attacks:** an attacker tries to fool the DNN to predict a particular class

$$argmax\ P(\mathbf{y}|\tilde{\mathbf{x}}) = \mathbf{y}_{target}$$

- **Untargeted attacks:** an attacker tries to fool the DNN to get any incorrect class

$$argmax\ P(\mathbf{y}|\widetilde{\mathbf{x}}) \neq \mathbf{y}_{true}$$

- In this project we will focus on the problem of generation of adversarial examples in the black-box setting and we will optimize an objective function to make untargeted attacks
- Now let's see how the problem can be formalized from a mathematical point of view

The General Problem

In general, we can formalize the problem as a constrained finite-sum minimization problem:

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

Where:

- $\Omega \in \mathbb{R}^d$ denotes a closed convex feasible set
- Each component function f_i is smooth and non-convex
- $n \in \mathbb{R}$ denotes the number of components function

- Frank-Wolfe algorithm is one of the oldest methods for non-linear constrained optimization like the problem above
- Contrary to another popular method like the projected gradient descent, this algorithm uses a routine that solves a linear sub-problem to find a feasible solution in Ω
- This routine is the so called linear minimization oracle (LMO):

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

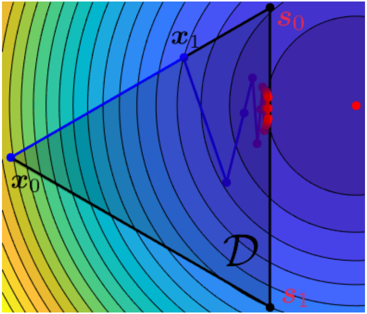


Figure 1: Example on a 2D problem.

- There are many different variants of this algorithm, but many of them are based on the availability of the gradient ∇f_i
- Given our project setting we can use only zeroth-order methods to tackle the problem, so we need some estimators to approximate the gradient of the objective function
- The zeroth-order methods in general are useful when:
  - Gradient information is infeasible to obtain
  - Gradient information is difficult to compute

In this project we will present:

- Zeroth-Order Stochastic Conditional Gradient (ZSCG), proposed by Balasubramian et al., (2018)
- Faster Zeroth-Order Frank-Wolfe (FZFW), proposed by Gao et al., (2020)
- Faster Zeroth-Order Conditional Gradient Sliding Method (FZSCG), proposed by Gao et al., (2020)

Assumptions

We will list some standard useful assumptions for the definition of the zeroth-order gradient estimators and for the convergence analysis:

1. The component function f_i $(i \in [n])$ is L-smooth:

$$||\nabla f_i(\mathbf{x}) - \nabla f_i(\mathbf{y})|| \leq L||\mathbf{x} - \mathbf{y}||, \quad \forall \mathbf{x}, \mathbf{y} \in \Omega \tag{3}$$

2. The diameter of the feasible set Ω is D
3. Assume that the variance of the stochastic gradient $\nabla f_i(\mathbf{x})$ $(i \in [n])$ is bounded as:

$$\frac{1}{n}\sum_{i=1}^{n} ||\nabla f_i(\mathbf{x}) - \nabla F(\mathbf{x})||^2 \leq \sigma^2 \tag{4}$$

Where $\sigma > 0$

Convergence criteria

- Frank-Wolfe gap (for ZSCG and FZFW):

$$\mathcal{G} = \max_{\mathbf{u} \in \Omega} \langle \mathbf{u} - \mathbf{x}, -\nabla F(\mathbf{x}) \rangle. \tag{5}$$

- Gradient mapping (for FZCGS):

$$\mathcal{G}(\mathbf{x}, \nabla F(\mathbf{x}), \gamma) = \frac{1}{\gamma}(\mathbf{x} - \psi(\mathbf{x}, \nabla F(\mathbf{x}), \gamma)), \tag{6}$$

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

$$\psi(\mathbf{x}, \nabla F(\mathbf{x}), \gamma) = \operatorname*{argmin}_{\mathbf{y} \in \Omega} \langle \nabla F(\mathbf{x}), \mathbf{y} \rangle + \frac{1}{2\gamma} ||\mathbf{y} - \mathbf{x}||^2 \rangle \tag{7}$$

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

Oracle Model

To compare the iteration complexity of different algorithms:

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

- **Averaged (gaussian) random gradient** estimator:

$$\hat{\nabla} f_i(\mathbf{x}) = (d/\nu q) \sum_{j=1}^{q} [f_i(\mathbf{x} + \nu \mathbf{u}_{i,j}) - f_i(\mathbf{x})] \mathbf{u}_{i,j} \tag{8}$$

Where $\nu > 0$ is a smoothing parameter, $\mathbf{u}_{i,j} \sim N(\mathbf{0}, \mathbf{I})$

- **Coordinate-wise gradient** estimator:

$$\hat{\nabla} f_i(\mathbf{x}) = \sum_{j=1}^{d} \frac{f_i(\mathbf{x} + \mu \mathbf{e}_j) - f_i(\mathbf{x} - \mu \mathbf{e}_j)}{2\mu} \mathbf{e}_j \tag{9}$$

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

Smoothing parameter ν

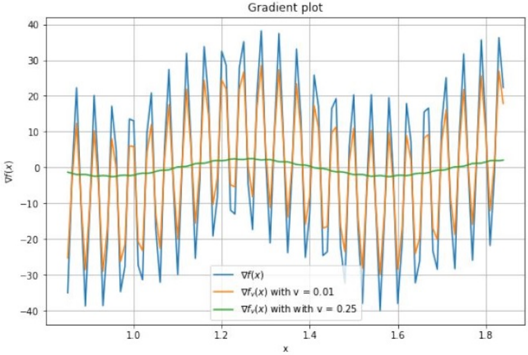


Figure 2: Gradient and approximated gradient on a toy example of f_ν.

ZSCG Scheme

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

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

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

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

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

2. Compute

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

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

end for

return z_k

Convergence Rate for ZSCG

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

$$\nu = \sqrt{\frac{2B_{L\sigma}}{K(d+3)^3}},\ \alpha_k = \frac{1}{\sqrt{K}},\ m_k = 2B_{L\sigma}(d+5)K,\ \forall k \geq 1 \quad (10)$$

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

$$\mathbb{E}[g_\Omega^R] \leq \frac{f(z_0) - f^* + LD_\Omega^2 + 2\sqrt{B^2+\sigma^2}}{\sqrt{K}} \quad (11)$$

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

Convergence Rate for ZSCG

With the same setting of Theorem 1 we have:

- Total number of calls to the zeroth-order stochastic oracle:

$$O\left(\frac{d}{\epsilon^4}\right) \tag{12}$$

- Total number of linear suproblems:

$$O\left(\frac{1}{\epsilon^2}\right) \tag{13}$$

FZFW and FZCGS

- *Can Stochastic Zeroth-Order Frank-Wolfe Method Converge Faster for Non-Convex Problems?*

FZFW Scheme

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

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

Convergence Rate for FZFW

Theorem 2. *Under the same assumptions made before, if the parameters are chosen as:*

$$|S_1| = n, q = |S_2| = \sqrt{n}, \gamma_k = \gamma = \frac{1}{D\sqrt{K}}, \mu = \frac{1}{\sqrt{dK}} \tag{14}$$

Then FZFW satisfies:

$$\mathbb{E}[\mathcal{G}(\mathbf{x}_\alpha)] \leq \frac{D(F(\mathbf{x}_0) - F(\mathbf{x}_*) + 11L)}{\sqrt{K}} \tag{15}$$

Convergence Rate for FZFW

With the same setting of Theorem 2 we have:

- Total number of calls to the zeroth-order oracle:

$$O\left(\frac{n^{\frac{1}{2}}d}{\epsilon^2}\right) \tag{16}$$

- Total number of linear sub-problems:

$$O\left(\frac{1}{\epsilon^2}\right) \tag{17}$$

FZCGS Scheme

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

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

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

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

FZCGS Condg

- If we define

$$\phi(y; x, \bigtriangledown F(x), \gamma) = min_{y \epsilon \Omega} \langle \bigtriangledown F(x), y \rangle + \frac{1}{2\gamma} ||y - x||^2 \quad (18)$$

- Then step 2 in Condg procedure is equivalent to optimize

$$max_{x \epsilon \Omega} \langle \phi^{'}(u_t; u, g, \gamma), u_t - x \rangle \quad (19)$$

because

$$\phi^{'}(u_t; u, g, \gamma) = g + \frac{1}{\gamma}(u_t - u) \quad (20)$$

And this is the Frank Wolfe Gap and when it is smaller than tolerance η, u_t is returned

Convergence Rate for FZCGS

Theorem 3. *Under the assumptions made before, if the parameters are chosen as:*

$$|S_1| = n, q = |S_2| = \sqrt{n}, \mu = \frac{1}{\sqrt{dK}}, \gamma_k = \gamma = \frac{1}{3L}, \eta_k = \eta = \frac{1}{K} \tag{21}$$

Then FZCSG satisfies:

$$\mathbb{E}[||\mathcal{G}(\boldsymbol{x}_\alpha, \nabla F(x_\alpha, \gamma))||^2] \leq \frac{\left(3(F(\boldsymbol{x}_0) - F(\boldsymbol{x}_*) + 1) + 7L\right)6L}{K} \tag{22}$$

Convergence Rate for FZCGS

With the same setting of Theorem 3 we have:

- Total number of calls to the zeroth-order oracle:

$$O\left(\frac{n^{\frac{1}{2}}d}{\epsilon}\right) \tag{23}$$

- Total number of linear suproblems:

$$O\left(\frac{1}{\epsilon^2}\right) \tag{24}$$

Convergence Rate Summary

Zeroth-Order	FQO
ZSCG	$O\left(\frac{d}{\epsilon^4}\right)$
FZFW	$O\left(\frac{n^{\frac{1}{2}} d}{\epsilon^2}\right)$
FZCGS	$O\left(\frac{n^{\frac{1}{2}} d}{\epsilon}\right)$

Table 1: FQO of the three different algorithms.

Our optimization problem

Let's assume to have:

- Dataset: $\{(\mathbf{x}_i, \mathbf{y}_i) : (\mathbf{x}_i \in \mathbb{R}^d, \mathbf{y}_i \in \{0, 1, \dots, c\}\}_{i=1}^n$
- Black-box DNN $f : \mathbb{R}^d \to \mathbb{R}$

The task is to find the universal adversarial perturbation $\delta \in \mathbb{R}^d$ for sample $\mathbf{x}_i$ s.t. the DNN makes the incorrect prediction $\hat{y}_i \neq y_i$.

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

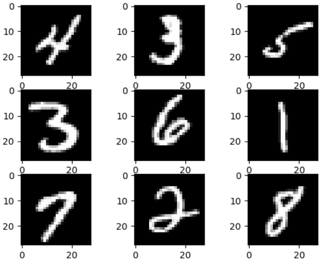
Where:

- $f(\mathbf{x}) = [f_1(\mathbf{x}), \dots, f_c(\mathbf{x})]$, denotes the output of the last layer before softmax
- s, represents the magnitude of the distortion that can be applied to the images, in our case $s = 0.1$

Dataset

The dataset we used is MNIST, a popular benchmark dataset for learning, classification and computer vision systems

- 60,000 images of handwritten digits (0-9)
- 28x28 pixels in gray-scale
- Normalization in the range [0,1]
- Flattened images



Setup

- To implement our code we used the Anaconda's standard library and and the library Keras
- To run our code we used Google Colab Pro that provides Intel Xeon E5-2699v4 CPU and Nvidia Tesla T4 GPU

Model

We used the same pretrained DNN used in (Gao et al., 2020):

Layer Type	Size
Reshape Layer[1]	-
Convolution + ReLU	3x3x32
Convolution + ReLU	3x3x32
Max Pooling	2x2
Convolution + ReLU	3x3x64
Convolution + ReLU	3x3x64
Max Pooling	2x2
Fully Connected + ReLU	200
Fully Connected + ReLU	200
Softmax	10

Hyperparameter	Value
Learning Rate	0.1
Momentum	0.9
Delay rate	-
Batch Size	128
Epoch	50

[1] Not present in the original DNN architecture.

Implementation Details

1. **Derivation of LMO**

The problem in Eq.27 can be rewritten as:

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

Given that, the closed-form solution for the LMO in our case is:

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

Where $\mathbf{g}_t$ denotes the gradient of the objective function.

Implementation Details

2. **Perturbation clipping**

- Clipping represents the operation where we rescale, at each iteration, the pixels of the new perturbed images in the range between [0,1]
- This is very important if we wanted to conduct a real adversarial attack
- This decreases the strength of the attack, we will present the results both when this procedure is applied and when it is not applied

Implementation Details

3. **Stopping criterion**

- We have implemented a stopping criterion based on how badly the DNN predicts the new perturbed images
- If the DNN fails to classify a certain percentage of the images, the algorithms terminate prematurely

Implementation Details

4. **Averaged gaussian random gradient implementation**

- We have implemented a variant of the avg-random gradient estimator in which we generate q random directions for each component function f_i
- We did this to overcome the computation complexity problem of using ZSCG with large batches of examples
- If we set $q = 1$ we have the usual avg-random gradient estimator[2]

[2] Further details in: `https://arxiv.org/pdf/1805.10367.pdf`

ZSCG Gradient Estimator

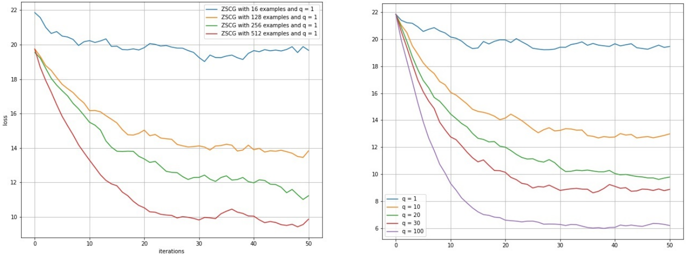


Figure 3: Examples of ZSCG varying batch sizes (left) and q (right).

Clip vs No CLip

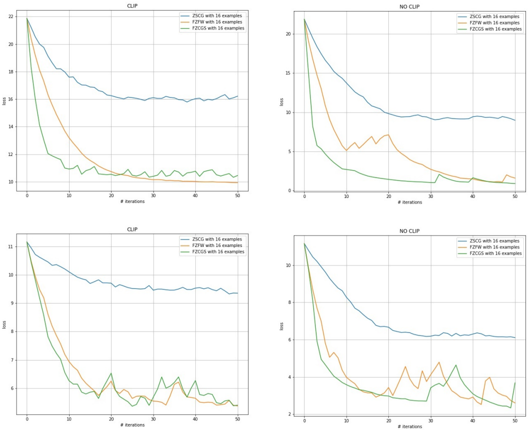
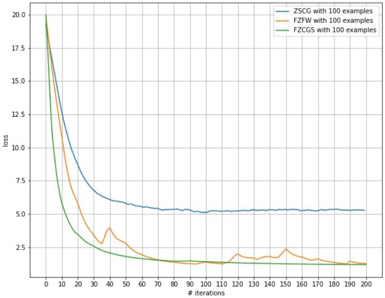


Figure 4: Comparison with $n = 16$ from class 4 (up) and comparison with $n = 16$ from different classes (down), $q = 30$.

Results



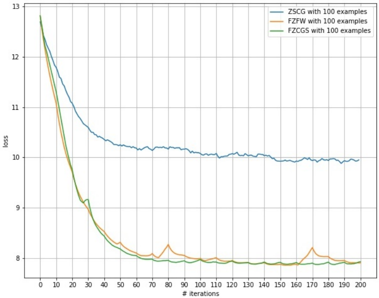


Figure 5: Comparison with $n = 100$ of class 4 (left) and different classes (right), $q = 30$ and no clip.

More consistent result

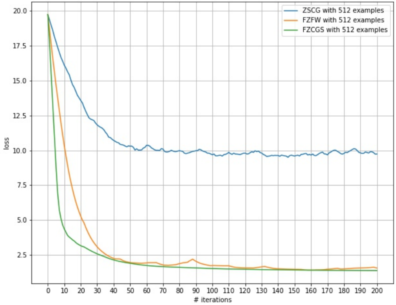


Figure 6: Comparison with $n = 512$ of class 4, $q = 1$ and no clip.

- An accurate gradient estimator is of paramount importance in zero-order methods
- We can confirm that ZSCG is the slower algorithm and instead the FZSCG algorithm is the fastest as regards the convergence speed in almost all the cases
- While the FZFW and FZCGS algorithms manage to perform well even when the batch of images to be attacked is small, the ZSCG algorithm turns out to have poor performance
- The magnitude of the distortion $s = 0.1$ is pretty low to attack a well trained model like the one used in this project

Thanks!