Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks

Damiano Clementel, Paolo Fantuz, Francesco Grimaldi, Domenico Solimini

University of Padova - Dipartimento di Matematica

DATA SCIENCE
UNIVERSITY OF PADOVA

Topics

Overview

- Security issues with Deep Neural Networks
- Deep Neural Networks involved
- Types of adversarial attacks
- Knowledge of the attacked Deep Neural Network

Black-Box vs White-Box

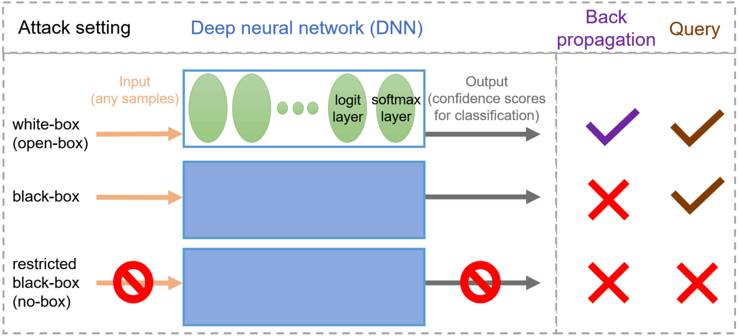


Figure: Types of white-box and black-box attack settings

Adversarial attack as optimization problem

By defining decision function for model $\mathcal{M}$ as $F_{\mathcal{M}}(x) : \chi \rightarrow [0;1]^m$ and constraint $C(x) = \arg\min_{i \in \{1,\dots,m\}} F_{\mathcal{M}}(x)$, we get Carlini and Wagner formulation:

$$\begin{aligned} \min \quad & ||\delta|| \\ \text{sub.to} \quad & C(x+\delta) = t \\ & (x+\delta) \in [0,1]^n \end{aligned}$$

Loss functions

New loss function $f : \chi \rightarrow \mathbb{R}$ introduced better suited for optimization algorithms:

- Targeted attack
  $f_t(x) = \max\{\max_{i \neq t} \log[F_{\mathcal{M}}(x)]_i \; - \; \log[F_{\mathcal{M}}(x)]_t \; , \; -k\}$
- Untargeted attack
  $f(x) = \max\{\log[F_{\mathcal{M}}(x)]_{t_0} \; - \; \max_{i \neq t_0} \log[F_{\mathcal{M}}(x)]_i \; , \; -k\}$

Constraining perturbation set

1. Add a regularization term to the loss function that takes into account the norm of δ

$$\begin{aligned} \min \quad & \|\delta\| + c \cdot f(x+\delta) \\ \text{sub.to} \quad & (x+\delta) \in [0,1]^n \end{aligned}$$

2. Modify the feasible region $\chi = [0;\ 1]^n$ by doing an intersection with the L_p Ball of radius set equal to a parameter ϵ and centered in the original input $\mathcal{B}_\epsilon^{(p)}(x)$

$$\begin{aligned} \min \quad & f(x+\delta) \\ \text{sub.to} \quad & (x+\delta) \in [0,1]^n \cap \mathcal{B}_\epsilon^{(p)}(x) \end{aligned}$$

General Idea

- We want to solve the constrained optimization problem overcoming the hard computational requirements it presents.
- Under regularity hypothesis, the gradient of a function can be approximated using zero-th order information.
- The loss function may be highly irregular but can be smoothed by considering

$$f_\nu(x) = \mathbb{E}_u[f(x + \nu u)]$$

for some $\nu \in (0;\ \infty)$ and $u \sim \mathcal{N}(0, I_d)$.

Gradient Smoothing Approach

We want $f_\nu(x)$ to be an accurate approximation of $f(x)$ and to show higher regularity.

It can be proved that, for any $\nu > 0$, f_ν is differentiable and its gradient is Lipschitz-continuous.

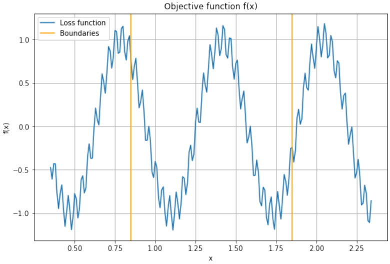
In [Nesterov2015] it is also shown that if f is Lispschitz-continous with constant L, then

$$\|f_\nu(x) - f(x)\| \leq \frac{\nu^2}{2} Ln$$

$$\|\nabla f_\nu(x) - \nabla f(x)\| \leq \frac{\nu^2}{2} L(n+3)^{3/2}$$

The Gaussian smoothing parameter ν. Pt1

Depending on the choice of ν, f_ν can approximate as tightly as wanted the function f or can show a much more regular behaviour.



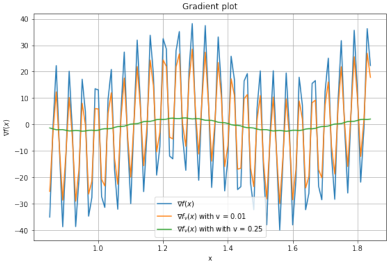


Figure: Left: the loss function; right: the gradient of $f(x)$ and the approximated gradient of $f_\nu(x)$ at different value of ν

Zero-th Order Stochastic Oracle - Regularity Assumptions

For any $x \in \mathbb{R}^d$, the zero-th order stochastic oracle outputs an estimator $F(x, \xi)$ of $f(x)$. We assume that:

1. $F(x, \xi)$ is an unbiased estimator of $f(x)$

$$\mathbb{E}_\xi[F(x, \xi)] = f(x)$$
$$\mathbb{E}_\xi[\nabla_x F(x, \xi)] = \nabla f(x)$$
$$\mathbb{E}_\xi[\|\nabla_x F(x, \xi) - \nabla f(x)\|^2] \leq \sigma^2$$

2. Function F has LCG with constant L, almost surely for any ξ.
3. The feasible set χ is bounded such that $\max_{x,y \in \chi} \|x - y\| \leq D_\chi$ for some $D_\chi \geq 0$.

Random Gradient-Free Oracles

Given $u \sim \mathcal{N}(0, I_d)$, the *stochastic gradient* of f is:

$$G_\nu(x, \xi, u) = \frac{F(x + \nu u, \xi) - F(x, \xi)}{\nu} u$$

where F is a zero-th order stochastic oracle of f. In particular we can use $F(x, \xi) = f(x)$.

Thanks to the *Stein Identity*, under Assumption 1, it can be shown that

$$\nabla f_\nu(x) = \mathbb{E}_{u,\xi}[G_\nu(x, \xi, u)].$$

By increasing m_k the variance of the estimate can be reduced as:

$$\mathbb{E}_{u,\xi}[\|G_\nu^k - \nabla f_\nu(x_{k-1})\|^2] \leq \frac{1}{m_k} \mathbb{E}_\xi[\|G_\nu^{k,j} - \nabla f_\nu(x_{k-1})\|^2].$$

ZOO

The problem is formulated as:

$$\min_{x \in [0,1]^p} \|x - x_0\|_2^2 + c \cdot f(x, t). \tag{1}$$

An approximated gradient is computed as:

$$\hat{g}_i := \frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + he_i) - f(x - he_i)}{2h}, \tag{2}$$

where h is a small constant and e_i is a standard basis vector with only the i-th component set to 1.

$$\hat{h}_i := \frac{\partial^2 (x)}{\partial x_{ii}^2} \approx \frac{f(x + he_i) - 2f(x) + f(x - he_i)}{h^2}. \tag{3}$$

ZOO - Main algorithm (ADAM)

Algorithm 1: ZOO with ADAM

Input: η, β_1, β_2, ϵ

Initialize: $M = \mathbf{0}$, $T = \mathbf{0}$, $v = \mathbf{0}$, $M, T, v \in \mathbb{R}^p$

while *not converged* **do**

Randomly pick coordinate $i \in \{1, ..., p\}$;

Estimate $\hat{g}_i$;

$T_i \leftarrow T_i + 1$;

$M_i \leftarrow \beta_1 M_i + (1 - \beta_1)\hat{g}_i$;

$v_i \leftarrow \beta_2 v_i + (1 - \beta_2)\hat{g}_i{}^2$;

$\hat{M}_i = \frac{M_i}{(1-\beta_1^{T_i})}$; $\hat{v}_i = \frac{v_i}{(1-\beta_2^{T_i})}$;

$\delta^* = -\eta \frac{\hat{M}_i}{\sqrt{\hat{v}_i} + \epsilon}$;

$x_i \leftarrow x_i + \delta^*$;

end

ZOO - Main algorithm (Newton)

Algorithm 2: ZOO with Newton

Input: η

while *not converged* **do**

    Randomly pick B coordinates $i \in \{1, ..., p\}$; Estimate $\hat{g_B}$ and $\hat{h_B}$; **if** $\hat{h_B} < 0$ **then**

        $\delta^* \leftarrow -\eta \hat{g_B}$;

    else

        $\delta^* \leftarrow -\eta \frac{\hat{g_B}}{\hat{h_B}}$;

    end

    $x_B \leftarrow x_B + \delta^*$;

end

High Dimensional ZOO

For large images we reduce the attack space by introducing a dimension reduction transformation $D(y)$ where $y \in \mathbb{R}^m$, range$(D) \in \mathbb{R}^n$, $m < n$ and using $D(y)$ to replace δ:

$$\min_{x_0+D(y)\in[0,1]^p} \|D(y)\|_2^2 + c \cdot f(x_0 + D(y), t).$$

To further improve the attack three different techniques are also introduced:

- Hierarchical Attack;
- Importance Sampling;
- ADAM State Reset.

ZOO - Convergence rate

Considering a non-smooth convex function f with Lipschitz continuous gradient with constant L, Nesterov proved:

$$\mathbb{E}_u\left(||\hat{g}_h(x)||^2\right) \leq \frac{h^2}{8}L^2(n+6)^3 + 2(n+4)||\nabla f(x)||^2$$

Convergence rate for a zeroth-order stochastic gradient method applied to non-smooth

- convex problems: $O\left(\frac{n}{\sqrt{k}}\right)$
- non-convex problems: $O\left(\sqrt{\frac{n}{k}}\right)$

Black-Box Frank-Wolfe

- Variant of Frank-Wolfe optimization algorithm
- Uses momentum term
- Projection free algorithm
- Guaranteed $O(1/\sqrt{T})$ with T number of epochs
- Guaranteed linear query complexity in data dimensions d

Black-Box Frank-Wolfe - Main algorithm

Algorithm 3: Black-Box Frank-Wolfe algorithm

Input: x_0, T number of iterations, $\{\gamma_t\}$ list of step sizes, b number of gradient estimation iterations, δ gradient smoothing parameter

Initialize: $m_t = \text{GRAD\_EST}(x_0, b, \delta)$

for $t = 0, ..., T-1$ **do**

$q_t = \text{GRAD\_EST}(x_t, b, \delta)$

$m_t = \beta \cdot m_{t-1} + (1-\beta) \cdot q_t$

$v_t = \arg\min_{x \in \chi} \langle x, m_t \rangle$ // LMO

$d_t = v_t - x_t$

$x_{t+1} = x_t + \gamma_t d_t$

end

Black-Box Frank-Wolfe - Gradient estimation

Algorithm 4: Gradient estimation algorithm

Input: x input variable (e.g. linearized image), number of gradient estimation iterations b, δ gradient smoothing parameters

Initialize: $q = 0$

for $i = 1, ..., b$ **do**

Option 1: sample u_i uniformly from Euclidean unit sphere with $\|u_i\|_2 = 1$

$$q = q + \frac{d}{2\delta b}(f(x + \delta u_i) - f(x - \delta u_i))u_i$$

Option 2: sample u_i uniformly from standard multivariate Gaussian distribution $N(0, I)$

$$q = q + \frac{1}{2\delta b}(f(x + \delta u_i) - f(x - \delta u_i))u_i$$

end

Return: q

Classic ZSCG - Main idea

The vanilla Zero-order Stochastic Conditional Gradient it's a variation of the well known *Frank-Wolfe* method. The difference in here, is that we only have an approximation of the gradient computed with the method described in the previous section.

The main steps are:

- Compute the approximated gradient $G_\nu^k(x, u)$
- Solve the linear problem $z_k = \arg\min_{u \in \chi} \langle G_\nu^k, u \rangle$
- Set x_k as a convex combination of x_{k-1} and z_k

Classic ZSCG - Algorithm

Algorithm 5: Vanilla version of ZSCG

Input: $x_0 \in \chi$, $\nu > 0$, α_k, m_k, $N \geq 1$, probability distribution $P_R(\cdot)$ over $\{1, ..., N\}$.

for $k = 1, ..., N$ **do**

1. Generate $u_k = [u_{k,1}, ..., u_{k,m_k}]$, where $u_k \sim \mathcal{N}(0, I_d)$;

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

2. Solve $z_k = \arg\min_{u \in \chi} \langle G_v^k, u \rangle$
3. Set $x_k = (1 - \alpha_k) x_{k-1} + \alpha_k z_k$

end

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

Classic ZSCG - The Frank-Wolfe Gap

In order to provide convergence analysis, since f is non-convex we can use the Frank-Wolfe Gap defined as:

$$g_\chi^k \equiv g_\chi(x_{k-1}) := \langle \nabla f(x_{k-1}), x_{k-1} - \hat{z}_k \rangle$$

where $\hat{z}_k = \arg\min_{u \in \chi} \langle \nabla f(x_{k-1}), u \rangle$.

This criteria is useful since $\langle \nabla f(x_{k-1}), x_{k-1} - u \rangle \leq g_\chi(x_{k-1})$, $\forall u \in \chi$ which implies that we can minimize g_χ^k to obtain a stationary point for our optimization problem.

Classic ZSCG - Convergence rate 1

Using the *Frank-Wolfe Gap* and by using the following parameters $\forall k \geq 1$:

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

where $B_{L\sigma} \geq \max\left\{\frac{\sqrt{B^2+\sigma^2}}{L}, 1\right\}$ and $B \geq \|\nabla f(x)\|$[1], $\forall x \in \chi$.

it can be showed that:

$$\mathbb{E}[g_\chi^R] \leq \frac{f(x_0) - f^* + LD^2 + 2\sqrt{B^2+\sigma^2}}{\sqrt{N}}$$

[1] $||\nabla f(x)|| - ||\nabla f(x^*)|| \leq ||\nabla f(x) - \nabla f(x^*)|| \leq L||x - x^*|| \leq DL = B - ||\nabla f(x^*)||$

Classic ZSCG - Convergence rate 2

Using the fact that $\mathbb{E}[g_{\chi}^{R}] \leq \frac{c}{\sqrt{N}} \leq \epsilon$, which implies $N \geq \frac{c}{\epsilon^2}$, and that at each step we call the zeroth-order stochastic oracle $2B_{L\sigma}(d+5)N$ times, we get that the number of calls to the zeroth-order stochastic oracle and linear subproblems required to be solved to find an ϵ-stationary points of our problem are, respectively, bounded by:

$$O\left(\frac{d}{\epsilon^4}\right), O\left(\frac{1}{\epsilon^2}\right)$$

Classic ZSCG for Adversarial Attacks

Algorithm 6: Classic ZSCG for Adversarial Attacks

Input: $x_0 \in \chi$, $\nu > 0$, α_k, m_k, $N \geq 1$, S be the set of accepted solutions

Set $q = 0$, $k = 0$

while $q = 0$ *and* $k \leq N$ **do**

1. Increment k
2. Generate $u_k = [u_{k,1}, ..., u_{k,m_k}]$, where $u_k \sim \mathcal{N}(0, I_d)$;

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

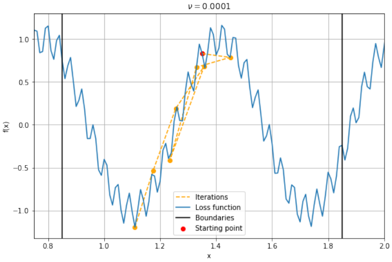
3. Solve $z_k = \arg\min_{u \in \chi} \langle G_v^k, u \rangle$
4. Set $x_k = (1 - \alpha_k) x_{k-1} + \alpha_k z_k$
5. Check stopping criterion: $q = SC\Big(x_k, F_M(\cdot, w), S\Big)$

end

Return x_k

Classic ZSCG - The Gaussian smoothing parameter ν.

As explained in the Section *Zero-th Order Methods* an important parameter is ν which can help to regularize a very unstructured function and increase the convergence rate.



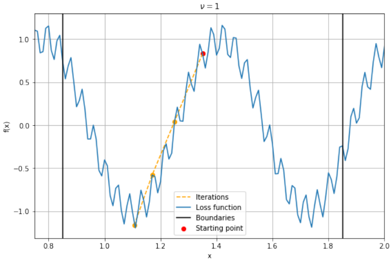


Figure: Right: the algorithm has an hard time to converge with a small value of ν; left: A bigger ν can be useful to avoid local minima around your x

Classic ZSCG - The parameter α

Another important parameter is α which regulates the convex combination between the result of the linear programming problem z_k and the old x_{k-1}. Small α can lead to be stuck in the very first local minima, while big α can mess up with the convergence rate and make you bounce between the boundaries of the feasible set χ

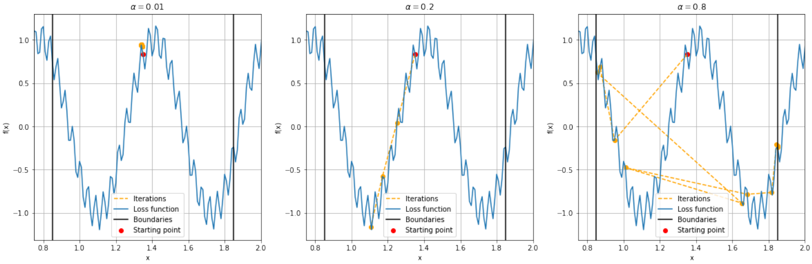


Figure: Different values of α (0.01, 0.2, 0.8) lead to very different results

Classic ZSCG - The parameter m

The performances can also be very influenced by the number of random Gaussian vector $u \sim \mathcal{N}(0, I_d)$; in fact we can increase the precision of the approximation of the gradient by increasing the number of function evaluations used to estimate it.

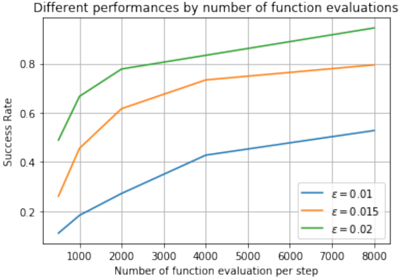


Figure: We can see that in all the attacks type we can increase the performance by increasing the parameter m

Inexact ZSCG - Main Idea

The main idea of this ZSCG variation is to update the current estimate x_k as

$$x_{k+1} = P_{\chi}(x, \xi, \gamma) := \arg\min_{u \in \chi} \left\{ \langle g_k, u \rangle + \frac{\gamma}{2} \|u - x_k\|^2 \right\}$$

where g_k is the estimated gradient in x_k and the second term plays the role of an *elastic potential* and forces x_{k+1} to be close to x_k.

This step is performed by the ICG algorithm.

Inexact ZSCG - ICG Algorithm

Algorithm 7: Inexact Conditional Gradient method (ICG)

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

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

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

Inexact ZSCG - ICG details

- Shares the general structure of Conditional Gradient algorithms;
- Minimizes the convex function:

$$H(u) = \langle g_k, u \rangle + \frac{\gamma}{2} \|u - x_k\|^2 \, ;$$

- Uses *Frank-Wolfe Gap* $h_\gamma(y_t)$ for the stopping criterion;
- Hence, the number of linear subproblems solved to find and a μ-stationary solution is $O\left(\frac{1}{\mu}\right)$.

Inexact ZSCG - Algorithm

Algorithm 8: Vanilla version of Inexact ZSCG

Input: $x_0 \in \chi$, $\nu > 0$, m_k, γ_k, μ_k, $N \geq 1$, probability distribution $P_R(\cdot)$ over $\{1, ..., N\}$.

for $k = 1, ..., N$ **do**

1. Generate $u_k = [u_{k,1}, ..., u_{k,m_k}]$, where $u_k \sim \mathcal{N}(0, I_d)$;

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

2. Set $x_k = ICG(x_{k-1}, G_\nu *^k, \gamma_k, \mu_k)$

end

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

Inexact ZSCG - Convergence rate

Instead of using the Frank-Wolfe Gap, an alternate convergence criterion is provided by using the *Gradient Mapping Function*

$$GP_{\chi}(x, g, \gamma) = \gamma(x - P_{\chi}(x, g, \gamma)).$$

By properly defining the parameters, and in particular setting $m_k = 6(d+5)N$ and $\mu = \frac{1}{4N}$, we have a convergence rate of

$$\mathbb{E}[\|GP_{\chi}(x_R, \nabla f(x_R), \gamma_R)\|^2] \leq \frac{8L}{N}\left(f(x_0) - f^* + L + B^2 + \sigma^2\right) .$$

The number of calls to the oracle and linear subproblems required to be solved to find an ϵ-stationary points of our problem are, respectively, bounded by $O\left(\frac{d}{\epsilon^2}\right)$, $O\left(\frac{1}{\epsilon^2}\right)$.

Inexact ZSCG for Adversarial Attacks

Algorithm 9: Inexact ZSCG for Adversarial Attacks

Input: Be $x_0 \in \chi$, be $\nu, m_k, \gamma_k, \mu_k > 0$, be $N \geq 1$, be S the set of accepted solutions

Set $q = 0,\ k = 0$

while $q = 0$ *and* $k \leq N$ **do**

1. Increment k
2. Generate $u_k = [u_{k,1}, ..., u_{k,m_k}]$, where $u_k \sim \mathcal{N}(0, I_d)$; call zero-th order oracle $m_k + 1$ times to generate

$$G_v^k \equiv G_v(x_{k-1}, \xi_k, u_k) = \frac{1}{m_k} \sum_{j=1}^{m_k} \frac{F(x_{k-1} + vu_{k,j}, \xi_{k,j}) - F(x_{k-1}, \xi_{k,j})}{v} u_{k,j}$$

3. Compute new value using ICG: $x_k = ICG(x_{k-1}, G_v^k, \gamma_k, \mu_k)$
4. Check stopping criterion $q = SC\left(x_k, F_M(\cdot, w), S\right)$

end

Return x_k

Setup

To execute the code we used Google Colaboratory, running on a Intel Xeon E5 CPU and a NVIDIA Tesla K80 GPU.
The chosen framework is PyTorch, using CUDA to run on the GPU.

Note: Results might be slightly inconsistent in terms of computational time due to Colab resources management.

Datasets

MNIST:

- 60.000 images of handwritten digits (0-9)
- single channel (i.e. black and white)
- 28×28 dimension, for a total of 784 pixels
- interval [0,1] for each pixel

Cifar10:

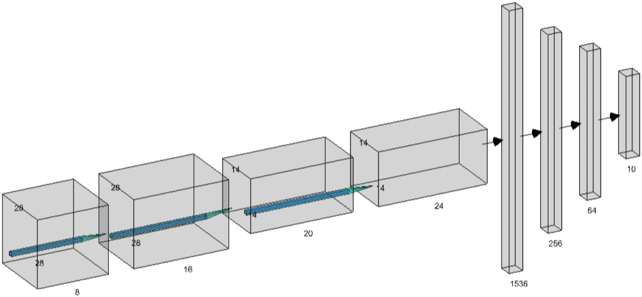
- 60.000 images divided in 10 classes (0-9)
- triple channel (i.e. colorized)
- 32×32 dimension, for a total of 3072 pixels
- interval [0,1] for each pixel

Models

Ad Hoc

Ad Hoc network trained on MNIST, consisting of:

- 4 Convolutional layers (kernel $= 2 \times 2$, ReLU activation)
- 2 Pooling layers (kernel $= 2 \times 2$)
- 3 Feed Forward layers (ReLU activation + Dropouts)



Models

VGG16

Used for the harder task of training a network on Cifar10. VGG16 with Batch Normalization is a deep network with the following structure:

- Feature Extraction Block
- 3 Feed Forward layers (ReLU activation + Dropouts)

where the Feature Extraction Block is made up of a total of 16 Convolutional layers (ReLU activation + Dropout) and 5 Max Pooling layers.

Models

InceptionV3

InceptionV3 is a 48 layers deep network created by Google in 2015. The networks requires an input of 3x299x299, for a total 268203 dimensions.

To use Inception on Cifar10 we had to rescale the images from 32×32 to 299×299.

Code Implementation

Source code:

`https://github.com/O4DS2020/final-project-paolofantuz`

Results - Attacks type

For each attack we evaluate the perturbation set using both L_2 and L_∞ norms.

- For the *Ad-Hoc (MNISTNet)* model and *VGG16* we perform both targeted and untargeted attacks.
- For *InceptionV3*, given the high complexity of the model and of the dimension space, only untargeted attacks have been performed.

Results - Evaluation metrics

Different evaluation metrics have been used to assess algorithm performance:

Success Rate - the percent of successful attacks;

Average Time - the average time taken to perform an attack;

Average Distance - the average distance between the original image x_0 and the last image found by the optimizer $\bar{x}$ (only for attacks with L_2)

$$\langle D \rangle = \frac{1}{N_{\text{examples}}} \sum_{n=1}^{N_{\text{examples}}} \|x_{0,n} - \bar{x}_n\|_2 .$$

Results - MNISTNet

Attacks against *MNIST* with L_∞ showed that *Inexact ZSCG* and *Frank-Wolfe* achieved the best performances but with the latter being much more slower (from 3 to 20 times slower), while in the case of L_2 the algorithm with the best Success Rate is ZOO with both the solvers, but it is also the slowest.

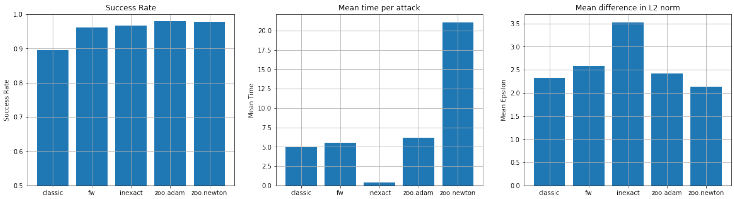


Figure: We can see how the *ZOO* algorithm achieve the best performance in term of Success Rate (right) and L_2 distance but not in terms of time (middle) since *Inexact ZSCG* is around 20 times faster.

Results - VGG16

Attacks against *VGG16* fine-tuned on *Cifar10* confirmed the robustness of *Frank-Wolfe* which had good performances in all the attacks against this model. It also showed a change of trajectory in the case of Inexact ZSCG, which had worse performances of its classic version (Classic ZSCG).

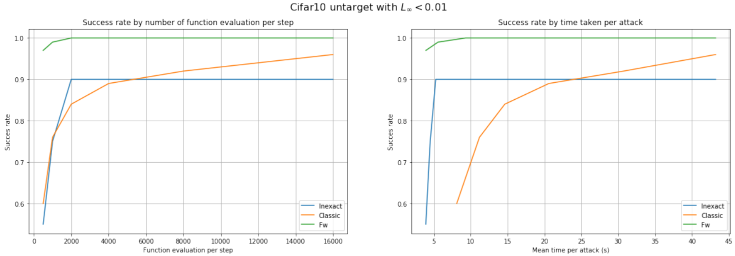


Figure: Untargeted attack against VGG16-Cifar10 with L_∞

Results - InceptionV3 - pt.1

In case of attacks against InceptionV3 a maximum time to perform an attack has been given to each algorithm: 300 seconds in case of attacks with L_∞ and 900 seconds in case of L_2.

This kind of time limitations didn't influence *Inexact ZSCG* very much, being the fastest algorithm but instead had a major effect on *Frank-Wolfe* which could perform an successful attack only once in ten (both with L_∞ and L_2). On the other hand *ZOO* achieved perfect Success rate but in almost twice the time of *Inexact ZSCG*.

Results - InceptionV3 - pt.2

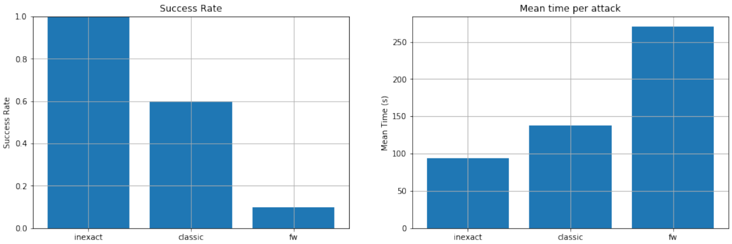


Figure: Untargeted attack against InceptionV3-Cifar10 with L_∞. Right: Sucess Rate for each algorithm; left: Average Time taken by each algorithms. We can see how *Inexact ZSCG* outperform *Frank-Wolfe* and *Classic ZSCG* both in terms of Success Rate and Average Time.

Results - InceptionV3 - pt.3

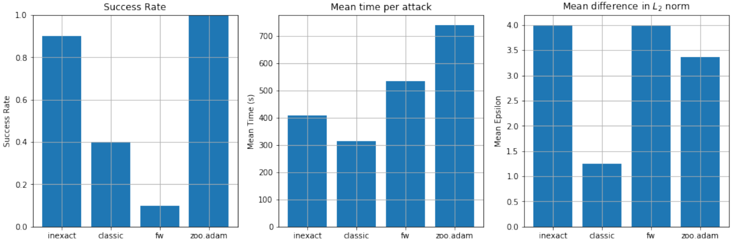


Figure: Untargeted attack against InceptionV3-Cifar10 with L_2. Right: Sucess Rate for each algorithm; middle: Average Time taken by each algorithms; left: Average L_2 distance. We can see how *ZOO* and *Inexact ZSCG* outperform *Frank-Wolfe* and *Classic ZSCG* in terms of Success Rate, but with *ZOO* being the slowest

.

Conclusion - Feasibility of the attacks

In the end, simulations, showed that performing an adversarial attack against a black box DNN is feasible but can be challenging, particularly if:

- the resolution of the image is high (hundreds of thousands to millions of dimensions);
- the number of (targeted) attacks to perform is of the order of thousand (e.g. an entire dataset);
- the restrictions on the feasible set χ are too strict (e.g. ϵ must be very small).

Conclusion - Methods comparison

Moreover if we analyse the results of the algorithms we can see that a clear winner does not exist, however some final points can be made:

- if the model to attack is fairly simple or the number of attacks to perform is low, using *ZOO* or *Frank-Wolfe* could be the best choice, since they proved to be the most accurate in terms of Success Rate if enough time is given;
- if an high number of attacks must be performed or the model attacked is very complex, ZSCG algorithms, particularly *Inexact ZSCG*, seems to be the best since the major factor to take in consideration is the time.

No Free Lunch Theorem?

It seems like we have a sort of No Free Lunch Theorem...

Bibliography

- Chen Pin-Yu, Zhang Huan, Sharma Yash. *ZOO: Zeroth Order Optimization based Black-box Attacks to Deep Neural Networks without Training Substitute Models*.
- Jinghui Chen, Dongruo Zhou, Jinfeng Yi, Quanquan Gu. *A Frank-Wolfe Framework for Efficient and Effective Adversarial Attacks*.
- Krishnakumar Balasubramanian, Saeed Ghadimi. *Zeroth-order Nonconvex Stochastic Optimization: Handling Constraints, High-Dimensionality and Saddle-Points*.
- Nicholas Carlini, David Wagner *Towards Evaluating the Robustness of Neural Networks*.
- Nesterov Yurii, Spokoiny Vladimir. *Random Gradient-Free Minimization of Convex Functions*