# FIRST ORDER APPROACHES FOR MIN-MAX PROBLEMS

Marco Baggio

## ABSTRACT

*In this paper we will analyze some of the first order approaches to min max problems presented in the papers 1 and 2. We will se some extensions of the Frank-Wolfe algorithm to solve constrained smooth convex-concave saddle point problems along with a novelty algorithm for solving min max saddle point games in the world of machine learning.*

## INTRODUCTION

The Frank-Wolfe optimization algorithm has seen a surge in popularity in recent machine learning applications due to his proficiency in analyzing structured constraining sets by accessing only to a linear minimization oracle. What we are going to see in this paper is an extended version of the Frank-Wolfe algorithm that extends his capabilities to convex-concave saddle point problems of the form

$$\min_{x \in X} \max_{y \in Y} \mathscr{L}\left(x, y\right)$$

where $\mathscr{L}$ is a smooth (with L-Lipschitz continuous gradient) convex-concave function.

Recently machine learning problems have been formulated as min max saddle point games. One example is the field of adversarial learning which can be formulated as such

$$\min_{\theta \in \Theta} \max_{\alpha \in \mathscr{A}} f\left(\theta, \alpha\right)$$

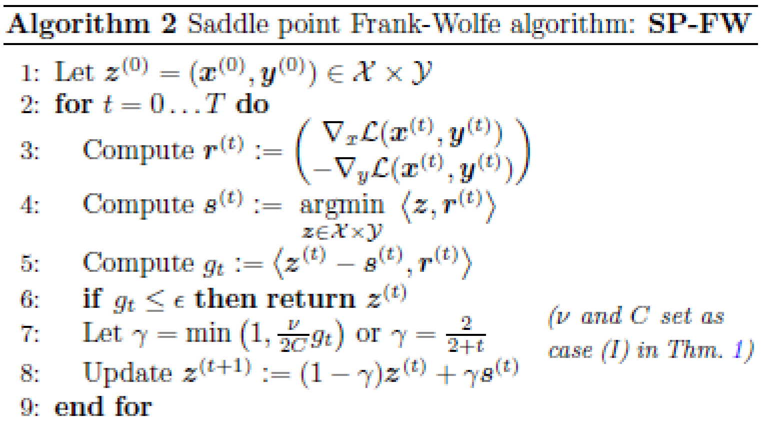
and can be viewed as a zero-sum game between two players in which the first player tries to minimize $f$ by tuning $\theta$ and the second player tries to maximize $f$ by tuning $\alpha$.
While the convex-concave settings has been extensively studied in the literature, recent machine learning applications urge the necessity of moving beyond these classical settings. The algorithm developed in paper 2 aims at solving more general non-convex non concave problems with a convergence to an $\epsilon$-first order Nash equilibrium.

## ALGORITHMS

### Saddle Point Frank Wolfe

The first variant of Frank Wolfe for saddle point problems is obtained by simultaneously computing a fw update on both convex functions $\mathscr{L}\left(\bullet, y^{(t)}\right)$ and $-\mathscr{L}\left(x^{(t)}, \bullet\right)$ with a properly chosen step-size. This generates at each iteration a point $z^{(t)} := \left(x^{(t)}, y^{(t)}\right)$. As in standard Frank Wolfe the point $z^{(t)}$ has a sparse representation as a convex combination of the points previously given by the Oracles

$$x^{(t)} = \sum_{v_x \in S_x^{(t)}} \alpha_{v_x} v_x \text{ and } y^{(t)} = \sum_{v_y \in S_y^{(t)}} \alpha_{v_y} v_y$$

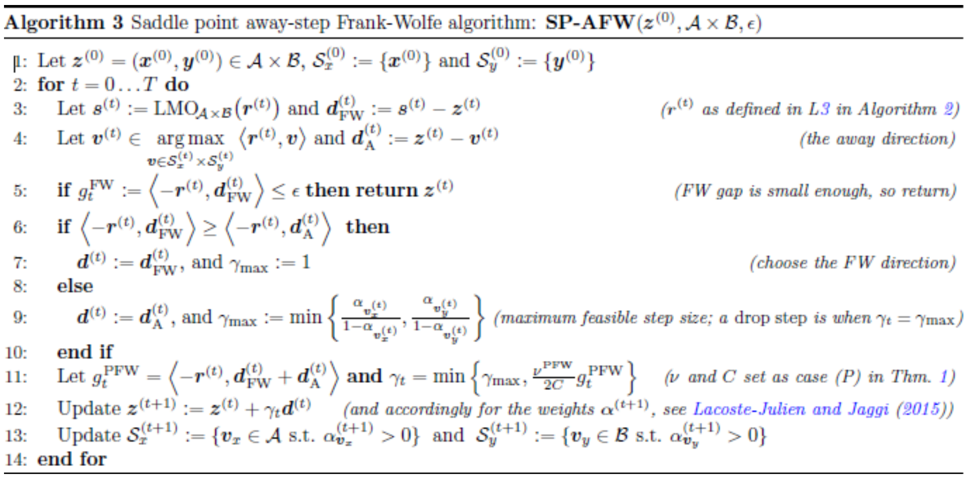


**Algorithm 2** Saddle point Frank-Wolfe algorithm: **SP-FW**

```
1: Let $z^{(0)} = (x^{(0)}, y^{(0)}) \in \mathcal{X} \times \mathcal{Y}$
2: for $t = 0 \ldots T$ do
3:     Compute $r^{(t)} := \begin{pmatrix} \nabla_x \mathcal{L}(x^{(t)}, y^{(t)}) \\ -\nabla_y \mathcal{L}(x^{(t)}, y^{(t)}) \end{pmatrix}$
4:     Compute $s^{(t)} := \underset{z \in \mathcal{X} \times \mathcal{Y}}{\operatorname{argmin}} \langle z, r^{(t)} \rangle$
5:     Compute $g_t := \langle z^{(t)} - s^{(t)}, r^{(t)} \rangle$
6:     if $g_t \leq \epsilon$ then return $z^{(t)}$
7:     Let $\gamma = \min\left(1, \frac{\nu}{2C} g_t\right)$ or $\gamma = \frac{2}{2+t}$      ($\nu$ and $C$ set as
8:     Update $z^{(t+1)} := (1-\gamma) z^{(t)} + \gamma s^{(t)}$                        case (I) in Thm. 1)
9: end for
```

**Figure 1:** Saddle Point Frank Wolfe

## Saddle point away-step Frank Wolfe



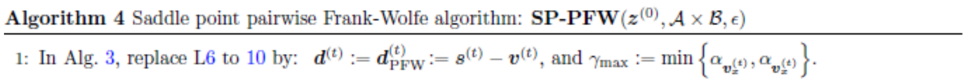
**Algorithm 3** Saddle point away-step Frank-Wolfe algorithm: **SP-AFW**$(z^{(0)}, \mathcal{A} \times \mathcal{B}, \epsilon)$

```
1: Let $z^{(0)} = (x^{(0)}, y^{(0)}) \in \mathcal{A} \times \mathcal{B}$, $\mathcal{S}_x^{(0)} := \{x^{(0)}\}$ and $\mathcal{S}_y^{(0)} := \{y^{(0)}\}$
2: for $t = 0 \dots T$ do
3:     Let $s^{(t)} := \mathrm{LMO}_{\mathcal{A}\times\mathcal{B}}(r^{(t)})$ and $d_{\mathrm{FW}}^{(t)} := s^{(t)} - z^{(t)}$    ($r^{(t)}$ as defined in L3 in Algorithm 2)
4:     Let $v^{(t)} \in \arg\max_{v \in \mathcal{S}_x^{(t)} \times \mathcal{S}_y^{(t)}} \langle r^{(t)}, v \rangle$ and $d_{\mathrm{A}}^{(t)} := z^{(t)} - v^{(t)}$    (the away direction)
5:     if $g_t^{\mathrm{FW}} := \left\langle -r^{(t)}, d_{\mathrm{FW}}^{(t)} \right\rangle \le \epsilon$ then return $z^{(t)}$    (FW gap is small enough, so return)
6:     if $\left\langle -r^{(t)}, d_{\mathrm{FW}}^{(t)} \right\rangle \ge \left\langle -r^{(t)}, d_{\mathrm{A}}^{(t)} \right\rangle$ then
7:         $d^{(t)} := d_{\mathrm{FW}}^{(t)}$, and $\gamma_{\max} := 1$    (choose the FW direction)
8:     else
9:         $d^{(t)} := d_{\mathrm{A}}^{(t)}$, and $\gamma_{\max} := \min\left\{\frac{\alpha_{v_x^{(t)}}}{1-\alpha_{v_x^{(t)}}}, \frac{\alpha_{v_y^{(t)}}}{1-\alpha_{v_y^{(t)}}}\right\}$    (maximum feasible step size; a drop step is when $\gamma_t = \gamma_{\max}$)
10:    end if
11:    Let $g_t^{\mathrm{PFW}} = \left\langle -r^{(t)}, d_{\mathrm{FW}}^{(t)} + d_{\mathrm{A}}^{(t)} \right\rangle$ and $\gamma_t = \min\left\{\gamma_{\max}, \frac{\nu^{\mathrm{PFW}}}{2C} g_t^{\mathrm{PFW}}\right\}$    ($\nu$ and $C$ set as case (P) in Thm. 1)
12:    Update $z^{(t+1)} := z^{(t)} + \gamma_t d^{(t)}$    (and accordingly for the weights $\alpha^{(t+1)}$, see Lacoste-Julien and Jaggi (2015))
13:    Update $\mathcal{S}_x^{(t+1)} := \{v_x \in \mathcal{A} \text{ s.t. } \alpha_{v_x}^{(t+1)} > 0\}$ and $\mathcal{S}_y^{(t+1)} := \{v_y \in \mathcal{B} \text{ s.t. } \alpha_{v_y}^{(t+1)} > 0\}$
14: end for
```

**Figure 2:** Saddle Point away-step Frank Wolfe

If we assume that $\mathscr{X}$ and $\mathscr{Y}$ are the convex hulls of two finite sets of points $\mathscr{A}$ and $\mathscr{B}$ we can also extend Away Step Frank Wolfe algorithm to saddle point problems. This way we can benefit from the away direction $d_{\mathscr{A}}$ of AFW to avoid the zig-zagging problem of standard Frank Wolfe.

## Saddle Point Pairwise Frank Wolfe



**Algorithm 4** Saddle point pairwise Frank-Wolfe algorithm: **SP-PFW**$(z^{(0)}, \mathcal{A} \times \mathcal{B}, \epsilon)$

```
1: In Alg. 3, replace L6 to 10 by:  $d^{(t)} := d_{\mathrm{PFW}}^{(t)} := s^{(t)} - v^{(t)}$, and $\gamma_{\max} := \min\left\{\alpha_{v_x^{(t)}}, \alpha_{v_x^{(t)}}\right\}$.
```

**Figure 3:** Saddle Point Pairwise Frank Wolfe

The last variant is an adaptation Saddle point away-step Frank Wolfe to Saddle point pairwise Frank Wolfe.

## Multi Step Frank Wolfe/Projected Gradient Step

When considering a two-player zero sum min-max game of the form

$$\min_{\theta \in \Theta} \max_{\alpha \in \mathscr{A}} f(\theta, \alpha)$$

where $\Theta$ and $\mathscr{A}$ are both convex sets and $f$ is a continuously differentiable function we can define a Nash equilibrium as $(\theta^\star, \alpha^\star) \in \Theta \times \mathscr{A}$ if

$$f(\theta^\star, \alpha) \leq f(\theta^\star, \alpha^\star) \leq f(\theta, \alpha^\star) \forall \theta \in \Theta, \forall \alpha \in \mathscr{A}$$

In convex-concave there is always a Nash equilibrium. However in non-convex non-concave problems finding the local Nash equilibrium is NP-hard. Therefore from now on we will use the first-order Nash equilibrium, defined as follows

**Definition**(FNE) *A point $(\theta^\star, \alpha^\star) \in \Theta \times \mathscr{A}$ is a first order Nash equilibrium of the two-player zero sum min-max game if*

$$\langle \nabla_\theta f(\theta^\star, \alpha^\star), \theta - \theta^\star \rangle \geq 0, \forall \theta \in \Theta \wedge \langle \nabla_\alpha f(\theta^\star, \alpha^\star), \alpha - \alpha^\star \rangle \leq 0, \forall \alpha \in \mathscr{A}$$

Since in practice we use iterative methods for computation, we need to define the notion of approximate-FNE

**Definition**(Approximate FNE) *A point $(\theta^\star, \alpha^\star)$ is said to be $\epsilon$-first-order Nash Equilibrium $(\epsilon - FNE)$ of the min max game if*

$$\mathscr{X}(\theta^\star, \alpha^\star) \leq \epsilon \; and \; \mathscr{Y}(\theta^\star, \alpha^\star) \leq \epsilon$$

*where*

$$\mathscr{X}(\theta^\star, \alpha^\star) \triangleq - \min_\theta \langle \nabla_\theta f(\theta^\star, \alpha^\star), \theta - \theta^\star \rangle \; s.t. \; \theta \in \Theta, \; \| \theta - \theta^\star \| \leq 1$$

*and*

$$\mathscr{Y}(\theta^\star, \alpha^\star) \triangleq \max_\alpha \langle \nabla_\alpha f(\theta, \alpha), \alpha - \alpha^\star \rangle \; s.t. \; \alpha \in \mathscr{A}, \; \| \alpha - \alpha^\star \| \leq 1$$

Under the assumption that the objective function $f$ is continuously differentiable in both $\theta$ and $\alpha$ and smooth.

Now that we have a definition of approximate FNE we can extend the algorithm for min-max games to work with Non-convex PL games. PL-games or Polyak-Lojasiewicz games are min max games where one of the players satisfies the PL condition defines as such

**Definition**(PL Condition)*A differentiable function $h(x)$ with the minimum value $h^\star = \min_x h(x)$ is said to be $\mu$-Polyak-Lojasiewicz if*

$$\frac{1}{2} \parallel \nabla h(x) \parallel^2 \geq \mu(h(x) - h^\star), \ \forall x$$

and therefore in our case a PL game can be defined as such

**Definition**(PL game)*We say that the min-max game is a PL-game if the max player is unconstrained, i.e. $\mathscr{A} \in \mathbb{R}^n$, and there exists a constant $\mu > 0$ such that the function $h_\theta(\alpha) \triangleq -f(\theta, \alpha)$ is $\mu$-PL for any fixed value of $\theta \in \Theta$*

If we rewrite our problem in the form

$$\min_{\theta \in \Theta} g(\theta)$$

where

$$g(\theta) \triangleq \max_{\alpha \in \mathscr{A}} f(\theta, \alpha)$$

we can write a multi-step gradient ascent descent algorithm that at each iteration runs multiple projected gradient ascent steps to estimate the solution of the inner maximization problem, hence providing the following result

$$\nabla_\theta g(\theta) = \nabla_\theta f\left(\theta, \alpha^\star\right) \ with \ \alpha^\star \in \ arg\max_{\alpha \in \mathscr{A}} f(\theta, \alpha)$$

**Algorithm 1** Multi-step Gradient Descent Ascent

```
1: INPUT: $K, T, \eta_1 = 1/L_{22}, \eta_2 = 1/L, \boldsymbol{\alpha}_0 \in \mathcal{A}$ and $\boldsymbol{\theta}_0 \in \Theta$
2: for $t = 0, \cdots, T-1$ do
3:     Set $\boldsymbol{\alpha}_0(\boldsymbol{\theta}_t) = \boldsymbol{\alpha}_t$
4:     for $k = 0, \cdots, K-1$ do
5:         Set $\boldsymbol{\alpha}_{k+1}(\boldsymbol{\theta}_t) = \boldsymbol{\alpha}_k(\boldsymbol{\theta}_t) + \eta_1 \nabla_{\boldsymbol{\alpha}} f(\boldsymbol{\theta}_t, \boldsymbol{\alpha}_k(\boldsymbol{\theta}_t))$
6:     end for
7:     Set $\boldsymbol{\theta}_{t+1} = \mathrm{proj}_\Theta\left(\boldsymbol{\theta}_\mathrm{t} - \eta_2 \nabla_{\boldsymbol{\theta}} \mathrm{f}(\boldsymbol{\theta}_\mathrm{t}, \boldsymbol{\alpha}_\mathrm{K}(\boldsymbol{\theta}_\mathrm{t}))\right)$
8: end for
9: Return $(\boldsymbol{\theta}_t, \boldsymbol{\alpha}_K(\boldsymbol{\theta}_t))$ for $t = 0, \cdots, T-1$.
```

**Figure 4:** Multi-step Gradient Ascent Descent

Going a step further we now analyze the case of non-convex concave games falling under the following assumption

**Assumption** *The objective function $f(\theta,\alpha)$ is concave in $\alpha$ for any fixed value of $\theta$. Moreover the set $\mathscr{A}$ is convex compact, and there exists a ball with radius R that contains the feasible set $\mathscr{A}$*

In this case we need to keep in mind that the aforementioned function $g(\theta)$ might not be differentiable therefore we use a little tweak to transform it into a differentiable function adding the following parameter

$$g_\lambda(\theta) \triangleq \max_{\alpha\in\mathscr{A}} f_\lambda(\theta,\alpha)$$

where $f_\lambda(\theta,\alpha) \triangleq f(\theta,\alpha) - \frac{\lambda}{2} \parallel \alpha - \bar{\alpha} \parallel^2$ where $\bar{\alpha} \in \mathscr{A}$ is some given fixed point and $\lambda > 0$ is a regularization parameter.
Since $f_\lambda$ is Lipschitz smooth and based on the compactness assumption we can define

$$g_\theta \triangleq \max_{\theta\in\Theta} \parallel \nabla g_\lambda(\theta) \parallel,\; g_\alpha \triangleq \max_{\theta\in\Theta} \parallel \nabla_\alpha f_\lambda(\theta,\alpha^\star(\theta)) \parallel,\; and\; g_{max} = \max\{g_\theta, g_\alpha, 1\}$$

where $\alpha^\star(\theta) \triangleq arg\max_{\alpha\in\mathscr{A}} f_\lambda(\theta,\alpha)$.

The proposed algorithm is composed of two major steps. The first step runs K step of accellerated gradient ascent algorithm over the variable $\alpha$ with restart every N iteration while the second step runs either Project Gradient Descent or Frank Wolfe update rules defined as such

$$\theta_{t+1} \triangleq proj_\Theta\left(\theta_t - \frac{1}{L_{11} + \frac{L_{12}^2}{\lambda}} \nabla_\theta f_\lambda(\theta_t, \alpha_{t+1})\right)$$

or

$$\theta_{t+1} \triangleq \theta_t + \frac{-\min_s \langle \nabla_\theta f_\lambda(\theta_t, \alpha_{t+1}, s\rangle}{\tilde{L}} arg\min_s \langle \nabla_\theta f_\lambda(\theta_t, \alpha_K(\theta_t)), s\rangle$$
$$s.t.\, \theta_t + s \in \Theta,\; \parallel s \parallel \leq 1$$

**Algorithm 2** Multi-Step Frank Wolfe/Projected Gradient Step Framework

```
Require: Constants $\widetilde{L} \triangleq \max\{L, L_{12}, g_{max}\}$, $N \triangleq \lfloor\sqrt{8L_{22}/\lambda}\rfloor$, $K$, $T$, $\eta$, $\lambda$, $\boldsymbol{\theta}_0 \in \Theta$, $\boldsymbol{\alpha}_0 \in \mathcal{A}$
1: for $t = 0, 1, 2, \ldots, T$ do
2:     Set $\boldsymbol{\alpha}_{t+1} = \text{APGA}(\boldsymbol{\alpha}_t, \boldsymbol{\theta}_t, \eta, N, K)$ by running $K$ steps of Accelerated Projected Gradient
       Ascent subroutine (Algorithm 3) with periodic restart at every $N$ iteration.
3:     Compute $\boldsymbol{\theta}_{t+1}$ using first-order information (Frank-Wolfe or projected gradient descent).
4: end for
```

**Figure 5:** Multi-Step Frank Wolfe/Project Gradient Step Framework

**Algorithm 3** APGA: Accelerated Projected Gradient Ascent with Restart

```
Require: Constants $\boldsymbol{\alpha}_t$, $\boldsymbol{\theta}_t$, $\eta$, $K$, and $N$.
1: for $k = 0, \ldots, \lfloor K/N \rfloor$ do
2:     Set $\gamma_1 = 1$
3:     if $k = 0$ then $\mathbf{y}_1 = \alpha_t$ else $\mathbf{y}_1 = \mathbf{x}_N$
4:     for $i = 1, 2, \ldots, N$ do
5:         Set $\mathbf{x}_i = \text{proj}_{\mathcal{A}}(\mathbf{y}_i + \eta \nabla_{\mathbf{y}} f_\lambda(\boldsymbol{\theta}_t, \mathbf{y}_i))$
6:         Set $\gamma_{i+1} = \dfrac{1 + \sqrt{1 + 4\gamma_i^2}}{2}$
7:         $\mathbf{y}_{i+1} = \mathbf{x}_i + \left(\dfrac{\gamma_i - 1}{\gamma_{i+1}}\right)(\mathbf{x}_i - \mathbf{x}_{i-1})$
8:     end for
9: end for
10: Return $\mathbf{x}_N$
```

**Figure 6:** Accelerated Projected Gradient Ascent with Restart

## EXPERIMENTS

To Test the multi-step Frank-Wolfe/Project Gradient Descent algorithm we propose an application in the field of machine learning. In particular we describe a problem of image recognition subject to adversarial attack. It has been studied in fact that image recognition machine learning algorithm are highly susceptibles to minor perturbation of the images. To train a robust neural network against adversarial attack the procedure of training has been reformulated to use a min-max optimization formulation as such

$$\min_{w} \sum_{i=1}^{N} \delta_i \max_{s.t. \|\delta_i\|_\infty \leq \epsilon} l(f(x_i + \delta_i; w), y_i)$$

where $w$ is the parameter fo the neural network, $(x_i, y_i)$ is the $i - th$ data point and $\delta_i$ is the perturbation added to the data point $i$.
However solving the presented problem is computationally challenging being non-concave non-convex. Therefore, to make use of the proposed algorithm a reformulation of the problem into a concave optimization problem as such

$$\min_{w} \sum_{i=1}^{N} \max\{l(f(\hat{x}_{i0}(w); w), y_i), \ldots, l(f(\hat{x}_{i9}(w); w), y_i)\}$$

where each $\hat{x}_{ij}(w)$ is the result of a targeted attack on the sample $x_i$ aiming at changing the output of the network label $j$. In particular in our case we are training our machine learning algorithm against the database MNIST, in the one but last layer we have 10 different neurons each corresponding with one category of classification so, for each sample $(x_i, y_i)$ we run gradient ascent to obtain the following chain of points

$$x_{ij}^{k+1} = Proj_{B(x,\epsilon)} \left[ x_{ij}^{k} + \alpha \nabla_x (Z_j(x_{ij}^{k}, w) - Z_{y_i}(x_{ij}^{k}, w)) \right], k = 0, \ldots, K-1$$

Where $Z_j$ is the network logit before softmax corresponding to label $j$, $\alpha > 0$ is the step-size and $Proj_{B(x,\epsilon)}[\bullet]$ is the projection on the infinity ball with radius $\epsilon$ centered at $x$. With this we set $\hat{x})ij(w) = x_{ij}^{K}$.

With this formulation we can apply our algorithm. Below there are the results for the loss computed with Project Gradient Descent. For each run of the algorithm the following parameters were tested in different configurations to test the best performing: The learning rate, the $\epsilon$ and the number of iteration for the internal cycle. Each time two out of the three parameter were fixed while the third changed to test different values in the same environment.

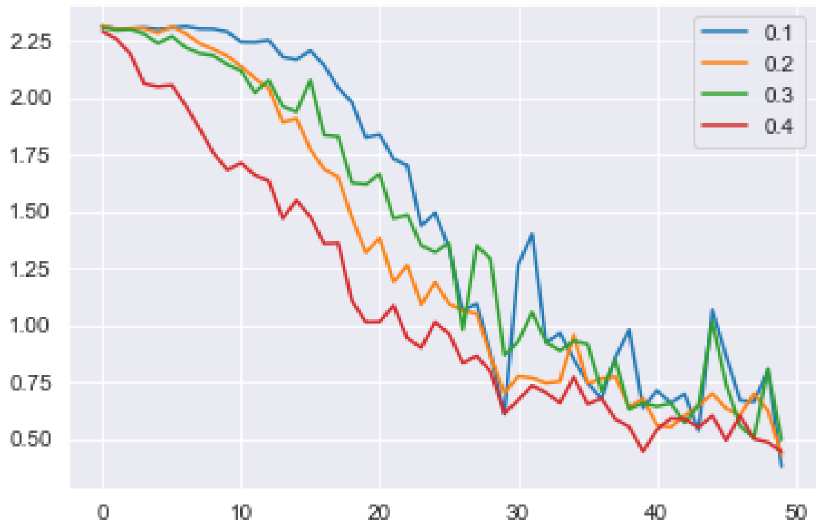


**Figure 7:** Loss with learning rate variations

In figure 7 we can see the evolution of the values of the loss with different learning rates. It is interesting to notice that, while all the variants converge to roughly the same final value a low learning rate might result in heavier fluctuations than a higher rate. However, it has to be considered that for the sake of testing, the overall epochs for the machine learning algorithms were few so the reduced cycles of training might have affected the results.

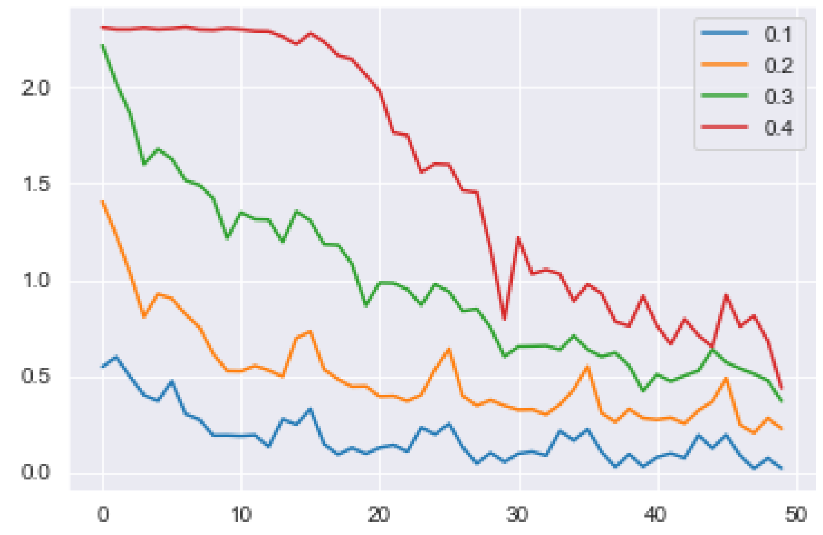


**Figure 8:** Loss with $\epsilon$ variations

Regarding figure 8 we can appreciate a clean distinction between values. With a lower tollerance we obtain not only a better loss value right from the fist iteration but we also reach a final better loss with consistency across the values $\epsilon$ takes.

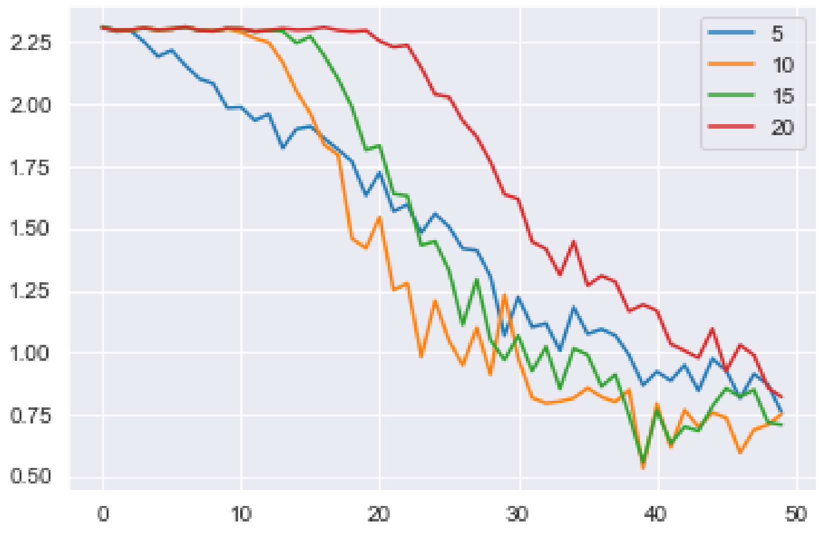


**Figure 9:** Loss with iteration variations

Finally we have a look on variations in the number of iterations for the internal cycle. As seen in figure 9 it is much harder to define the right number of iterations to choose for the algorithm. One take away from this results is that a lower number of iterations tend to show results from the first steps of the algorithm, while an higher number seems to struggle more in the initial phases while having a steeper descent toward the final cycles of the learning process.

To test the overall validity of the obtained results each trained model has been tested against both Project Gradient Descent and Fast Gradient Sign Attacks with different $\epsilon$ parameters. Considering the low number of training epochs available to the machine learning algorithm the acceptable results against these attacks were kept pretty loose.

## REFERENCES

1. "Frank-Wolfe Algorithms for Saddle Point Problems" Gauthier Gidel, Tony Jebara Simon, Lacoste-Julien
2. "Solving a Class of Non-Convex Min-Max Games Using Iterative First Order Methods" Maher Nouiehed, Maziar Sanjabi, Tianjian Huang, Jason D. Lee