Variance Reduction in Projection-Free Optimization

Optimization For Data Science Final Project

Gianmarco Cracco, Alessandro Manente, Riccardo Mazzieri

27th June 2020

Contents

1 Introduction

The Frank-Wolfe optimization algorithm has recently regained popularity in Big Data applications thanks to its projection-free property and its ability to handle structured constraints. In fact, for important classes of constraints, such as the unit norm ball of the ℓ_1 or nuclear norm, linear optimization over the constraint set is much faster than projection onto that same set. In this work, we will describe several improved versions of the classic Frank-Wolfe method, which exploit stochastic gradients and variance reduction techniques. We will also see the main theoretical results in terms of convergence and show the performances of those algorithms applied on real world datasets, in the setting of multiclass classification. To give a complete comparison even with projection-based algorithms, at the end we will add the results of stochastic and variance reduced version of Projected Gradient Method. We report in Table 1 a summary of the main results that are going to be discussed. Note that the efficiency of any projection-free algorithm is measured by how many numbers of exact gradient evaluations, stochastic gradient evaluations and linear optimizations (LO) respectively are needed to achieve 1-ϵ accuracy.

Algorithm	*Extra Conditions*	*#Exact Gradients*	*#Stochastic Gradients*	*#Linear Optimizations*
FW		$\mathcal{O}\left(\frac{LD^2}{\epsilon}\right)$	0	$\mathcal{O}\left(\frac{LD^2}{\epsilon}\right)$
SFW	G-Lipschitz	0	$\mathcal{O}\left(\frac{G^2LD^4}{\epsilon^3}\right)$	$\mathcal{O}\left(\frac{LD^2}{\epsilon}\right)$
SCGS		0	$\mathcal{O}\left(\sqrt{\frac{LD^2}{\varepsilon}}+\frac{\sigma^2D^2}{\epsilon^2}\right)$	$\mathcal{O}\left(\frac{LD^2}{\epsilon}\right)$
SVRF		$\mathcal{O}\left(\ln\frac{LD^2}{\epsilon}\right)$	$\mathcal{O}\left(\frac{L^2D^4}{\epsilon^2}\right)$	$\mathcal{O}\left(\frac{LD^2}{\epsilon}\right)$
STORC	G-Lipschitz	$\mathcal{O}\left(\ln\frac{LD^2}{\epsilon}\right)$	$\mathcal{O}\left(\frac{\sqrt{LD^2G}}{\epsilon^{1.5}}\right)$	$\mathcal{O}\left(\frac{LD^2}{\epsilon}\right)$
	$\nabla f(w*)=0$	$\mathcal{O}\left(\ln\frac{LD^2}{\epsilon}\right)$	$\mathcal{O}\left(\frac{LD^2}{\epsilon}\right)$	$\mathcal{O}\left(\frac{LD^2}{\epsilon}\right)$
	α strongly convex	$\mathcal{O}\left(\ln\frac{LD^2}{\epsilon}\right)$	$\mathcal{O}\left(\mu^2\ln\frac{LD^2}{\epsilon}\right)$	$\mathcal{O}\left(\frac{LD^2}{\epsilon}\right)$

Table 1: Number of exact gradient evaluations, stochastic gradient evaluations and linear optimizations needed to achieve 1-ϵ accuracy. L = smoothness constant, G = G-Lipschitz constant.

In the end, we also analyze an online version of the Frank Wolfe algorithm which is presented in [3].

1.1 Basic Definitions

Firstly let's give some definitions useful for the comprehension of the present work.

Diameter.

$$D \coloneqq \max_{x,y\in\Omega}\|x-y\|_* < +\infty. \tag{1}$$

L-smoothness (or **Lipschitz Continuous Gradient).** Let f be a convex function. f is L-smooth in Ω if for any $v, w \in \Omega$,

$$\|\nabla f(w)-\nabla f(v)\| \leq L\|w-v\| \tag{2}$$

Equivalently

$$f(v+w) \leq f(v)+\nabla f(v)\cdot w+\frac{L}{2}\|w\|^2 \tag{3}$$

Furthermore, explicitly using the convexity we can write:

$$\nabla f(v)^T(w-v) \leq f(w)-f(v) \leq \nabla f(v)^T(w-v)+\frac{L}{2}\|w-v\|^2 \tag{4}$$

If we have that f is defined in $\mathbb{R}^d$, two important properties follow:

1. $\|\nabla f_i(w)-\nabla f_i(v)\|^2 \leq 2L(f_i(w)-f_i(v)-\nabla f_i(v)^T(w-v))$;
2. $f_i(\lambda w+(1-\lambda)v) \geq \lambda f_i(w)+(1-\lambda)f_i(v)-\frac{L}{2}\lambda(1-\lambda)\|w-v\|^2$.

G-Lipschitz. f is G-Lipschitz if $\forall w \in \Omega$ we have $\|\nabla f_i(w)\| \leq G$.
If we also have that f is α-strongly convex than we call $\mu=\frac{L}{\alpha}$ the condition number of f.

1.2 Problem Setting

From now on we will consider a multiclass classification problem, with a given training set $(e_i, y_i)_{i=1,\dots,n}$ where $e_i \in \mathbb{R}^m$ is the feature vector and $y_i \in H = \{1, \dots, h\}$ is a label.
Let's define $\Omega \coloneqq \{\omega \in \mathbb{R}^{h\times m} : \|\omega\|_* \leq \tau\}$, where $\|\cdot\|_*$ is the *trace norm*; hence the aim is to find a matrix $\omega = [\omega_1, \dots, \omega_h]^\top \in \mathbb{R}^{h\times m}$ such that:

$$y_i = \underset{\ell}{\operatorname{argmax}} \left(\omega_\ell^\top e_i\right) \quad \forall i. \tag{5}$$

Note that the dimensionality is $d = h \cdot m$: this translates in a problem with the following form

$$\min_{\omega\in\Omega} f(\omega) = \min_{\omega\in\Omega} \frac{1}{n}\sum_{i=1}^{n} f_i(\omega) \tag{6a}$$

$$f_i(\omega) = \log\left(1 + \sum_{\ell\neq y_i} \exp\left(\omega_\ell^\top e_i - \omega_{y_i}^\top e_i\right)\right) \tag{6b}$$

note that f_i is convex and L-smooth $\forall \omega \in \Omega$.
Here we explicitly report the computation we performed to evaluate $\nabla f(w)$. Setting

$$\alpha_i(\omega) = \sum_{\ell\neq y_i}^{h} \exp\left(\omega_\ell^\top e_i - \omega_{y_i}^\top e_i\right)$$

our loss function becomes

$$f(\omega) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + \alpha_i(\omega)).$$

Therefore

$$\begin{aligned}
\frac{\partial f_i}{\partial \omega_{bk}}(\omega) &= \frac{\partial}{\partial \omega_{bk}} \log(1+\alpha_i(\omega)) = \frac{1}{1+\alpha_i(\omega)} \frac{\partial}{\partial \omega_{bk}} \alpha_i(\omega) = \frac{1}{1+\alpha_i(\omega)} \sum_{\ell\neq y_i} \frac{\partial}{\partial \omega_{bk}} \exp\left(\left(\omega_l - \omega_{y_i}\right)^\top e_i\right) = \\
&= \frac{1}{1+\alpha_i(\omega)} \sum_{\ell\neq y_i} \exp\left(\left(\omega_l - \omega_{y_i}\right)^\top e_i\right) \frac{\partial}{\partial \omega_{bk}} \left(\left(\omega_l - \omega_{y_i}\right)^\top e_i\right) = \\
&= \frac{1}{1+\alpha_i(\omega)} \sum_{\ell\neq y_i} \exp\left(\left(\omega_l - \omega_{y_i}\right)^\top e_i\right) \left[\frac{\partial}{\partial \omega_{bk}} \omega_l \cdot e_i - \frac{\partial}{\partial \omega_{bk}} \omega_{y_i} \cdot e_i\right] = \\
&= \frac{1}{1+\alpha_i(\omega)} \sum_{\ell\neq y_i} \exp\left(\left(\omega_l - \omega_{y_i}\right)^\top e_i\right) \left[\frac{\partial}{\partial \omega_{bk}} \left(\omega_{l1} e_{i1} + \dots + \omega_{lm} e_{im}\right) - \frac{\partial}{\partial \omega_{bk}} \left(\omega_{y_i l} e_{i1} + \dots + \omega_{jk} c_{im}\right)\right]
\end{aligned}$$

So finally we obtain

$$\frac{\partial f_i(\omega)}{\partial \omega_{b,k}} = \frac{1}{1+\alpha_i(\omega)} \begin{cases} \exp\left(\left(\omega_b - \omega_{y_i}\right)^\top e_i\right) e_{ik} & \text{if} \quad b \neq y_i \\ e_{ik} & \text{if} \quad b = y_i \end{cases} \tag{7}$$

From now on the descend direction will be defined as:

$$\hat{v} = \underset{v\in\Omega}{\operatorname{argmin}} \nabla f\left(\omega_{k-1}\right)^\top v \tag{8}$$

but since Ω is a $matrix-$space, it's worth noticing that the actual problem becomes:

$$\hat{v} = \underset{v\in\Omega}{\operatorname{argmin}}\ tr\left(\nabla f\left(\omega_{k-1}\right)^\top v\right) \tag{9}$$

From now on, for sake of clarity we will often omit the tr in order to have a lighter and more general notation. So, the solution is given by [2]:

$$\hat{v}_k = -\tau u_1 v_1^\top \tag{10}$$

where $u_1,\ v_1$ are top singular vectors of $\nabla f\left(\omega_{k-1}\right)$ and τ is the upper bound for $\|\omega\|_*$.
Each algorithm will try to solve the problem until the stopping condition is satisfied, namely until:

$$\mathbb{E}[f(\omega) - f(\omega^*)] \leq \varepsilon \tag{11}$$

for a given $\varepsilon > 0$.

2 The Algorithms

2.1 Frank-Wolfe or Conditional Gradient Method (FW)

The Frank-Wolfe method contains the main idea that is shared between all the algorithms that we will present. It is an iterative first-order optimization algorithm, which starts with a feasible solution and, at each iteration, solves the following linear optimization problem to find the descent direction a stated in (9). From compactness of Ω, there exists $\hat{v}_k \in \Omega$ solution for the problem. This idea allows to avoid the very costly projection step required by other popular optimization methods like Projected Gradient Method, and is the reason why this method has seen an impressive revival, in particular for machine learning applications.

Once the descent direction $\hat{v}$ is found there are two possibilities:

1. If $\nabla f\left(\omega_{k-1}\right)^{\top} \hat{v} = 0 \Rightarrow \omega_{k-1}$ satisfies first order optimality conditions;
2. If $\nabla f\left(\omega_{k-1}\right)^{\top} \hat{v} < 0 \Rightarrow d_k = \hat{v}$ is a new descent direction.
   Finally the new iterate is:
$$\omega_k = \omega_{k-1} + \gamma_k(\hat{v_k} - \omega_{k-1}) \qquad \gamma_k \in (0, 1] \tag{12}$$

The pseudocode of the method is:

Algorithm 1 *Frank-Wolfe method (FW)*

Initialize: $\omega_0 \in \Omega$;
for $k = 1, \ldots$ **do**
  Set $\hat{v}_k = \underset{v \in \Omega}{\operatorname{argmin}}\ tr\left(\nabla f\left(\omega_{k-1}\right)^{\top} v\right)$;
  If $\hat{v}_k$ satisfies the first order optimality condition, then **STOP**;
  Else, set $\omega_k = \omega_{k-1} + \gamma_k(\hat{v}_k - \omega_{k-1})$, with $\gamma_k \in (0, 1]$ suitably chosen stepsize;
end for

To achieve $1 - \varepsilon$ accuracy the method requires [5]:

- $\mathcal{O}\left(\frac{LD^2}{\varepsilon}\right)$ exact gradients
- $\mathcal{O}\left(\frac{LD^2}{\varepsilon}\right)$ linear optimizations.

2.2 Stochastic Frank-Wolfe (SFW)

This algorithm is obtained starting from FW and simply replacing $\nabla f\left(w_{k-1}\right)$ with some $\nabla f_i\left(w_{k-1}\right)$ where $\nabla f_i\left(\cdot\right)$ is the gradient computed only w.r.t the sample i.
The pseudocode of the method is:

Algorithm 2 *Stochastic Frank-Wolfe method (SFW)*

Input: batch size m_k;
Initialize: $\omega_0 \in \Omega$;
for $k = 1, \ldots$ **do**
  Consider the set $\{i_1, \ldots, i_{m_k}\} \subset \{1, \ldots, n\}$ picked uniformly at random, and define $\nabla f_{m_k}\left(\omega_{k-1}\right) := \frac{1}{m_k} \sum_{j=1}^{m_k} \nabla f_{i_j}\left(\omega_{k-1}\right)$;
  Set $\hat{v}_k = \underset{v \in \Omega}{\operatorname{argmin}}\ tr\left(\nabla f_{m_k}\left(\omega_{k-1}\right)^{\top} v\right)$;
  If $\hat{v}_k$ satisfies the first order optimality condition, then **STOP**;
  Else, set $\omega_k = \omega_{k-1} + \gamma_k(\hat{v}_k - \omega_{k-1})$, with $\gamma_k \in (0, 1]$ suitably chosen stepsize;
end for

To achieve $1 - \varepsilon$ accuracy the method requires, if the function is G-Lipschitz[5]:

- $\mathcal{O}\left(\frac{G^2 LD^4}{\varepsilon^3}\right)$ stochastic gradients
- $\mathcal{O}\left(\frac{LD^2}{\varepsilon}\right)$ linear optimizations.

Let's now prove the convergence of the method with the following:

Theorem 2.1. *If f_i is $G-Lipschitz$ $\forall i$, then with $\gamma_k = \frac{2}{k+1}$ and $m_k = \left(\frac{G(k+1)}{LD}\right)^2$, SFW ensures for any k,*

$$\mathbb{E}[f(\omega_k) - f(\omega*)] \leq \frac{4LD^2}{k+2}. \tag{13}$$

Proof. We first prove that:

$$f(\omega_k) \leq f(\omega_{k-1}) + \gamma_k \left(f(\omega^*) - f(\omega_{k-1})\right) + \gamma_k D \left\|\bar{\nabla}_k - \nabla f(\omega_{k-1})\right\| + \frac{LD^2\gamma_k^2}{2}, \tag{14}$$

using smoothness, optimality of v_k, convexity and Cauchy-Schwartz inequality. Since f_i is $G-Lipschitz$, with Jensen's inequality, we further have, calling $\tilde{\nabla}_k$ the average of m_k iid samples of stochastic gradient $\nabla f_i(\omega_{k-1})$, that $\mathbb{E}[\|\tilde{\nabla}_k - \nabla f(\omega_{k-1})\|] \leq \frac{G}{\sqrt{m_k}}$, which is at most $\frac{LD\gamma_k}{2}$ with the choice of γ_k and m_k. So we obtain that $\mathbb{E}[f(\omega_k) - f(\omega*)] \leq (1-\gamma_k)\mathbb{E}[f(\omega_k) - f(\omega*)] + LD^2\gamma_k^2$.
With a simple induction the proof is concluded. □

2.3 Stochastic Variance Reduced Frank-Wolfe (SVRF)

A variance-reduced stochastic gradient at a point $\omega \in \Omega$ with a *snapshot* $\omega_0 \in \Omega$ is defined as

$$\tilde{\nabla} f(\omega; \omega_0) := \nabla f_i(\omega) - (\nabla f_i(\omega_0) - \nabla f(\omega_0)) \tag{15}$$

with $i \in \{1, \ldots, n\}$ picked uniformly at random, ω_0 decision point from a previous iteration; note that computing $\tilde{\nabla} f(\omega; \omega_0)$ requires only two standard stochastic gradient evaluation, because $\nabla f(\omega_0)$ has been pre-computed before.
Again, starting from a standard Frank-Wolfe algorithm, the algorithm substitutes the exact gradient with a mini-batch approximation and takes snapshots every once in a while.

Algorithm 3 *Stochastic Variance-Reduced Frank-Wolfe method (SVRF)*

Input: objective function $f = \frac{1}{n}\sum_{i=1}^n f_i$.
Input: γ_k, m_k *and* N_t.
Initialize: $\omega_0 = \arg\min_{\omega \in \Omega} tr\left(\nabla f(x)^\top \omega\right)$ for an arbitrary $x \in \Omega$.
for $t = 1, 2, \ldots, T$ **do**
  Take snapshot: $x_0 = \omega_{t-1}$ and compute $\nabla f(x_0)$;
  for $k = 1, \ldots, N_t$ **do**
    Compute $\tilde{\nabla}_k$ the average of m_k iid samples of $\tilde{\nabla} f(x_{k-1}; x_0)$;
    Set $v_k = \operatorname{argmin}_{v \in \Omega} tr\left(\tilde{\nabla}_k^T v\right)$;
    Set $x_k = x_{k-1} + \gamma_k(v_k - x_{k-1})$;
  end for
  Set $\omega_t = x_{N_t}$.
end for

Given the following choice of parameters:

$$\gamma_k = \frac{2}{k+1}, \quad m_k = 96(k+1), \quad N_t = 2^{t+3} - 2 \tag{16}$$

to achieve $1-\varepsilon$ accuracy the method requires [5]:

- $\mathcal{O}\left(\ln \frac{LD^2}{\varepsilon}\right)$ exact gradients;
- $\mathcal{O}\left(\frac{L^2D^4}{\varepsilon^2}\right)$ stochastic gradients;
- $\mathcal{O}\left(\frac{LD^2}{\varepsilon}\right)$ linear optimizations.

Let's now prove the convergence of the method with the following:

Theorem 2.2. *With the parameters chosen as above the algorithm ensures:*

$$\mathbb{E}[f(\omega_t) - f(\omega*)] \leq \frac{LD^2}{2^{t+1}} \quad \forall t. \tag{17}$$

Proof. By induction. For $t = 0$, by smoothness, the optimality of ω_0 and convexity we have:

$$f(\omega_0) \leq f(\omega^*) + \frac{LD^2}{2}. \tag{18}$$

Assuming $\mathbb{E}[f(\omega_k) - f(\omega*)] \leq \frac{LD^2}{2^t}$, we consider iteration t of the algorithm and use another induction to show that $\mathbb{E}[f(x_k) - f(\omega*)] \leq \frac{4LD^2}{k+2} \quad \forall k \leq N_t$. The base case is trivial since $x_0 = \omega_{t-1}$. Suppose $\mathbb{E}[f(x_{s-1}) - f(\omega*)] \leq \frac{4LD^2}{s+1} \quad \forall s \leq k$. Now because $\tilde{\nabla}_s$ is the average of m_k iid samples of stochastic gradient $\tilde{\nabla} f_i(x_{s-1}; x_0)$, its variance is reduced by a factor of m_s. With Lemma 1 present in [5] we have $\mathbb{E}\left[\left\|\tilde{\nabla}_s - \nabla f(x_{s-1})\right\|^2\right] \leq \frac{6L}{m_s}\left(\frac{8LD^2}{s+1} + \frac{8LD^2}{s+1}\right) = \frac{L^2D^2}{(s+1)^2}$, given by $s \leq N_t = 2^{t+3} - 2$ and the choice of m_s. Then the condition of Lemma 2 [5] is satisfied and the induction is completed. Finally with the choice of N_t we prove $\mathbb{E}[f(\omega_t) - f(\omega^*)] \leq \frac{LD^2}{2^{t+1}}$. □

2.4 Stochastic Conditional Gradient Sliding (SCGS)

The Conditional Gradient Sliding algorithm (CGS), presented in [4], is a projection free method which can skip the computation for the gradient of f from time to time when performing LO over the feasible region. The basic scheme of the CGS method is obtained by applying the classic FW method to solve the projection subproblems existing in the Accelerated Gradient (AG) method[4] approximately. The Stochastic Conditional Gradient Sliding algorithm (SCGS), also presented in [4], is the stochastic counterpart of the above algorithm, where only a mini-batch of stochastic gradients is computed instead of the full gradient.

The pseudocode is the following:

Algorithm 4 *Stochastic Conditional Gradient Sliding method (SCGS)*

Input: objective function $f = \frac{1}{n}\sum_{i=1}^{n} f_i$.
Input: Initial point $x_0 \in \Omega$, initialise $x_0 = y_0$ and number of iterations N.
Input: $\gamma_k, \beta_k, \eta_k,$ *and* m_k.
for $k = 1, \ldots, N$ **do**
    Set $z_k = y_{k-1} + \gamma_k(x_{k-1} - y_{k-1})$;
    Compute the average of m_k iid samples of $\nabla f(z_k)$, so $\nabla f_{m_k}(z_k) = \sum_{j=1}^{m_k} \nabla f_j(z_k)$;
    Set $x_k = CndG(\nabla f_{m_k}(z_k), x_{k-1}, \beta_k, \eta_k)$;
    Set $y_k = y_{k-1} + \gamma_k(x_k - y_{k-1})$;
    Set $\omega_t = y_N$.
end for

Where the *CndG* procedure is a specialized version of the standard FW algorithm, and is defined as follows:

Sub-routine *CndG Procedure*

procedure $u^+ = CndG(g, u, \beta, \eta)$
    Set $u_1 = u$ and $t = 1$.
    Let v_t be an optimal solution for the subproblem of

$$V_{g,u,\beta}(u_t) := \max_{x \in \Omega} \langle g + \beta(u_t - u), u_t - x \rangle; \tag{19}$$

    if $V_{g,u,\beta}(u_t) \leq \eta$ **then**
        Set $u^+ = u_t$ and **terminate** the procedure;
    else
        Set $u_{t+1} = (1 - \alpha_t)u_t + \alpha_t v_t$ with

$$\alpha_t = \min\left\{1, \frac{\langle \beta(u - u_t) - g, v_t - u_t \rangle}{\beta \|v_t - u_t\|^2}\right\}$$

    ;
        Set $t = t + 1$ and go to step 3.
    end if
end procedure

Note that, defining $\phi(x) := \langle g, x \rangle + \beta\|x - u\|^2/2$ the *CndG* procedure is a specialized version of the FW method applied to $\min_{x \in X} \phi(x)$. In particular, it can be easily seen that $V_{g,u,\beta}(u_t)$ in (19) is equivalent to

$\max_{x\in X} \langle \phi'(u_t), u_t - x\rangle$, which is often called the Wolfe gap, and the CndG procedure terminates whenever $V_{g,u,\beta}(u_t)$ is smaller than the prespecified tolerance η. Also note that the selection of α_t in (2.5) explicitly solves

$$\alpha_t = \operatorname*{argmin}_{\alpha\in[0,1]} \phi((1-\alpha)u_t + \alpha v_t)$$

Finally we can easily see that x_k obtained in step 7 of the SCGS algorithm is an **approximate** solution for the projection subproblem

$$\min_{x\in X}\left\{\phi_k(x) := \langle f'(z_k), x\rangle + \frac{\beta_k}{2}\|x - x_{k-1}\|^2\right\} \tag{20}$$

such that

$$\langle \phi_k'(x_k), x_k - x\rangle = \langle f'(z_k) + \beta_k(x_k - x_{k-1}), x_k - x\rangle \le \eta_k \quad \forall x \in \Omega$$

for some $\eta_k \ge 0$. This is the main difference from the original AG method where the x_k is set to the exact solution of (20) (i.e. $\eta_k = 0$).

Given the following choice of parameters:

$$\gamma_k = \frac{3}{k+2},\ \ \beta_k = \frac{4L}{k+2},\ \ \eta_k = \frac{LD^2}{k(k+1)}\ \ and\ \ m_k = \left\lceil \frac{\sigma^2(k+2)^3}{L^2D^2}\right\rceil \tag{21}$$

where σ^2 is an upper bound for the variance of the stochastic gradients, to achieve $1-\varepsilon$ accuracy the method requires[5]:

- $\mathcal{O}\left(\sqrt{\frac{LD^2}{\varepsilon}} + \frac{\sigma^2D^2}{\epsilon^2}\right)$ stochastic gradients;
- $\mathcal{O}\left(\frac{LD^2}{\varepsilon}\right)$ linear optimizations.

Let's now prove the convergence of the method with the following:

Theorem 2.3. *With the parameters chosen as above the algorithm ensures:*

$$\mathbb{E}[f(\omega_t) - f(\omega*)] \le \frac{6LD^2}{(k+2)^2} + \frac{9LD^2}{(k+1)(k+2)} \quad \forall k \ge 1. \tag{22}$$

And the number of stochastic gradients and linear optimizations are the same as above.

Proof. The parameters satisfy $\gamma_1 = 1$ and $L\gamma_k \le \beta_k \quad \forall k \ge 1$; also, calling $\Gamma_k = \frac{6}{k(k+1)(k+2)}$ it holds $\frac{\beta_k\gamma_k}{\Gamma_k} = \frac{2Lk(k+1)}{k+2}$ and then it is easy to check that:

$$\sum_{i=1}^{k}\frac{\eta_i\gamma_i}{\Gamma_i} \le \frac{kLD^2}{2}, \sum_{i=1}^{k}\frac{\gamma_i}{\Gamma_i B_i(\beta_i - L\gamma_i)} \le \frac{kLD^2}{2\sigma^2} \tag{23}$$

under the assumptions 3.1 and 3.2 in [4].

Hence from Theorem 3.1 in [4], the inequality 22 is obtained and it implies that the total number of outer iterations can be bounded by $\mathcal{O}\left(\sqrt{\frac{LD^2}{\varepsilon}}\right)$, if we assume that $\mathbb{E}\left[\|G(z_k,\xi_k) - f'(z_k)\|_*^2\right] \le \sigma^2$, where $\mathbb{E}[G(z_k,\xi_k)] = f'(z_k)$.
Then the bounds we want to prove immediately follow from this and the fact that the number of stochastic gradients and linear optimizations is bounded by $\frac{\sigma^2(N+3)^4}{4L^2D^2} + N$ and $12N^2 + 13N$. □

2.5 Stochastic Variance Reduced Conditional Gradient Sliding (STORC)

This algorithm applies variance reduction to the SCGS algorithm, so it replaces the stochastic gradients with the average of a mini-batch of variance-reduced stochastic gradients and takes snapshots every once in a while. The pseudocode is given below:

Algorithm 5 *Stochastic Variance Reduced Conditional Gradient Sliding method (STORC)*

Input: objective function $f = \frac{1}{n}\sum_{i=1}^{n} f_i$.
Input: $\gamma_k, \beta_k, \eta_{t,k},\ m_{t,k}$ *and* N_t.
Initialize: $\omega_0 = \underset{\omega\in\Omega}{\operatorname{argmin}}\ tr\left(\nabla f(x)^\top \omega\right)$ for an arbitrary $x \in \Omega$.
for $t = 1, 2, \ldots, T$ **do**
  Take snapshot: $y_0 = \omega_{t-1}$ and compute $\nabla f(y_0)$;
  Initialize $x_0 = y_0$;
  for $k = 1, \ldots, N_t$ **do**
    Set $z_k = y_{k-1} + \gamma_k(x_{k-1} - y_{k-1})$;
    Compute $\tilde{\nabla}_k$ the average of $m_{t,k}$ iid samples of $\tilde{\nabla} f(z_k; y_0)$;
    Set $x_k = CndG(\tilde{\nabla}_k, x_{k-1}, \beta_k, \eta_{t,k})$;
    Compute $y_k = y_{k-1} + \gamma_k(x_k - y_{k-1})$;
  end for
  Set $\omega_t = y_{N_t}$.
end for

Given the following choice of parameters:

$$\gamma_k = \frac{2}{k+1},\ \ \beta_k = \frac{3L}{k},\ \ \eta_{t,k} = \frac{2LD_t^2}{N_t k} \tag{24}$$

and calling D_t any constant such that $D_t \leq D$, to achieve $1-\varepsilon$ accuracy the method requires:

- $\mathcal{O}\left(\ln \frac{LD^2}{\varepsilon}\right)$ exact gradients [5];
- 1. If $\nabla f(\omega^*) = 0$ and $D_t = D,\ N_t = \lceil 2^{\frac{t}{2}+2} \rceil,\ m_{t,k} = 900 N_t \Rightarrow \mathcal{O}\left(\frac{LD^2}{\varepsilon}\right)$ stochastic gradients;
  2. If f is G-Lipschitz and $D_t = D,\ N_t = \lceil 2^{\frac{t}{2}+2} \rceil,\ m_{t,k} = 700 N_t + \frac{24 N_t G(k+1)}{LD} \Rightarrow \mathcal{O}\left(\frac{LD^2+}{\varepsilon} + \frac{\sqrt{L}D^2 G}{\varepsilon^{1.5}}\right)$ stochastic gradients;
  3. If f is α-strongly convex and $D_t^2 = \frac{\mu D^2}{2^{t-1}},\ N_t = \lceil \sqrt{32\mu} \rceil,\ m_{t,k} = 5600 N_t \mu \Rightarrow \mathcal{O}\left(\mu^2 \ln \frac{LD^2}{\varepsilon}\right)$ stochastic gradients;
- $\mathcal{O}\left(\frac{LD^2}{\varepsilon}\right)$ linear optimizations.

Theorem 2.4. *With the parameters said above, the algorithm ensures:*

$$\mathbb{E}\left[f\left(w_t\right) - f\left(w^*\right)\right] \leq \frac{LD^2}{2^{t+1}} \quad \forall t \tag{25}$$

if any of the following three cases holds:

1. $\nabla f\left(w^*\right) = \mathbf{0}$ *and* $D_t = D, N_t = \left\lceil 2^{\frac{t}{2}+2} \right\rceil, m_{t,k} = 900 N_t$;
2. f *is* G *-Lipschitz and* $D_t = D, N_t = \left\lceil 2^{\frac{t}{2}+2} \right\rceil, m_{t,k} = 700 N_t + \frac{24 N_t G(k+1)}{LD}$;
3. f *is* α *-strongly convex and* $D_t^2 = \frac{\mu D^2}{2^{t-1}}, N_t = \lceil \sqrt{32\mu} \rceil, m_{t,k} = 5600 N_t \mu$ *where* $\mu = \frac{L}{\alpha}$.

Proof. By induction, very similar to proof of Theorem 2.2. The base case $t = 0$ holds by the exact same argument. Suppose $\mathbb{E}\left[f\left(w_{t-1}\right) - f\left(w^*\right)\right] \leq \frac{LD^2}{2^t}$ and consider iteration t. Again by induction we can prove $\mathbb{E}\left[f\left(y_k\right) - f\left(w^*\right)\right] \leq \frac{8LD_t^2}{k(k+1)}$ for any $1 \leq k \leq N_t$.
From Lemma 1 [5] we can write: $\mathbb{E}\left[\left\|\nabla_s - \nabla f\left(z_s\right)\right\|^2\right] \leq \frac{6L}{m_{t,s}}\left(2\mathbb{E}\left[f\left(z_s\right) - f\left(w^*\right)\right] + \mathbb{E}\left[f\left(y_0\right) - f\left(w^*\right)\right]\right)$ so the key is to bound $\mathbb{E}\left[f\left(z_s\right) - f\left(w^*\right)\right]$. It can be easily proven that $\mathbb{E}\left[f\left(y_s\right) - f\left(w^*\right)\right] \leq \frac{8LD_t^2}{s(s+1)}$ holds for $s = 1$ by Lemma 3 [5] for all three cases. Now suppose that holds for any $s < k$ and we will discuss the three cases separately for $s = k$.

1. By smoothness, the condition $\nabla f\left(w^*\right) = 0$, the construction of $z_s = y_{s-1}$, and some inequalities, we can get, for any $1 < s \leq k$: $\mathbb{E}\left[f\left(z_s\right) - f\left(w^*\right)\right] < \frac{55LD^2}{(s+1)^2}$. On the other hand $\mathbb{E}\left[f\left(y_0\right) - f\left(w^*\right)\right] \leq \frac{40LD^2}{(s+1)^2}$ so the choice of $m_{t,s}$ ensures the condition of Lemma 3 [5] and thus completing the induction.

2. Using the G-Lipschitz condition, previous bounds and the chosen $m_{t,s}$ we proceed similarly until we can bound $\mathbb{E}\left[\left\|\tilde{\nabla}_s - \nabla f(z_s)\right\|^2\right] < \frac{L^2 D^2}{N_t(s+1)^2}$ again completing the induction.

3. Following more or less the same steps we can get to $\mathbb{E}\left[\left\|\tilde{\nabla}_s - \nabla f(z_s)\right\|^2\right] \leq \frac{6L}{m_{t,s}}\left(\frac{896\mu L D_t^2}{(s+1)^2} + \frac{32 L D_t^2}{(s+1)^2}\right)$ that with the chosen $m_{t,s}$ completes the induction by Lemma 3 [5].

□

2.6 Online Frank-Wolfe (OFW)

Up to now we studied algorithms in the setting of offline convex optimization, meaning that in our problem the cost function is known in advance. An example of a online convex programming problem could be the following[1]:

> Imagine a farmer who decides what to plant each year. She has certain restrictions on her resources, both land and labour, as well as restrictions on the output she is allowed to produce. How can she select which crops to grow without knowing in advance what the prices will be? Here we present an algorithm $[\cdots]$ which will earn her almost as much as she could given that she knew all prices in advance but produced the same amount of each crop year after year.

Definition 1. *An **online convex programming problem** consists of a feasible set $\Omega \subseteq \mathbb{R}^n$ and an infinite sequence $\{f_1, f_2, \dots\}$ where each $f_t : \Omega \to \mathbb{R}$ is a convex function. At each time step t, an online convex programming algorithm selects a vector $\omega_t \in \Omega$. After the vector is selected, it receives the cost function f_t.*

Because all information is not available before decisions are made, online algorithms do not reach "solutions": we therefore need a different way of evaluating the performance of the algorithm. Specifically, we measure the performance of an algorithm in comparison to the best algorithm in hindsight that knows all of the cost functions and selects one fixed vector.

Definition 2. *Given an algorithm A, and a convex programming problem $(\Omega, \{f_1, f_2, \dots\})$, if $\{\omega_1, \omega_2, \dots\}$ are the vectors selected by A, then the cost of A until time T is*

$$F_A(T) = \sum_{t=1}^{T} f_t(\omega_t)$$

The cost of a static feasible solution $\omega \in \Omega$ until time T is

$$F_\omega(T) = \sum_{t=1}^{T} f_t(\omega)$$

We can therefore define the **regret**[3] of algorithm A until time T:

$$R_A(T) \coloneqq F_A(T) - \min_{\omega \in \Omega} F_\omega(T)$$

The goal of the learner is to produce points ω_t such that the regret is sublinear in T.
Note that if the cost functions are stochastic, the regret is measured using $f = \mathbb{E}[f_t]$. We define again the diameter of Ω using (1) as we made previously in FW algorithm.
Also, we say that $\omega \in \Omega$ is t-sparse w.r.t Ω if it can be written as a convex combination of t boundary points of Ω. The special feature of this algorithm is that there is no parameter for the learning rate, this makes it easy to implement since there is no parameter tuning needed.

The pseudocode of the method is:

Algorithm 6 *Online Frank-Wolfe method (OFW)*

Input: constant $a \geq 0$;
Initialize: $\omega_0 \in \Omega$ arbitrarily;
for $t = 0, 1, 2, \dots, T$ **do**
    Play ω_t and observe f_t;
    Compute $F_t = \frac{1}{t}\sum_{\tau=1}^{t} f_\tau$;
    Set $\hat{v}_t = \underset{v \in \Omega}{\operatorname{argmin}}\ tr\left(\nabla F_t(w_t)^\top v\right)$;
    Set $\omega_{t+1} = \omega_t + t^{-a}(\hat{v}_t - \omega_t)$;
end for

Define $\Delta_t = F_t(\mathbf{x}_t) - F_t(\mathbf{x}_t^*)$, where $F_t = \frac{1}{t}\sum_{\tau=1}^{t} f_\tau$ as defined in step 1 of the algorithm, and $\mathbf{x}_t^* = \arg\min_{\mathbf{x}\in\Omega} F_t(\mathbf{x})$. We now see that the regret Δ_t is sublinear in t.

Theorem 2.5. *Assume that for $t = 1, 2, \ldots, T$, the function f_t is L-Lipschitz, Bt^{-b}-smooth for some constants $b \in [-1, 1/2]$ and $B \geq 0$, and St^{-s} -strongly convex for some constants $s \in [0, 1)$ and $S \geq 0$. Then we have*

$$\Delta_t \leq Ct^{-d} \quad \forall t > 1$$

for both the following values of C and d :

$$(C, d) = \left(\max\left\{9D^2B, 3LD\right\}, \tfrac{1+b}{2}\right)$$
$$\text{and } (C, d) = \left(\max\left\{9D^2B, 36L^2/S, 3LD\right\}, \tfrac{2+2b-s}{3}\right)$$

In either case, this bound is obtained by setting $a = d - b$.

Proof. First, we note that $d \leq 1$ and $a \geq 0$ in either case, so that $t^{-a} \in [0,1]$ and hence all iterates lie in Ω. We now proceed by induction on t, for either values of C and d. The statement is true for $t = 1$ since f_1 is L -Lipschitz, so

$$C \geq LD \geq L\|\mathbf{x}_1 - \mathbf{x}_1^*\| \geq f_1(\mathbf{x}_1) - f_1(\mathbf{x}_1^*) = \Delta_1$$

So, assume that for some $t \geq 1$, we have $\Delta_t \leq Ct^{-d}$. Now by convexity of F_t we have

$$F_t(\mathbf{x}_t^*) \geq F_t(\mathbf{x}_t) + \nabla F_t(\mathbf{x}_t) \cdot (\mathbf{x}_t^* - \mathbf{x}_t)$$

Thanks to convexity and to the definition of $\mathbf{v}_t$ (see step 6 of OFW) we have

$$(\mathbf{v}_t - \mathbf{x}_t) \cdot \nabla F_t(\mathbf{x}_t) \leq -\Delta_t \tag{26}$$

Now, knowing that the smoothness of F_t is bounded by $3Bt^{-b}$ for $b \leq 1/2$, and using (26) we have

$$\begin{aligned} F_t(\mathbf{x}_{t+1}) &\leq F_t(\mathbf{x}_t) + t^{-a}(\mathbf{v}_t - \mathbf{x}_t) \cdot \nabla F_t(\mathbf{x}_t) + 3D^2Bt^{-b-2a} \\ &\leq F_t(\mathbf{x}_t) - t^{-a}\Delta_t + 3D^2Bt^{-b-2a} \end{aligned}$$

Using the induction hypothesis, the fact that $a = d - b$ and the inequality $F_t(\mathrm{x}_t^*) \leq F_t\left(\mathrm{x}_{t+1}^*\right)$ in the bound above we get:

$$F_t(\mathrm{x}_{t+1}) - F_t\left(\mathrm{x}_{t+1}^*\right) \leq Ct^{-d} - \frac{2}{3}Ct^{b-2d} \tag{27}$$

since $C \geq 9D^2B$. Thanks to Lemma 3.1 in [3] we also have that:

$$f_{t+1}(\mathrm{x}_{t+1}) - f_{t+1}\left(\mathrm{x}_{t+1}^*\right) \leq \frac{2}{3}Ct^{1+b-2d}$$

Multiplying (27) by t, adding the above bound, and dividing by $t + 1$, we get

$$F_{t+1}(\mathbf{x}_{t+1}) - F_{t+1}\left(\mathbf{x}_{t+1}^*\right) \leq \frac{t}{t+1}Ct^{-d} \leq C(t+1)^{-d}$$

since $d \leq 1$, thus completing the induction. □

3 Datasets

To test the algorithms we just described, three un-normalized datasets were chosen and from each of them we required that the product $(\#features) \cdot (\#classes)$ was at least of the order of 5×10^5. They are the following:

1. **Fashion MNIST**[1]: a dataset of Zalando's article images made of $60,000$ samples. Each one of them is a 28x28 grayscale image, so it has 784 features, associated with a label from 10 classes. Each class has the same number of images, so the dataset is balanced.

2. **Street View House Numbers (SVHN)**[2]: a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN samples are obtained from house numbers through Google Street View images.
There are 10 classes, one for each digit. The Training set is made up of 73257 digits and each sample is 3D tensor 32x32x3 where the last dimension is the one for the color channel. For practicality reasons we only kept one color channel, hence the total number of features in our case is 1024.

3. **small NORB**[3]: intended for recognising 3D object from their shape. For each instance, from two cameras, it contains a pair of 96x96 grayscale images for two different channels. We used a downsampled version of the dataset, where each channel of the origin data is reduced to 32x32 by selecting the maximum pixel value within every 3x3 disjoint region. It has 5 classes, 2048 features and 18.432 instances.

dataset	$\#features$	$\#classes$	$\#samples$
Fashion MNIST	784	10	60000
SVHN	1024	10	73257
small NORB	2048	5	18432

[1] `https://github.com/zalandoresearch/fashion-mnist`
[2] `http://ufldl.stanford.edu/housenumbers/`
[3] `https://cs.nyu.edu/~ylclab/data/norb-v1.0-small/`

4 Results

Here we presents the results obtained: it's worth noticing that we didn't implement any stopping condition, instead we decided to fix a running time for all of the algorithms in order to ease the comparison.
In this specific case we set $t = 180\,\text{s}$ and $\tau = 50$.

4.1 Fashion MNIST

The parameters used are reported in Table 2 while in Fig. 1 there is the *Time vs Loss* plot.

Algorithm	γ_k	$\gamma_k^{sub-rutine}$	*Batch Size*	*Epochs*	L	D	β_k	η_k	s_k
FW	1×10^{-4}								
SFW	1×10^{-4}		$iter^2$						
SVRF	1×10^{-4}		$iter$	50					
SCGS	1×10^{-4}	0.05	$iter^3$		1	50	$4L/(iter+2)$	$1/iter^2$	
STORC	1.5×10^{-4}	0.05	100	50	1	50	$3L/iter$	$1/iter$	
SGM	1×10^{-4}		100						0.5
SVRG	1×10^{-4}		100	50					0.5

Table 2: Summary of parameters used for **Fashion MNIST**.

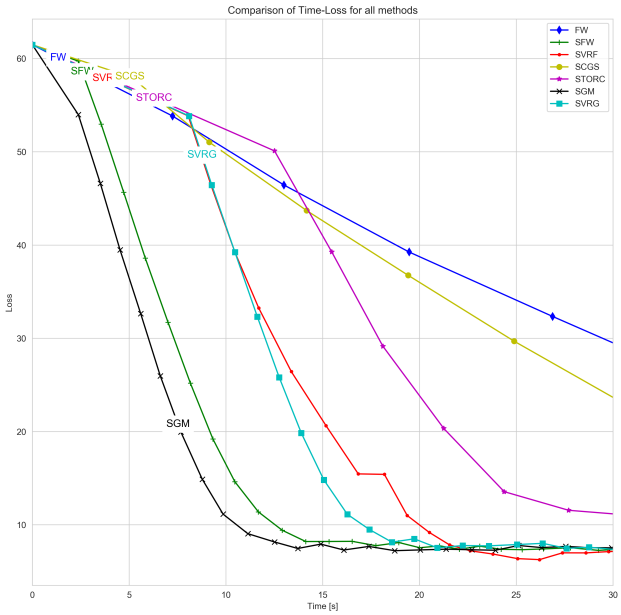

Figure 1: **Fashion MNIST**: Time vs Loss.

4.2 SVHN

The parameters used are reported in Table 3 while in Fig. 2 there is the *Time vs Loss* plot.

Algorithm	γ_k	$\gamma_k^{sub-rutine}$	*Batch Size*	*Epochs*	L	D	β_k	η_k	s_k
FW	1×10^{-4}								
SFW	1×10^{-4}		$iter^2$						
SVRF	1×10^{-4}		$iter$	50					
SCGS	1×10^{-4}	0.1	$iter^3$		1	50	$4L/(iter+2)$	$1/iter^2$	
STORC	1×10^{-4}	0.5	100	50	1	50	$3L/iter$	$1/iter$	
SGM	1×10^{-4}		100						0.5
SVRG	1×10^{-4}		100	50					0.5

Table 3: Summary of parameters used for **SVHN**.

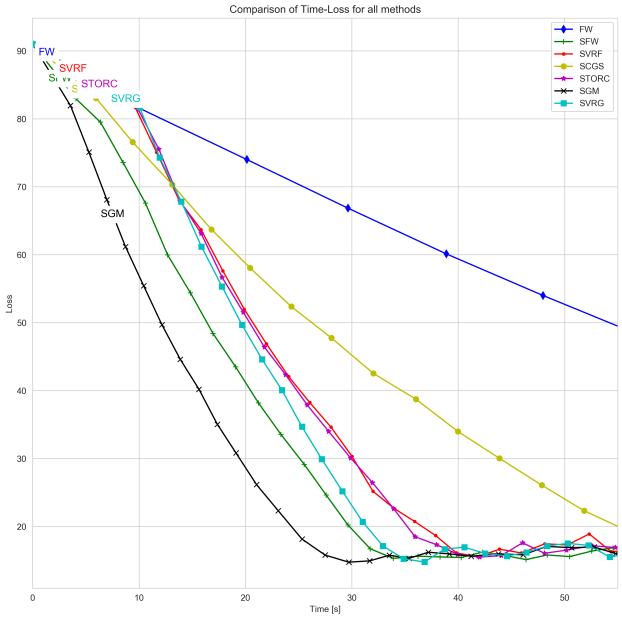

Figure 2: **SVHN**: Time vs Loss.

4.3 small NORB

The parameters used are reported in Table 4 while in Fig. 3 there is the *Time vs Loss* plot.

Algorithm	γ_k	$\gamma_k^{sub-rutine}$	*Batch Size*	*Epochs*	L	D	β_k	η_k	s_k
FW	2×10^{-5}								
SFW	1×10^{-5}		$iter^2$						
SVRF	1×10^{-5}		$iter$	50					
SCGS	1.5×10^{-5}	0.3	$iter^3$		1	50	$4L/(iter+2)$	$1/iter^2$	
STORC	2×10^{-5}	0.3	100	50	1	50	$3L/iter$	$1/iter$	
SGM	1×10^{-5}		100						0.5
SVRG	1×10^{-5}		100	50					0.5

Table 4: Summary of parameters used for **smallNORB**.

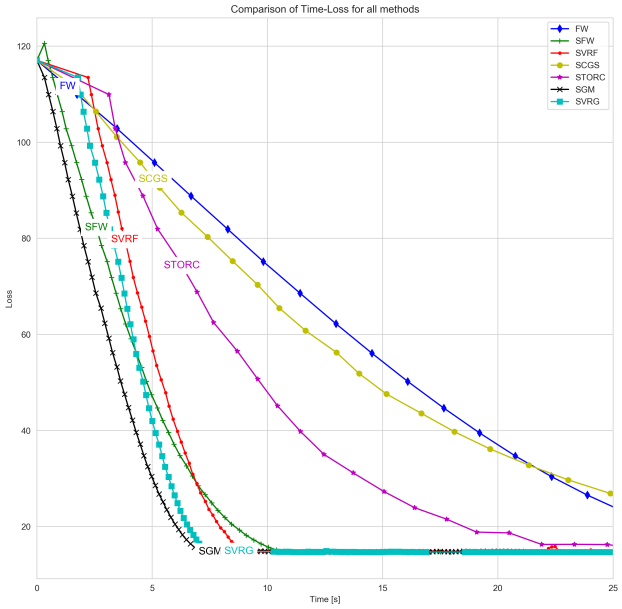

Figure 3: **smallNORB**: Time vs Loss.

As we can see from the images above (Fig. 1, Fig. 2, Fig. 3), both the stochastic (SGM and SFW) and variance reduced (SVRG and SVRF) algorithms, and more importantly the projected and projection-free methods performs really close to each other. This is probably due to the fact that in this configuration, the CPU time needed to compute the projection (which translates into the entire SVD), is still less than the one required to solve the linear optimization problem of Franke-Wolfe's methods (which requires the computation of only the top singular vectors).
The dimensions of the datasets, as well as the ones of the weights, play a central role in our performances: to ensure a significant difference between the two methods, way bigger datasets are required: $(\#features) \cdot (\#classes)$ should be at least one order of magnitude bigger.

References

[1] Martin Zinkevich. "Online Convex Programming and Generalized Infinitesimal Gradient Ascent". In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML'03. Washington, DC, USA: AAAI Press, 2003, pp. 928–935. ISBN: 1577351894.

[2] Martin Jaggi. "Sparse convex optimization methods for machine learning". In: 2011.

[3] Elad Hazan and Satyen Kale. "Projection-Free Online Learning". In: ICML'12 (2012), pp. 1843–1850.

[4] Dan Garber and Elad Hazan. *A Linearly Convergent Conditional Gradient Algorithm with Applications to Online and Stochastic Optimization*. 2013. arXiv: `1301.4666 [cs.LG]`.

[5] Elad Hazan and Haipeng Luo. "Variance-Reduced and Projection-Free Stochastic Optimization". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 1263–1271.