Variance Reduction in Projection-Free Optimization

Optimization For Data Science Final Project

Gianmarco Cracco, Alessandro Manente, Riccardo Mazzieri

30th June 2020

Università
degli Studi
di Padova

Overview

Intro

- The Frank-Wolfe optimization algorithm has recently regained popularity in Big Data applications thanks to its projection-free property.
- 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 see the main theoretical results and [illegible] of those algorithms applied on real world datasets.

Definitions

Definition (Diameter)

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

Definition (L-smoothness (or Lipschitz Continuous Gradient))

f, a convex function, is L-smooth in Ω if for any $v, w \in \Omega$,

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

Definition (G-Lipschitz)

f is G-Lipschitz if $\forall w \in \Omega$ we have $\|\nabla f_i(w)\| \leq G$.

Definitions

Definition (Diameter)

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

Definition (L-smoothness (or Lipschitz Continuous Gradient))

f, a convex function, is L-smooth in Ω if for any $v, w \in \Omega$,

$$f(v + w) < f(v) + \nabla f(v) \qquad \frac{L}{2}$$

Definition (G-Lipschitz)

f is G-Lipschitz if $\forall w \in \Omega$ we have $\|\nabla f_i(w)\| \leq G$.

Definitions

Definition (Diameter)

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

Definition (L-smoothness (or Lipschitz Continuous Gradient))

f, a convex function, is L-smooth in Ω if for any $v, w \in \Omega$,

$$f(v + w) < f(v) + \nabla f(v) \qquad \frac{L}{2}$$

Definition (G-Lipschitz)

f is G-Lipschitz if $\forall w \in \Omega$ we have $\|\nabla f_i(w)\| \leq G$.

The Problem

- We consider a multiclass classification problem, with a given training set $(e_i, y_i)_{i=1,\dots,n}$
- Our costrain set will be $\Omega := \{\omega \in \mathbb{R}^{h\times m} : \|\omega\|_* \leq \tau\}$, where $\|\cdot\|_*$ is the *trace norm*;
- Our aim is to find a matrix $\omega = [\omega_1, \dots, \omega_h]^\top \in \mathbb{R}^{h\times m}$ such that:

$$y_i = \operatorname*{argmax}_{\ell} (\omega_\ell^\top e_i) \quad \forall i. \tag{3}$$

The Problem

This translates into a problem with the following form:

$$\min_{\omega \in \Omega} f(\omega) = \min_{\omega \in \Omega} \frac{1}{n} \sum_{i=1}^{n} f_i(\omega) \tag{4a}$$

$$f_i(\omega) = \log \left(1 + \sum_{\ell \neq y_i} \exp \left(\omega_\ell^\top e_i - \omega_{y_i}^\top e_i \right) \right) \tag{4b}$$

note that f_i is convex and L-smooth $\forall \omega \in \Omega$.

The Problem

From now on the descend direction will be defined as:

$$\hat{v} = \underset{v \in \Omega}{\operatorname{argmin}} \nabla f\left(\omega_{k-1}\right)^{\top} v \tag{5}$$

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

$$\hat{v} = \underset{v \in \Omega}{\operatorname{argmin}}\ tr\left(\nabla f\left(\omega_{k-1}\right)^{\top} v\right) \tag{6}$$

So, the solution is given by:

$$\hat{v}_k = -\tau u_1 v_1^{\top} \tag{7}$$

where u_1, v_1 are top singular vectors of $\nabla f\left(\omega_{k-1}\right)$ and τ is the upper bound for $\|\omega\|_*$.

The Problem

From now on the descend direction will be defined as:

$$\hat{v} = \underset{v \in \Omega}{\operatorname{argmin}} \nabla f\left(\omega_{k-1}\right)^{\top} v \tag{5}$$

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

$$\hat{v} = \underset{v \in \Omega}{\operatorname{argmin}}\ tr\left(\nabla f\left(\omega_{k-1}\right)^{\top} v\right) \tag{6}$$

So, the solution is given by:

$$\hat{v}_k = -\tau u_1 v_1^{\top} \tag{7}$$

where u_1, v_1 are top singular vectors of $\nabla f\left(\omega_{k-1}\right)$ and τ is the upper bound for $\|\omega\|_*$.

The FW 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

$$\hat{v} = \underset{v \in \Omega}{\operatorname{argmin}} \nabla f(x_k)^\top v$$

The FW pseudocode

Algorithm 1 *Frank-Wolfe method (FW)*

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

The SFW Method

This algorithm is obtained starting from FW and simply replacing $\nabla f(\omega_{k-1})$ with some $\nabla f_{m_k}(\omega_{k-1})$ where $\nabla f_{m_k}(\cdot)$ is the gradient computed only w.r.t to m_k samples with indexes $\{i_1, \ldots, i_{m_k}\} \subset \{1, \ldots, n\}$:

$$\nabla f_{m_k}(\omega_{k-1}) := \frac{1}{m_k} \sum_{j=1}^{m_k} \nabla f_{i_j}(\omega_{k-1})$$

The SFW pseudocode

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

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

SFW Convergence

Theorem

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,

$$\mathbb{E}[f(\omega_k) - f(\omega*)] \leq \frac{4LD^2}{k+2}. \tag{9}$$

To achieve $1-\varepsilon$ accuracy the method requires, if the functions are G-Lipschitz:

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

The SVRF Method

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

$$\tilde{\nabla} f(\omega; \omega_0) := \nabla f_i(\omega) - (\nabla f_i(\omega_0) - \nabla f(\omega_0)) \tag{10}$$

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.

The SVRF Method

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

$$\tilde{\nabla} f(\omega; \omega_0) := \nabla f_i(\omega) - (\nabla f_i(\omega_0) - \nabla f(\omega_0)) \tag{10}$$

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.

The SVRF pseudocode

Algorithm 3 *SVR 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**
        $\tilde{\nabla}_k$ the average of m_k iid samples of $\tilde{\nabla} f(x_{k-1}; x_0)$;
        Set $v_k = \arg\min_{v\in\Omega} tr\left(\tilde{\nabla}_k^T v\right)$;
        Set $x_k = x_{k-1} + \gamma_k(v_k - x_{k-1})$;
    end for
    Set $\omega_t = x_{N_t}$.
end for

SVRF Convergence

Theorem

Given the following choice of parameters:

$$\gamma_k = \frac{2}{k+1}, \quad m_k = 96(k+1), \quad N_t = 2^{t+3} - 2 \tag{11}$$

the algorithm ensures:

$$\mathbb{E}[f(\omega_t) - f(\omega*)] \leq \frac{LD^2}{2^{t+1}} \quad \forall t. \tag{12}$$

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

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

The CGS Method

- The Conditional Gradient Sliding algorithm (CGS) is a projection free method similar in spirit to the Accelerated Gradient method (AG) by Nesterov.
- The basic idea is to apply the classic FW method to solve the projection subproblems existing in AG method approximately, where the quality of the solutions is measured via the Wolfe Gap;
- This allows us to save a lot of calls to the FO oracle.

The CGS Method

- The Conditional Gradient Sliding algorithm (CGS) is a projection free method similar in spirit to the Accelerated Gradient method (AG) by Nesterov.
- The basic idea is to apply the classic FW method to solve the projection subproblems existing in AG method approximately, where the quality of the solutions is measured via the Wolfe Gap;
- This allows us to save a lot of calls to the FO oracle.

The CGS Method

- The Conditional Gradient Sliding algorithm (CGS) is a projection free method similar in spirit to the Accelerated Gradient method (AG) by Nesterov.
- The basic idea is to apply the classic FW method to solve the projection subproblems existing in AG method approximately, where the quality of the solutions is measured via the Wolfe Gap;
- This allows us to save a lot of calls to the FO oracle.

The SCGS Method

The Stochastic Conditional Gradient Sliding algorithm (SCGS) is the stochastic counterpart of the above algorithm, where only a mini-batch of stochastic gradients is computed instead of the full gradient.

The SCGS Pseudocode

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 $\nabla f(z_k)$: $\nabla f_{m_k}(z_k) = \sum_{j=1}^{m_k} \nabla f_j(z_k)$;
    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

CndG Procedure

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

$$V_{g,u,\beta}(u_t) := \max_{x \in \Omega} \langle g + \beta(u_t - u), u_t - x \rangle \tag{13}$$

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

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

8: Set $t = t + 1$ and go to step 3.
9: **end if**
10: **end procedure**

CndG 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 $V_{g,u,\beta}(u_t)$ is equivalent to $\max_{x\in X}\langle \phi'(u_t), u_t - x\rangle$, which is the Wolfe gap.
- Also note that the selection of α_t in (2.5) explicitly solves

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

SCGS Convergence

Theorem

Given the following choice of parameters, when f is smooth:

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

the algorithm ensures:

$$\mathbb{E}[f(\omega_t) - f(\omega*)] \leq \frac{6LD^2}{(k+2)^2} + \frac{9LD^2}{(k+1)(k+2)} \quad \forall k \geq 1. \quad (14)$$

SCGS Convergence

- The total number of calls to the SFO and LO oracles can be bounded respectively by:

$$\mathcal{O}\left(\sqrt{\frac{LD^2}{\epsilon}+\frac{\sigma^2 D^2}{\epsilon^2}}\right) \quad \textit{and} \quad \mathcal{O}\left(\frac{LD^2}{\epsilon}\right) \tag{15}$$

- Also, when f is strongly convex, the bound on the SFO oracles can be reduced to:

$$\mathcal{O}\left(\frac{1}{\epsilon}\right)$$

.

The STORC Method

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 STORC Pseudocode

Algorithm 5 *SVR 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: $\omega_0 = \underset{\omega\in\Omega}{\text{argmin}}\ tr\left(\nabla f(x)^\top \omega\right)$ 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 $z_k = y_{k-1} + \gamma_k(x_{k-1} - y_{k-1})$;
        $\tilde{\nabla}_k$ the average of $m_{t,k}$ iid samples of $\nabla f(z_k; y_0)$;
        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

STORC Convergence

Theorem

Given the following choice of parameters:
$\gamma_k = \frac{2}{k+1}$, $\beta_k = \frac{3L}{k}$, $\eta_{t,k} = \frac{2LD_t^2}{N_t k}$, *the algorithm ensures:*

$$\mathbb{E}\left[f\left(w_t\right) - f\left(w^*\right)\right] \leq \frac{LD^2}{2^{t+1}} \forall t \tag{16}$$

if any of the following three cases holds:

1. $\nabla f\left(w^*\right) = 0$ *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* $\mu = \frac{L}{\alpha}$

STORC Convergence

Therefore, to achieve $1-\epsilon$ accuracy, the STORC algorithm requires:

- $\mathcal{O}(\log \frac{LD^2}{\epsilon})$ exact gradient evaluations;
- $\mathcal{O}(\log \frac{LD^2}{\epsilon})$ linear optimizations;
- $\mathcal{O}(\log \frac{LD^2}{\epsilon}), \mathcal{O}(\log \frac{LD^2}{\epsilon} + \frac{\sqrt{L}D^2G}{\epsilon^{1,5}})$ and $\mathcal{O}(\mu^2 \log \frac{LD^2}{\epsilon})$ stochastic gradient evaluations for case 1, 2, ...

The OFW Method Pt.1

Up to now $\Rightarrow$ offline convex optimization: in our problem the cost function is known in advance.

Example: A farmer knows in advance the restrictions on labor and land but he's not aware what is the demand for his product. After a year he has sold his items and becomes aware of his prot $\Rightarrow$ can use this to plan the next year.

Definition

An **online convex programming problem** *consi*
set $\Omega \subseteq \mathbb{R}^n$ *and an infinite sequence* $\{f_1, f_2, \ldots\}$ *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*.*

The OFW Method Pt.1

Up to now $\Rightarrow$ offline convex optimization: in our problem the cost function is known in advance.

Example: A farmer knows in advance the restrictions on labor and land but he's not aware what is the demand for his product. After a year he has sold his items and becomes aware of his prot $\Rightarrow$ can use this to plan the next year.

Definition

An **online convex programming problem** *consists of a convex set* $\Omega \subseteq \mathbb{R}^n$ *and an infinite sequence* $\{f_1, f_2, \ldots\}$ *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*.*

The OFW Method Pt.1

Up to now $\Rightarrow$ offline convex optimization: in our problem the cost function is known in advance.

Example: A farmer knows in advance the restrictions on labor and land but he's not aware what is the demand for his product. After a year he has sold his items and becomes aware of his prot $\Rightarrow$ can use this to plan the next year.

Definition

An **online convex programming problem** *consi*
set $\Omega \subseteq \mathbb{R}^n$ *and an infinite sequence* $\{f_1, f_2, \ldots\}$ *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*.*

The OFW Method Pt.2

All information is not available before decisions are made $\Rightarrow$ online algorithms do not reach *solutions*!

New way to measure performance: regret.

Definition

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

$$R_A(T) := \sum_{t=1}^{T} f_t(\omega_t) - \min_{\omega \in \Omega} \sum_{t=1}^{T} f_t(\omega)$$

The goal of the learner is to produce points ω_t such that the regret is sublinear in T.

The OFW Method Pt.2

All information is not available before decisions are made $\Rightarrow$ online algorithms do not reach *solutions*!

New way to measure performance: regret.

Definition

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

$$R_A(T) := \sum_{t=1}^{T} f_t(\omega_t) - \min_{\omega \in \Omega} \sum_{t=1}^{T} f_t(\omega)$$

The goal of the learner is to produce points ω_t such that the regret is sublinear in T.

The OFW Method Pt.2

All information is not available before decisions are made $\Rightarrow$ online algorithms do not reach *solutions*!

New way to measure performance: regret.

Definition

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

$$R_A(T) := \sum_{t=1}^{T} f_t(\omega_t) - \min_{\omega \in \Omega} \sum_{t=1}^{T} f_t(\omega)$$

The goal of the learner is to produce points ω_t such that the regret is sublinear in T.

OFW Pseudocode

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

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

OFW convergence

Theorem

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 :

$$(C, d) = \left(\max\left\{9D^2B, 3LD\right\}, \frac{1+b}{2}\right)$$

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

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

Datasets

Three un-normalized datasets with following condition:

$$(\#features) \cdot (\#classes) \text{ at least } \sim 5 \times 10^5.$$

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

Global parameters:

- $\tau = 50$;
- running time $= 180\,\text{s}$.

SGM & SVRG added for comparison!

Datasets

Three un-normalized datasets with following condition:

$$(\#features) \cdot (\#classes) \text{ at least } \sim 5 \times 10^5.$$

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

Global parameters:

- $\tau = 50$;
- running time $= 180\,\text{s}$.

SGM & SVRG added for comparison!

Datasets

Three un-normalized datasets with following condition:

$$(\#features) \cdot (\#classes) \text{ at least } \sim 5 \times 10^5.$$

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

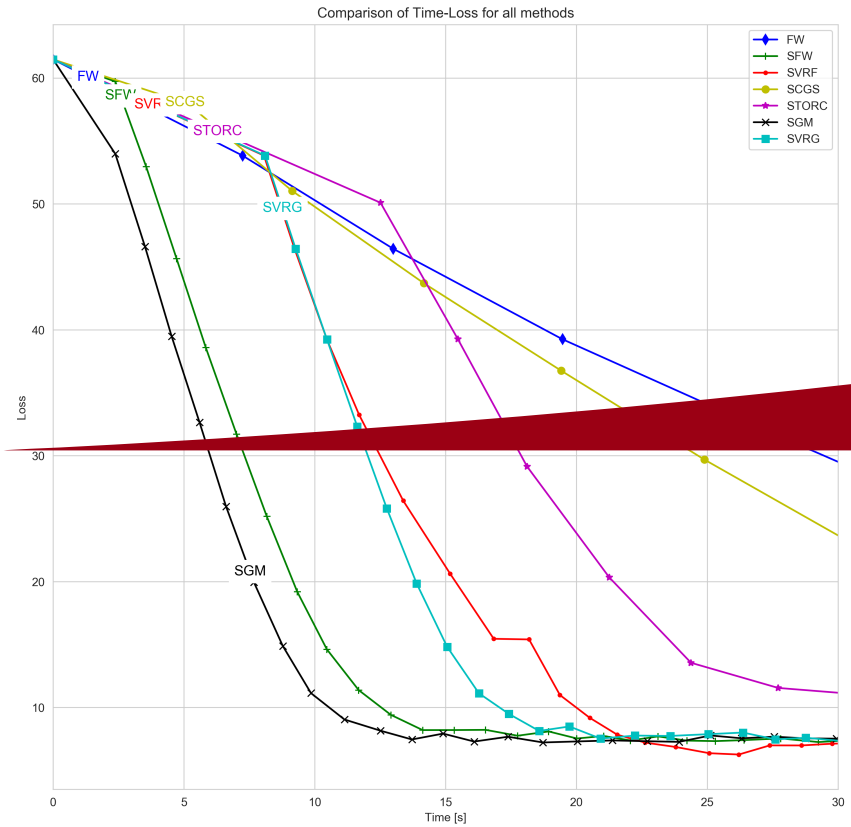
Global parameters:

- $\tau = 50$;
- running time $= 180\,\text{s}$.

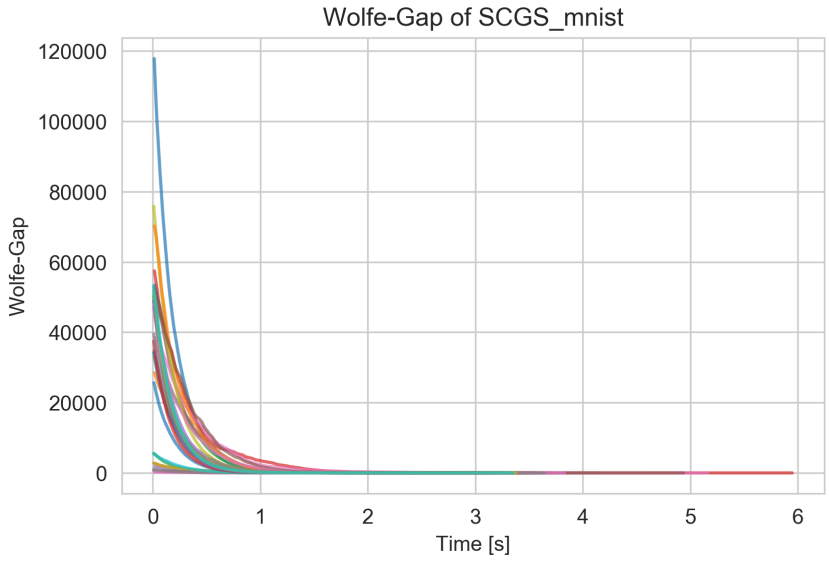
SGM & SVRG added for comparison!

Fashion MNIST

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

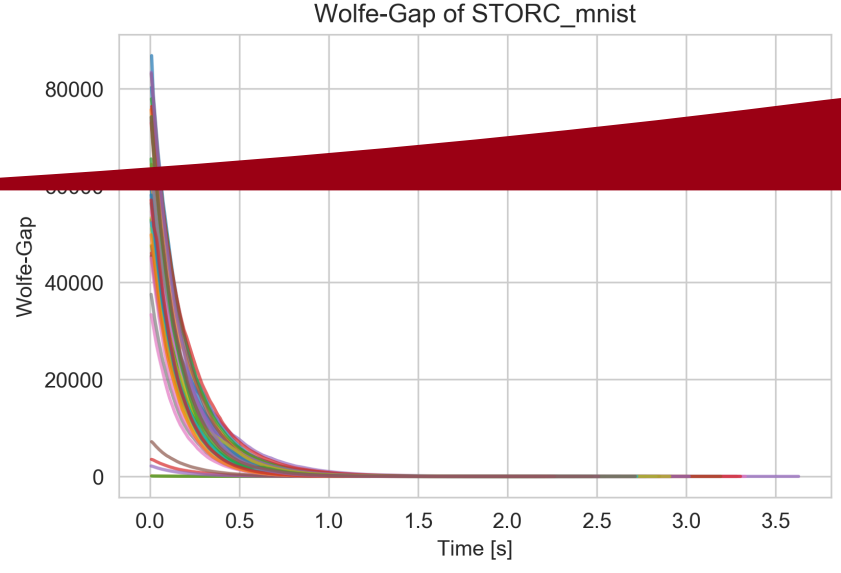


Fashion MNIST: Wolfe-Gap



of SCGS$_m$*nist.png*

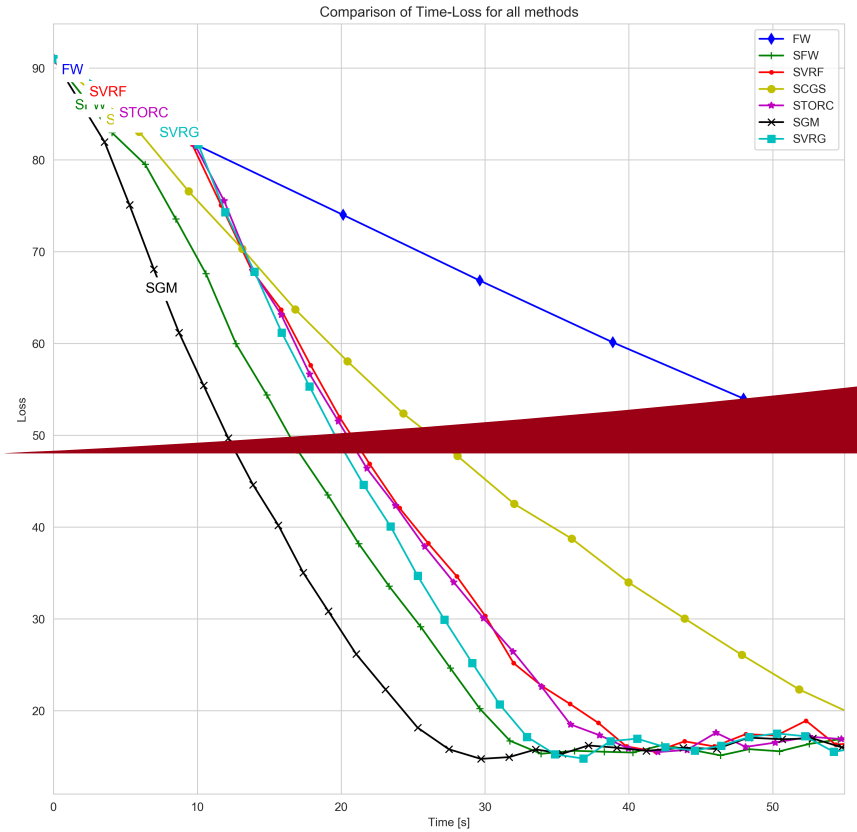
of



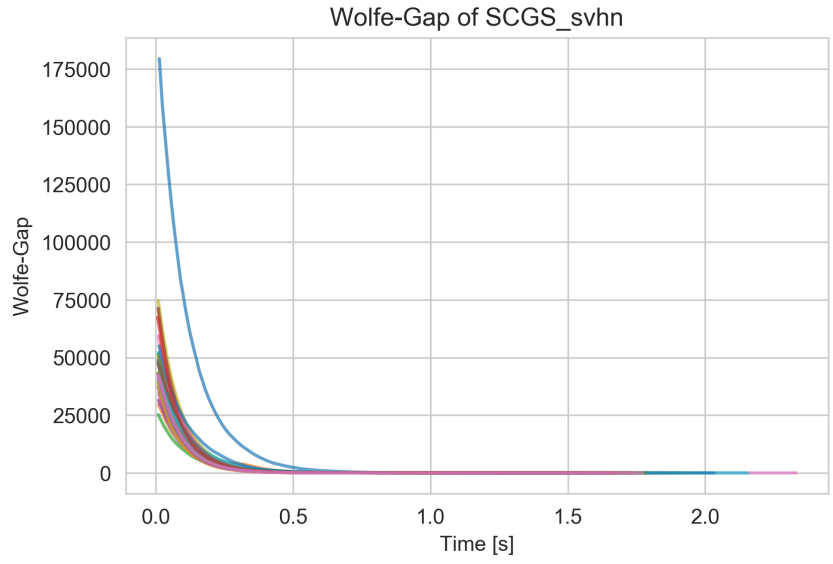
STORC$_m$*nist.png*

SVHN

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

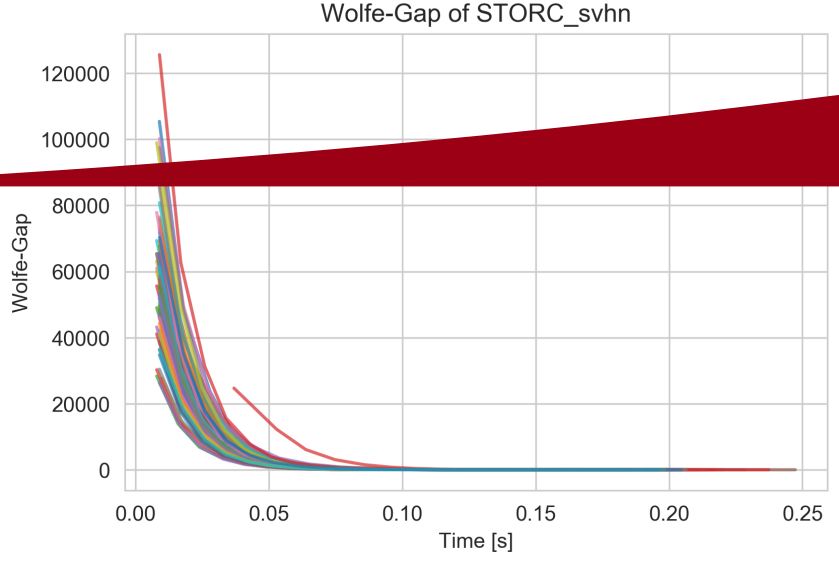


SVHN: Wolfe-Gap



of SCGS$_s$*vhn.png*

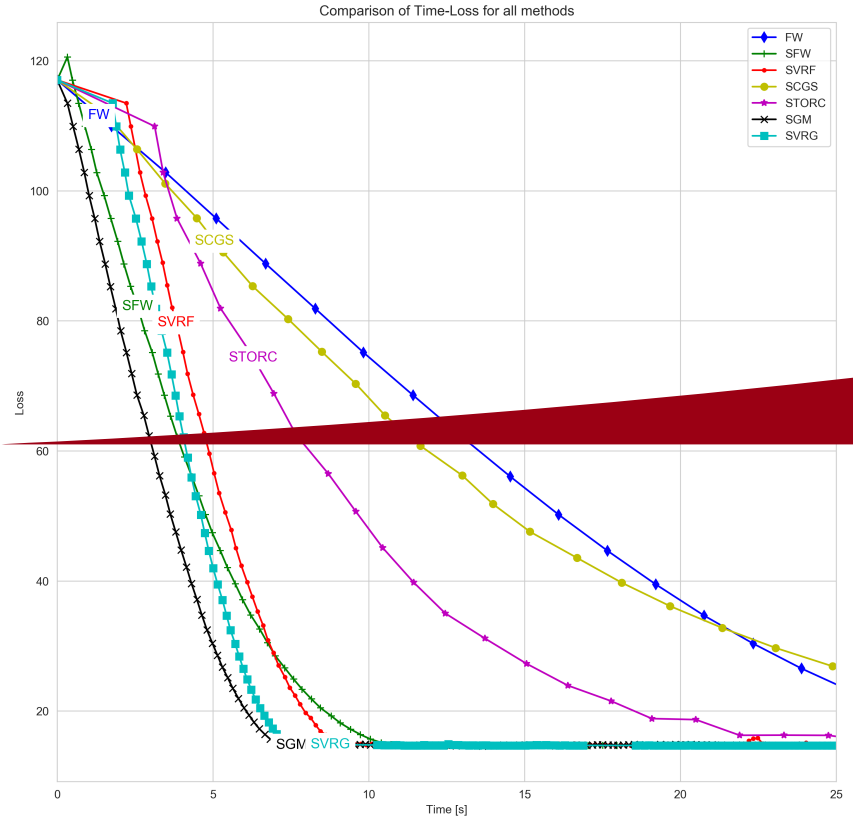
of



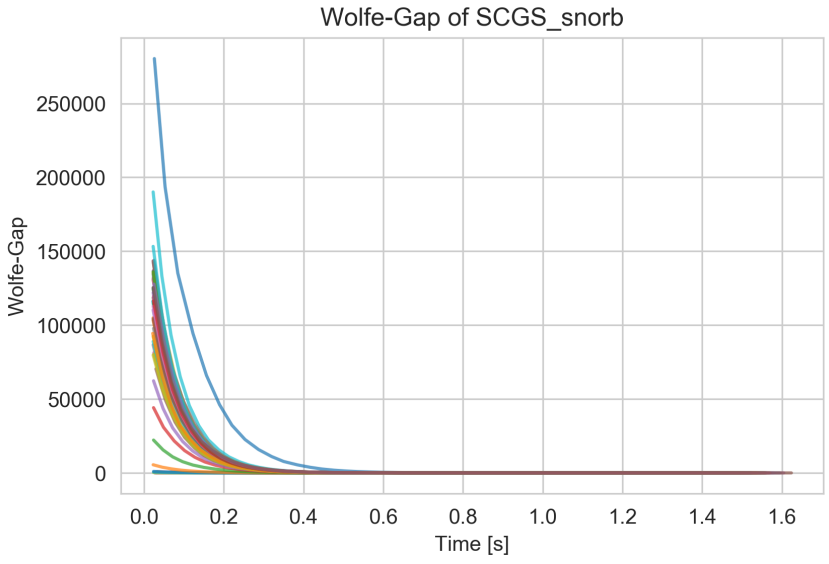
STORC$_s$*vhn.png*

small NORB

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

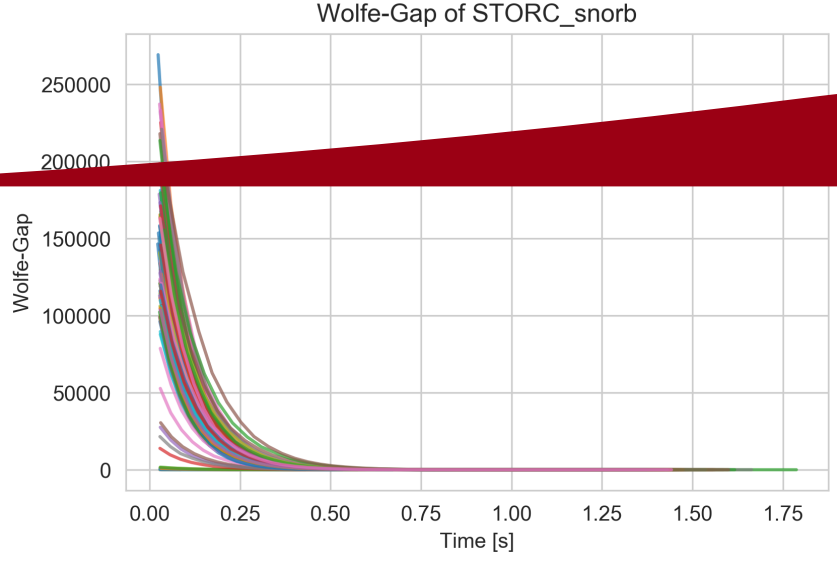


small NORB: Wolfe-Gap



of SCGS$_s$*norb.png*

of



STORC$_s$*norb.png*

Conclusions

SGM & SFW + SVRG & SVRF, but even the projected vs projection-free methods performs really close to each other.

Dimensions of the datasets, as well as the ones of the weights, play a central role in our performances!

In this configuration CPU time to compute the projection (entire SVD) $\simeq$ time required to solve the linear optimization problem of Franke-Wolfe's methods (only top singular vectors com

To ensure a significant difference between the two methods: $(\#features) \cdot (\#classes)$ at least one order of magnitude bigger.

Conclusions

SGM & SFW + SVRG & SVRF, but even the projected vs projection-free methods performs really close to each other.

Dimensions of the datasets, as well as the ones of the weights, play a central role in our performances!

In this configuration CPU time to compute the projection (entire SVD) $\simeq$ time required to solve the linear optimization problem of Franke-Wolfe's methods (only top singular vectors com

To ensure a significant difference between the two methods: $(\#features) \cdot (\#classes)$ at least one order of magnitude bigger.

Conclusions

SGM & SFW + SVRG & SVRF, but even the projected vs projection-free methods performs really close to each other.

Dimensions of the datasets, as well as the ones of the weights, play a central role in our performances!

In this configuration CPU time to compute the projection (entire SVD) $\simeq$ time required to solve the linear optimization problem of Franke-Wolfe's methods (only top singular vectors com

To ensure a significant difference between the two methods: $(\#features) \cdot (\#classes)$ at least one order of magnitude bigger.

Conclusions

SGM & SFW + SVRG & SVRF, but even the projected vs projection-free methods performs really close to each other.

Dimensions of the datasets, as well as the ones of the weights, play a central role in our performances!

In this configuration CPU time to compute the projection (entire SVD) $\simeq$ time required to solve the linear optimization problem of Franke-Wolfe's methods (only top singular vectors com

To ensure a significant difference between the two methods: $(\#features) \cdot (\#classes)$ at least one order of magnitude bigger.