Stochastic Frank-Wolfe Algorithms: an Experimental Evaluation

Gioele Ceccon 2079425 Gaia Nardella 2091413 Pietro Renna 2089068 Valerio Rocca 2094861

Abstract

This paper presents an implementation and evaluation of Stochastic Frank-Wolfe algorithms for constrained optimization on the ℓ_1-ball of both Convex and Nonconvex objectives. The focus is to understand how the algorithms perform on different tasks and whether they perform better than the (Stochastic) Gradient Descent method.

Introduction

Consider a constrained optimization problem defined as:

$$\min_{x \in \Omega} f(x) \tag{1}$$

with:

- $f : \mathbb{R}^d \to \mathbb{R}$ continuously differentiable function;
- $\Omega \subseteq \mathbb{R}^d$ closed convex set.

If the problem is dealt with the gradient method, each iterate is defined as:

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t) \tag{2}$$

where $t = 1, ..., T$ represent the t-th iteration and $\alpha_t \in (0, 1]$ is the stepsize.
This approach may result in $x_{t+1} \notin \Omega$. The problem can be addressed by employing the Projected Gradient Method, which involves projecting the given point onto the set Ω, and selecting the point in Ω closest to $x_t - \alpha_t \nabla f(x_t)$. However, Projected Gradient Methods may be very expensive or even intractable, especially for huge-scale problems and Data Science real-world applications.

Frank-Wolfe approach

The Frank-Wolfe (FW) algorithm, also known as Conditional Gradient Algorithm, is a projection-free first order algorithm for constrained optimization that overcomes the previously mentioned cost issues. It avoids the projection by leveraging the *linear minimization oracle*:

$$lmo_\Omega(v) = \arg\min_{x \in \Omega} \langle x, v \rangle \tag{3}$$

which is significantly cheaper compared to the projection operation (Combettes and Pokutta 2021).

The work focuses on the ℓ_1-ball constraint with radius r, defining the lmo as:

$$lmo_\Omega(v, r) = r \cdot \text{sign}(-v) \cdot e_i \tag{4}$$

e_i standard basis vector
$i = \arg\max_i |v|$

In terms of Frank-Wolfe optimization methods, v is referred to the Gradient estimation $\nabla f(\mathbf{x})$.
The feasible set Ω is then defined as:

$$\Omega = \{x \in \mathbb{R}^d : \|x\|_1 \leq r\} \tag{5}$$

Finite-sum minimization

The applications of this paper are based on a finite-sum minimization problem, that is formally defined as:

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

where $f_1, ..., f_n : \mathbb{R}^d \to \mathbb{R}$, f are continuously differentiable functions.
A specific function f is determined based on the task, so it can be Convex or Nonconvex. In Machine Learning context d refers to the number of features while n refers to the number of examples in the considered set.

Algorithms

This section introduces the algorithms used in the analysis and defines how their optimization process works, including the specific constraint that must be satisfied by each algorithm (excluding the SGD, which is unconstrained). The idea is to compare SFW, SPIDER-FW, AdaSVRF and the Stochastic Gradient Descent in order to assess the performances between different Frank-Wolfe approaches and the more classic SGD approach.
The weights values have been initialized respecting the ℓ_1-ball constraint for the given radius, whose value is assigned depending on the task.

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is a variant of Gradient Descent that uses a minibatch (small subset) of the original training

set to perform the optimization process.
Stochastic gradient method generates a new iterate as follows:

$$x_{t+1} = x_t - \alpha_t \nabla F(x_t, \xi_t), \tag{7}$$

where ξ_t is a sample realization of ξ and α_t a chosen stepsize.
Stochastic gradient is unbiased since:

$$\mathbb{E}[\nabla F(x_t, \xi_t)] = \nabla f(x). \tag{8}$$

Algorithm 1: Stochastic Gradient Descent (SGD)

1: Choose a point $x_1 \in \Omega$
2: **for** $t = 1, ...$ **do**
3: **if** x_t satisfies some specific condition, then STOP
4: Choose ξ_t a sample realization of ξ
5: Set $x_{t+1} = x_t - \alpha_t \nabla F(x_t, \xi_t)$, with $\alpha_t > 0$ a suitably chosen stepsize
6: **end for**
7: **Output** Last iterate x_{T+1}

Moreover, the algorithm requires a decreasing stepsize α_t to ensure convergence, since the optimum solution is obtained as:

$$x^* = x^* - \alpha_t \nabla F(x^*, \xi) \tag{9}$$

and since $\nabla F(x^*, \xi)$ is random, it is not possible to guarantee that

$$\nabla F(x^*, \xi) = 0 \tag{10}$$

for all $\xi \in \Omega$.
One of the key strengths of the Stochastic Gradient method is its computational efficiency per iteration. This makes it particularly advantageous for large-scale optimization problems where computational resources are limited or when dealing with massive datasets. By performing updates based on a random subset of the data at each iteration, Stochastic Gradient significantly reduces the computational burden compared to traditional Gradient-Based methods, allowing faster convergence and more convenient model training.
Stochastic Gradient method algorithm guarantees a sublinear convergence rate of $\mathcal{O}(\frac{1}{t})$, where t is the number of iterations, and a cost $\mathcal{O}(1)$ per iteration.

Stochastic Frank-Wolfe (SFW)

Stochastic Frank-Wolfe algorithm is an extension of the Frank-Wolfe algorithm that incorporates stochasticity: instead of computing the exact gradient of the objective function, it uses stochastic estimates of the gradients at each iteration. This makes it more suitable for large-scale or computationally intensive optimization problems where computing the full gradient may be expensive.

Algorithm 2: Stochastic Frank-Wolfe (SFW)

Input: $x_0 \in \Omega$, number of iterations T, $\{\gamma_i\}_{i=0}^{T-1}$ where $\gamma_i \in [0, 1]$ for all $i \in \{0, ..., T-1\}$, minibatch size $\{b_i\}_{i=0}^{T-1}$
1: **for** $t = 0$ to $T - 1$ **do**
2: Uniformly randomly pick i.i.d samples $\{z_{1t}, ..., z_{bt}\}$ according to the distribution $\boldsymbol{P}$
3: Compute $v_t = \arg\max_{v \in \Omega} \langle v, -\frac{1}{b_t} \sum_{i=1}^{b_t} \nabla f(x_t, z_i) \rangle$
4: Compute update direction $d_t = v_t - x_t$
5: $x_{t+1} = x_t + \gamma_t d_t$
6: **end for**
7: **Output** Iterate x_α chosen uniformly random from $\{x_t\}_{t=0}^{T-1}$

Samples z_i are chosen independently according to the distribution P. Thus, $\mathbb{E}[\nabla f(x_i, z_i)] = \nabla F(x)$ (i.e., the estimate of the gradient is unbiased). It is also worth noting that the implemented version presented in Algorithm 2 (Sashank J. Reddi 2016) the output is randomly selected from all the iterates of the algorithm.

AdaSVRF

Algorithm 3: AdaSVRF

Input: Start point $x_0 \in \Omega$, snapshot times $s_k < s_{k+1}$ with $s_0 = 0$, batch sizes $b_t \in \mathbb{N} \setminus \{0\}$, bounds $0 < \lambda_t^- \leq \lambda_{t+1}^- \leq \lambda_{t+1}^+ < \lambda_t^+$, number of inner iterations $K \in \mathbb{N} \setminus \{0\}$, learning rates $\eta_t > 0$, step-size bounds $\gamma_t \in [0, 1]$.

1: **for** $t = 0$ to $T - 1$ **do**
2: **if** $t \in \{s_k | k \in N\}$ **then**
3: $\widetilde{x}_t \leftarrow x_t$
4: $\widetilde{\nabla} f(x_t) \leftarrow \nabla f(\widetilde{x}_t)$
5: **else**
6: $\widetilde{x}_t \leftarrow x_{t-1}$
7: $i_1, \ldots, i_{bt} \overset{\text{i.i.d.}}{\sim} \mathcal{U}([1, m])$
8: $\widetilde{\nabla} f(x_t) \leftarrow \nabla f(\widetilde{x}_t) + \frac{1}{b_t} \sum_{i=1}^{b_t} (\nabla f_i(x_t) - \nabla f_i(\widetilde{x}_t))$
9: **end if**
10: Update H_t and clip its entries to $[\lambda_t^-, \lambda_t^+]$
11: $y_0^t \leftarrow x_t$
12: **for** $k = 0$ to $K - 1$ **do**
13: $\nabla Q_t(y_k^t) \leftarrow \widetilde{\nabla} f(x_t) + \frac{1}{\eta_t} H_t(y_k^t - x_t)$
14: $v_k^t \leftarrow \text{argmin}_{v \in \Omega} \langle \nabla Q_t(y_k^t), v \rangle$
15: $\gamma_k^t \leftarrow \min\left(\eta_t \frac{\langle \nabla Q_t(y_k^t), (y_k^t - v_k^t) \rangle}{\|y_k^t - v_k^t\|_{H_t}^2}, \gamma_t\right)$
16: $y_{k+1} \leftarrow y_k^t + \gamma_k^t (v_k^t - y_k^t)$
17: **end for**
18: $x_{t+1} \leftarrow y_K^t$
19: **end for**
20: **Output** Last iterate x_T

The Stochastic Variance Reduced Frank-Wolfe (SVRF)

method is a variant of the Stochastic Frank-Wolfe algorithm that combines a stochastic gradient estimator with a variance reduction technique to achieve faster convergence rates in high-dimensional optimization problems. The Adaptive Gradient (AdaGrad) algorithm, on the other hand, is a gradient-based optimization method that adapts the learning rate of the gradient descent algorithm to the geometry of the optimization problem. This way, infrequent features receive large step-sizes whenever they appear, allowing the algorithm to notice these rare but potentially very informative features. AdaSVRF (Combettes, Spiegel, and Pokutta 2020) combines the SVRF method with AdaGrad to overcome a limitation of the latter. Specifically, in the context of minimizing an objective over a constrained set, AdaGrad can be computationally expensive at each iteration. By incorporating Frank-Wolfe into the optimization process, the proposed algorithm aims to address this issue while preserving the desirable properties of the descent directions. In particular, what AdaSVRF does is applying K iterations of SVRF to

$$\min_{\mathbf{x}\in\Omega} Q_t(\mathbf{x}) = f(\mathbf{x}_t) + \langle \widetilde{\nabla} f(\mathbf{x}_t), \mathbf{x} - \mathbf{x}_t \rangle + \frac{1}{2\eta_t}\|x - x_t\|^2_{H_t} \tag{11}$$

where $\eta_t > 0$ is a time-varying learning rate, H_t is any diagonal matrix with positive entries and $\|x\|_{H_t} = \sqrt{\langle x, xH_t \rangle}$. It is the same sub-problem that AdaGrad needs to solve at each iteration. In addition to that, entries of H_t are clipped to ensure that $\sup_{t\in\mathbb{N}} \lambda_{\max}(H_t) < +\infty$, as done in (Luo et al. 2019). At every iteration $t = s_k, k \in N$, the algorithm computes the exact gradient of the iterate, saves it into memory, then builds the gradient estimator $\widetilde{\nabla} f(x_t)$ in the following iterations $t \in [s_k + 1, s_{k+1} - 1]$.
Moreover, at each iteration it updates H_t using AdaGrad's strategy:

$$H_t = \delta + \sqrt{\sum_{s=0}^{t} (\widetilde{\nabla} f(x_s))^2} \tag{12}$$

where $\delta \leftarrow 10^{-8}$ is a constant useful to prevent from dividing by zero. Although the previously discussed clipping has been implemented, using a non-decreasing function as a lower bound and a non-increasing function as an upper bound, it appears that this approach has a negative impact on the algorithm's performance in practice. Furthermore, the approach was not implemented in (Combettes, Spiegel, and Pokutta 2020) and was only considered for the convergence analysis.

The proposed implementation then uses the following functions to initialize at each iteration the parameters:

- $s_k \leftarrow 2^{k+k_0} - 2^{k_0}$;
- $K = 5$, since it is a good default value in general (Combettes, Spiegel, and Pokutta 2020);
- $b_t \leftarrow 8(2^{k_0+1} + 1)(K + 3)$;
- $\gamma_t \leftarrow \frac{2}{t+2}$;
- $\eta_t \leftarrow \frac{\lambda_t^-}{L}$.

where $k_0 \in N^+$ is a constant ($k_0 = 2$ in the proposed implementation), useful in practice to avoid computing exact gradients too many times in the early iterations. and L is the Lipschitz constant (fixed at 0.001 in the proposed implementation).

SPIDER-FW

It is a Stochastic Frank-Wolfe method based on the *Stochastic Path Integrated Differential Estimator* (SPIDER) technique, which can be seen as a stochastic variant of the Normalized Gradient Descent to avoid excessive access of the stochastic oracle and to reduce the complexity (Fang et al. 2018).
The implementation in Algorithm 4 has been developed by (Yurtsever, Sra, and Cevher 2019)

Algorithm 4: SPIDER Frank-Wolfe

Input: $x_1 \in \Omega$

for $t = 1, 2, ..., T$ **do**
  Set $x_1 = \bar{x}_t$
  Draw Q_t samples $\boldsymbol{Q_t}$
  Compute $v_1 = \nabla f_{Q_t}(x_1)$
  Compute $w_1 \in lmo_\Omega(v_1)$
  Update $x_2 = x_1 + \eta_{t,1}(w_1 - x_1)$
  for $k = 2, 3, ..., K_t$ **do**
    Draw $S_{t,k}$ samples $\boldsymbol{S_{t,k}}$
    Compute $v_k = \nabla f_{S_{t,k}}(x_k) - \nabla f_{S_{t,k}}(x_{k-1}) + v_{k-1}$
    Compute $w_k \in lmo_\Omega(v_k)$
    Update $x_{k+1} = x_k + \eta_{t,k}(w_k - x_k)$
  end for
  Set $\bar{x}_{t+1} = x_{K_{t+1}}$
end for
Output Last iterate $\bar{x}_{T+1}$

Depending on the task, there are two distinct settings:

- **Convex finite-sum**: used for Linear Regression optimization.
  - $K_t = 2^{t-1}$ for $t = 1, ..., T$;
  - Sampling parameters: $\boldsymbol{S_{t,k}} = K_t$, $\boldsymbol{Q_t} = [n]$
  - Learning Rate (Stepsize): $\eta_{t,k} = \frac{2}{s_{t,k}+1}$, where $s_{t,k} = K_t + k - 1$
- **Nonconvex finite-sum**: used for Regularized Logistic Regression (Lasso, L1 norm penalty), Support Vector Classifier (even if it is a convex problem, due to major performance issues with the Convex setting) and Neural Networks.
  - $K_t = K = \lceil \sqrt{n} \rceil$;
  - Sampling parameters: $\boldsymbol{S_{t,k}} = \boldsymbol{S} = K = \lceil \sqrt{n} \rceil$, $\boldsymbol{Q_t} = [n]$;
  - Learning Rate (Stepsize): $\eta_{t,k} = \eta = \frac{1}{\sqrt{s_{T,K}+1}}$, where $s_{T,K} = TK$.

Computational Experiments

The aforementioned algorithms have been tested using real-world data on the following regression and classification models: Linear Regression, Support Vector Classification, L1-Regularized Logistic Regression, Shallow Feed-Forward Neural Network and Shallow Convolutional Neural Network.
SPIDER-FW and AdaSVRF uses the learning rates described in the respective sections. SGD and SFW's learning rates were set respectively as:

- $\gamma_t = \frac{\gamma_{t-1}}{T}$ with $\gamma_1 = 0.05$ for $t = 1, ..., T$;
- $\gamma_t = \frac{2}{t+2}$ for $t = 0, ..., T-1$.

Convex Objectives

This section describes the tasks involving Convex Objectives. The algorithms have been compared on two convex optimization problems using the duality gap measured with respect to CPU Time and Epochs. The duality gap is defined as:

$$\text{dualitygap}(\bar{x}_t) = \langle \nabla f(\bar{x}_t), x_t - lmo(\nabla f(\bar{x}_t)) \rangle \quad (13)$$

which serves as a measure of convergence (Combettes, Spiegel, and Pokutta 2020).

Linear Regression

This experiment has been conducted on the Boston dataset. The goal is to predict the median value $y_1, ..., y_n$ of owner-occupied homes in the area of Boston, Massachusetts. A sparsity-inducing constraints via the L1-norm is also included. The objective function to minimize for the task, the Mean-Squared Error (MSE), is shown in Equation 14.

$$\begin{aligned} & f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\mathbf{w})^2 \\ & s.t. \|x\|_1 \leq 100 \end{aligned} \quad (14)$$

The dataset contains $n = 404$ samples and $d = 13$ features. Equation 15 and Equation 16 define, respectively, the full gradient and the mini-batch gradient used to run the algorithms for this specific task:

$$\nabla f(\mathbf{w}) = -\frac{2}{n}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) \quad (15)$$

$$\nabla f_i(\mathbf{w}) = -\frac{2}{|idx|}\sum_{i \in idx} \mathbf{x}_i^T(y_i - \mathbf{x}_i^T\mathbf{w}) \quad (16)$$

The algorithms ran for 100 epochs and the radius r for the ℓ_1-ball was fixed to 100.
Looking at Table 1, it is possible to see the MSE values obtained by the algorithms. It is easy to notice that AdaSVRF obtain the worst result, while the others score closer values.

Moreover, looking at Table 2, it can be noticed how all the algorithms respect the constraint (even the SGD).
Figure 1 showcases the behavior of the duality gap for the Frank-Wolfe algorithms. Some considerations are:

Table 1: MSE for Linear Regression

Algorithm	MSE
SGD	583.17
SFW	607.68
AdaSVRF	1213.88
SPIDER-FW	626.13

Table 2: L1-Norm of parameters for Linear Regression

Algorithm	L1-Norm
SGD	23.96
SFW	27.02
AdaSVRF	54.94
SPIDER-FW	40.53

- AdaSVRF takes the longest time to execute, with SPIDER-FW and SFW following at a distance. This outcome is not surprising, given the computations it needs to carry out at each iteration;
- All three algorithms appear to converge, but it is worth noticing that the SFW algorithm achieves significantly higher duality gap scores in the initial iterations compared to the other two algorithms.

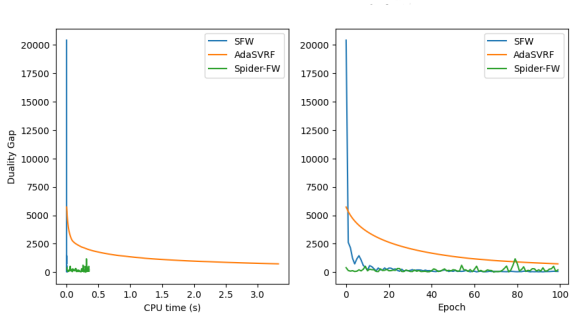

Figure 1: Linear Regression on the Boston dataset

Support Vector Classification

The method performs binary classification on the Breast Cancer dataset. The objective function to minimize for the task, the Smoothed Hinge Loss (SHL) Function, is shown in Equation 17.

$$\begin{aligned} & f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\max\{0, 1 - y_i\langle x_i, w\rangle\}^2) \\ & s.t. \|x\|_1 \leq 1 \end{aligned} \quad (17)$$

The number of samples is $n = 455$ and the number of features is $d = 30$. The full gradient and the mini-batch gradient used to run the algorithms for this specific task are shown, respectively, in Equation 18 and Equation 19:

$$\nabla f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\left[-2y_i\mathbf{x}_i(1-y_i\mathbf{x}_i^T\mathbf{w})\right]\mathbb{I}(1-y_i\mathbf{x}_i^T\mathbf{w}>0) \tag{18}$$

$$\nabla f_i(\mathbf{w}) = \frac{1}{|idx|}\sum_{i\in idx}\left[-2y_i\mathbf{x}_i(1-y_i\mathbf{x}_i^T\mathbf{w})\right]\mathbb{I}(1-y_i\mathbf{x}_i^T\mathbf{w}>0) \tag{19}$$

To perform this task, the algorithms ran for 100 epochs and the radius r for the ℓ_1-ball was set to 1.
Table 3 displays the accuracy scores obtained by the algorithms, revealing that they all achieve comparable results, although SFW performs slightly worse.

Table 3: Accuracy for Support Vector Classification

Algorithm	**Accuracy**
SGD	0.98
SFW	0.94
AdaSVRF	0.96
SPIDER-FW	0.97

Table 4 showcases the L1-Norm of the solutions obtained. As expected, the Frank-Wolfe algorithms respect the constraint.

Table 4: L1-Norm of parameters for Support Vector Classification

Algorithm	**L1-Norm**
SGD	3.83
SFW	0.99
AdaSVRF	0.96
SPIDER-FW	1.0

Figure 2 showcases the behavior of the duality gap for the Frank-Wolfe algorithms. Some considerations are:

- As expected, AdaSVRF takes the longest time to carry out the execution;
- It appears that all three algorithms converge, and their behavior is similar to what was observed in the Linear Regression experiment, even though there are less smooth trajectories on this task.

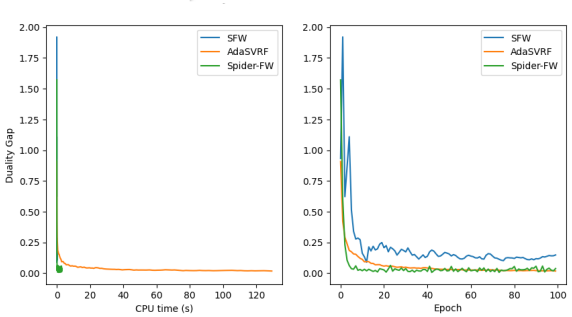

Figure 2: Support Vector Classification on the Breast Cancer dataset

Nonconvex Objectives

This section focuses on classification tasks with Nonconvex Objectives. Specifically, the algorithms' performances are compared on the following architectures:

- L1-Regularized Logistic Regression (Lasso), using the Duality Gap as the evaluation metric;
- Feed-Forward NN and Convolutional NN, using Loss on the training set and Test Accuracy as the evaluation metrics.

The metrics are plotted against CPU Time and Epochs.

Regularized Logistic Regression (Lasso, L1-Norm penalty)

The Lasso Logistic Regression is a regularized version of the Logistic Regression, which adds the L1-Norm penalty term on the weights $\mathbf{w}$. It shrinks the coefficients exactly to zero, performing feature selection. The objective function to minimize is then the Regularized Cross Entropy, shown in Equation 20.

$$\begin{aligned} &f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\log\left(1+\exp(-y_i\mathbf{x}_i^T\mathbf{w})\right) + \lambda\|\mathbf{w}\| \\ &s.t. \|x\|_1 \leq 100 \\ &\lambda > 0 \text{ regularization hyperparameter.} \end{aligned} \tag{20}$$

This experiment is performed on the Breast Cancer dataset, with $n = 455$ samples and $d = 30$ features. The full gradient and the mini-batch gradient used to run the algorithms for this specific task are shown, respectively, in Equation 21 and Equation 22:

$$\nabla f_i(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i^T\left(y_i\frac{\exp(-y_i(\mathbf{x}_i^T\mathbf{w}))}{1+\exp(-y_i(\mathbf{x}_i^T\mathbf{w}))}\right) + \text{sign}(\mathbf{w})\lambda \tag{21}$$

$$\nabla f_i(\mathbf{w}) = -\frac{1}{|idx|}\sum_{i\in idx}\mathbf{x}_i^T\left(y_i\frac{\exp(-y_i(\mathbf{x}_i^T\mathbf{w}))}{1+\exp(-y_i(\mathbf{x}_i^T\mathbf{w}))}\right) + \text{sign}(\mathbf{w})\lambda \tag{22}$$

To perform this task, the algorithms ran for 100 epochs. The radius for the ℓ_1-ball was set to 100, to represent the constraint imposed on the problem. The accuracy scores obtained by the algorithms are presented in Table 3. It is evident that SFW outperforms the other algorithms, while SGD and AdaSVRF perform poorly.

Table 5: Accuracy for Regularized Logistic Regression

Algorithm	**Accuracy**
SGD	0.08
SFW	0.92
AdaSVRF	0.08
SPIDER-FW	0.33

Table 6 showcases the L1-norm of the solutions obtained by the algorithms. It is evident that they all satisfy the constraint. Notably, SFW, which achieves the best accuracy result, produces a solution with a lower L1-norm compared to the other solutions.

Table 6: L1-Norm of parameters for Lasso Logistic Regression

Algorithm	**L1-Norm**
SGD	61.33
SFW	28.94
AdaSVRF	73.47
SPIDER-FW	70.91

Figure 3 displays the behavior of the duality gap for the Frank-Wolfe algorithms. Some considerations are:

- The only algorithm that seems to converge is SFW. This is coherent with the results in Table 5;
- While AdaSVRF exhibits a smooth trajectory, SPIDER-FW's one is characterized by continuous ups and downs. Interestingly enough, SPIDER-FW still obtains a better accuracy compared to SGD and AdaSVRF.

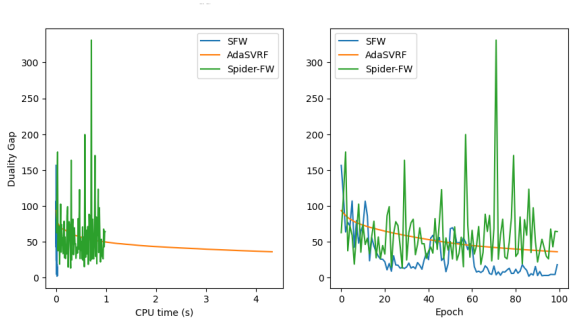

Figure 3: Regularized Logistic Regression on the Breast Cancer dataset

Shallow Feed-Forward Neural Network (Shallow FFNN)

A Shallow FFNN is a Feed-Forward Neural Network that has only one hidden layer. The architecture used for the experiments in this paper solves a binary classification problem on the Breast Cancer dataset.

Activation Functions
The hidden layer uses the ReLU activation function:

$$\text{ReLU(x)} = \max(0, x)$$

$$\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

The output layer uses the sigmoid function:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Parameters initialization
W=(W_1, W_2) set of weights to optimize
W_1 : Input to hidden weight matrix
b_1 : Bias
W_2 : Hidden to output weight matrix
b_2 : Bias

Forward Propagation
Used to compute predictions in both train (minibatch) and test phase.
For this reason, X here means a generic input matrix X

$$Z_1 = W_1 X^T + b_1$$
$$A_1 = \text{ReLU}(Z_1)$$
$$Z_2 = W_2 A_1 + b_2$$
Prediction: $A_2 = \sigma(Z_2)$

Objective Function: Cross Entropy

$$J(\mathbf{W}) = -\frac{1}{n}\sum_{i=1}^{n} y_{\text{train}}^{(i)} \log(A_2^{(i)}) + (1 - y_{\text{train}}^{(i)}) \log(1 - A_2^{(i)}) \quad (23)$$

Backward Propagation
Used to compute the gradients in train phase. Takes as input X_{train_i}, y_{train_i}.

$$dZ_2 = A_2 - y_{\text{train}}$$
$$dW_2 = \frac{1}{n} dZ_2 A_1^T$$
$$db_2 = \frac{1}{n}\sum_{i=1}^{n} dZ_{2,i}$$
$$dZ_1 = W_2^T dZ_2 \ \text{ReLU}'(Z_1)$$
$$dW_1 = \frac{1}{n} dZ_1 X_{\text{train}}$$
$$db_1 = \frac{1}{n}\sum_{i=1}^{n} dZ_{1,i}$$

Experiment description
The implemented architecture has 2 Hidden Neurons and it

is constrained using radius $r = 1$. To perform this task, the algorithms ran for 300 epochs.

Table 7: Test Accuracy of the NN for binary classification task

Algorithm	**Accuracy**
SGD	0.94
SFW	0.88
AdaSVRF	0.90
SPIDER-FW	0.91

- SGD is the fastest algorithm in terms of convergence (Figure 6) and it is also the one with the highest final accuracy (0.94). Perhaps, looking at Figure 7, it is evident that it got stuck until epoch 150, unlike other algorithms which were able to achieve high accuracy values even in the first epoch iterations;
- Figure 7 highlights that SPIDER-FW is the fastest in reaching high accuracy values right from the first epoch iterations. AdaSVRF takes longer with respect to SPIDER-FW and SFW, although it's worth noting that SFW reaches much lower values at certain epoch iterations. In general, Frank-Wolfe methods need less epochs to achieve high accuracy epochs with respect to the SGD.

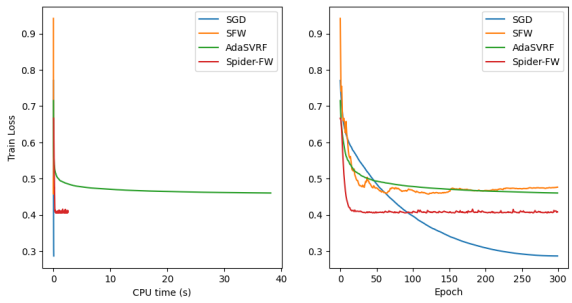

Figure 4: Train Loss of the NN on the Breast Cancer dataset

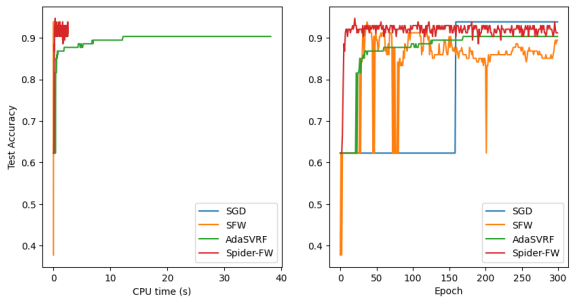

Figure 5: Test accuracy of the NN on the Breast Cancer dataset

Convolutional Neural Network

A Convolutional Neural Network (CNN) is a deep learning algorithm commonly used for image recognition, processing, and analysis.
The model has been applied to solve a constrained variant of the multiclass classification problem over the CIFAR-10 dataset via ℓ_1-ball with radius $r = 100$.
The dataset contains 60000 images of dimension 32x32, 6000 for each class: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Data has been split into training, validation, and test set having, respectively, 10000, 40000, and 10000 images. The low number of images in the training set has been chosen because of computational feasibility reasons.
Follows the specific architecture of the model used:

- a convolutional layer with 32 filters of dimension 3x3;
- a ReLU activation function;
- a max-pooling layer of dimension 3x3;
- a convolutional layer with 16 filters of dimension 2x2;
- a ReLU activation function;,
- a max-pooling layer of dimension 3x3;
- Objective function: Cross Entropy.

The experiment
Following the guidelines in paragraph 4.2 of (Combettes, Spiegel, and Pokutta 2020), it was decided not to include AdaSVRF in this experiment. Furthermore, due to AdaSVRF's lengthy execution time, it was not practical to assess it for this particular task given the available resources. SPIDER-FW has been trained with the batch setting described in its setting. Instead, SGD and SFW batch dimension has been set equal to $\sqrt{N}$. Moreover, it was decided to train the SFW considering all the batches at each iteration: this allows the algorithm to converge faster, even though it makes the assumption of independence not respected.
The algorithms were run for 100 epochs.

Table 8: Validation Accuracy of the CNN

Algorithm	**Accuracy**
SGD	0.17
SFW	0.10
SPIDER-FW	0.10

No algorithm is able to achieve a satisfactory accuracy. Further experiments with different rates have shown that, while SGD was able to dramatically improve its performance, SFW and SPIDER-FW's accuracy values were still around 0.1: this is the reason why analysis on test set are not presented.

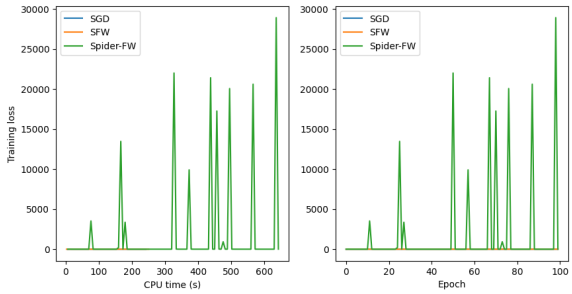

Figure 6: Train Loss of the CNN

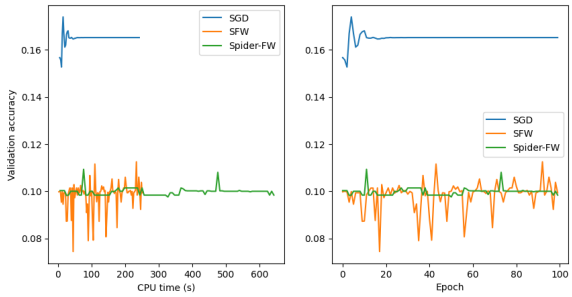

Figure 7: Validation accuracy of the CNN

Conclusions

The proposed experiments have revealed the behavior of each algorithm on various architectures and tasks within a constrained optimization framework. Among the Frank-Wolfe algorithms, the SFW stands out as the best one, capable of achieving satisfactory performances despite being constrained when compared to the unconstrained SGD algorithm. Perhaps, it is worth mentioning that each algorithm may have different behaviours with different datasets or with slightly different architectures (e.g., more hidden neurons in the Shallow Neural Network). Interestingly, this is not true for the Convolutional Neural Network, as the employment of different hyperparameters did not lead to any significant improvement. Given the architecture complexity, further applications with more refined constraints could be performed. The subpar performance of AdaSVRF and SPIDER-FW in the proposed experiments may be caused by their design for resilience in large-scale problems, where instead they should excel.

Further leveraging the capabilities of Python libraries (e.g.,PyTorch) for the algorithms code may lead to overall performance improvements.

References

Combettes, C. W.; and Pokutta, S. 2021. Complexity of linear minimization and projection on some sets. *Operations Research Letters*, 49(4): 565–571.

Combettes, C. W.; Spiegel, C.; and Pokutta, S. 2020. Projection-free adaptive gradients for large-scale optimization. *arXiv preprint arXiv:2009.14114*.

Fang, C.; Li, C. J.; Lin, Z.; and Zhang, T. 2018. Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator. *Advances in neural information processing systems*, 31.

Luo, L.; Xiong, Y.; Liu, Y.; and Sun, X. 2019. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*.

Sashank J. Reddi, B. P. A. S., Suvrit Sra. 2016. Stochastic frank-wolfe methods for nonconvex optimization. *54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*.

Yurtsever, A.; Sra, S.; and Cevher, V. 2019. Conditional gradient methods via stochastic path-integrated differential estimator. In *International Conference on Machine Learning*, 7282–7291. PMLR.