# Optimization for Data Science: A comparison between SFW and AdaSVRF

Cogo Giulio (2097433) - Nazli Hanifi (2072020)

***Abstract*—This paper presents an implementation and evaluation of two different Stochastic Frank-Wolfe algorithms for constrained optimization on the $\ell^1$-ball of both Convex and Non-convex objectives. The two considered algorithms are the Stochastic Frank-Wolfe (SFW) and AdaSVRF, the objective is to comprehend the algorithms' performance on various tasks and determine which one is more efficient.**

## I. 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 $t = 1, \ldots, T$ represents the $t$-th iteration and $\alpha_t \in (0, 1]$ is the stepsize.
Using this method could lead to $x_{t+1} \notin \Omega$. The Projected Gradient Method, which entails projecting the supplied point onto the set $\Omega$, and choosing the point in $\Omega$ closest to $x_t - \alpha_t \nabla f(x_t)$ can be used to solve the problem. However, Projected Gradient Methods could be prohibitively expensive or even unmanageable, particularly for large-scale issues and practical applications of data science.

## II. Frank-Wolfe approach

The conditional gradient method, or Frank-Wolfe (FW) algorithm, is a projection-free first order approach for constrained optimization that solves the cost problems previously described. It uses the linear minimization oracle to avoid the projection:

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

This, when compared to the projection process, is far less expensive (Combettes and Pokutta 2021). The $\ell_1$-ball constraint with radius r is the main focus of the work, and the lmo is defined as:

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

where $e_i$ is the standard basis vector with $i = \arg\max_i |v|$.In terms of Frank-Wolfe optimization methods, $v$ is referred to as the gradient estimation $\nabla f(x)$.
The feasible set $\Omega$ is then defined as:

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

## III. Finite-sum minimization

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

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

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

## IV. Algorithms

This segment provides an overview of the algorithms employed in the analysis and elucidates the mechanics of their optimization processes. It outlines the distinctive constraints that each algorithm must meet. The objective is to evaluate the performance disparities between SFW and AdaSVRF. The weight values are initialized while adhering to the $\ell^1$-ball constraint, with the assigned radius value contingent upon the specific task.

## V. Stochastic Frank-Wolfe (SFW)

A variant of the Frank-Wolfe method that takes stochasticity into account is the stochastic Frank-Wolfe algorithm, which employs stochastic estimates of the gradients at each iteration rather than computing the exact gradient of the objective function. This makes it more appropriate for large-scale or computationally demanding optimization problems, where it could be costly to compute the entire gradient.

***Algorithm 1: 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 $i \in \{0, \ldots, T-1\}$, minibatch size $\{b_i\}_{i=0}^{T-1}$
- for $t = 0$ to $T - 1$ do
  - Uniformly randomly pick i.i.d samples $\{z_{1t}, \ldots, z_{bt}\}$ according to the distribution $P$
  - Compute $v_t = \arg\max_{v\in\Omega} \langle v, -\frac{1}{b_t} \sum_{i=1}^{b_t} \nabla f(x_t, z_i)\rangle$
  - Compute update direction $d_t = v_t - x_t$
  - $x_{t+1} = x_t + \gamma_t d_t$
- end for
- *Output:* Iterate $x_\alpha$ chosen uniformly random from $\{x_t\}_{t=0}^{T-1}$

$z_i$ are randomly selected samples based on the distribution $P$. As a result, the estimate of the gradient is unbiased: $\mathbb{E}[\nabla f(x_i, z_i)] = \nabla F(x)$. It's also important to note that in

the implemented version of Algorithm (Sashank J. Reddi, 2016), the output is chosen at random from among all of the algorithm's iterations.

## VI. AdaSVRF

***Algorithm 2: AdaSVRF***

- *Input:* Start point $x_0 \in \Omega$, snapshot times $s_k < s_{k+1}$ with $s_0 = 0$, batch sizes $b_t \in \mathbb{N}\backslash\{0\}$, 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 $\gamma_t \in [0, 1]$.
- for $t = 0$ to $T - 1$ do
  - if $t \in \{s_k | k \in \mathbb{N}\}$ then
    - $\tilde{x}_t \leftarrow x_t$
    - $\nabla ef(x_t) \leftarrow \nabla f(x_{et})$
  - else
    - $\tilde{x}_t \leftarrow x_{t-1}$
    - $i_1, \ldots, i_{b_t}$ i.i.d. $\sim U([1, m])$
    - $\nabla ef(x_t) \leftarrow \nabla f(x_{et}) + \frac{1}{b_t}\sum_{i=1}^{b_t}(\nabla f_i(x_t) - \nabla f_i(x_{et}))$
  - Update $H_t$ and clip its entries to $[\lambda_t^-, \lambda_t^+]$
  - $y_t^0 \leftarrow x_t$
  - for $k = 0$ to $K - 1$ do
    - $\nabla Q_t(y_t^k) \leftarrow \nabla ef(x_t) + \frac{1}{\eta_t} H_t(y_t^k - x_t)$
    - $v_t^k \leftarrow \arg\min_{v\in\Omega}\langle \nabla Q_t(y_t^k), v\rangle$
    - $\gamma_t^k \leftarrow \min\left(\eta_t\langle \nabla Q_t(y_t^k), y_t^k - v_t^k\rangle \frac{1}{\|y_t^k - v_t^k\|_{2H_t}^2}, \gamma_t\right)$
    - $y^{k+1} \leftarrow y_t^k + \gamma_t^k(v_t^k - y_t^k)$
  - $x_{t+1} \leftarrow y_t^K$

*Output:* Last iterate $x_T$

The Stochastic Variance Reduced Frank-Wolfe (SVRF) methodis a variation of the stochastic Frank-Wolfe algorithm that uses a variance reduction strategy in conjunction with a stochastic gradient estimator to accelerate convergence in high-dimensional optimization problems. On the other hand, the gradient-based optimization technique known as the Adaptive Gradient (AdaGrad) algorithm modifies the gradient descent algorithm's learning rate to fit the geometry of the optimization problem. In this manner, when rare features do arise, they get huge step-sizes, which makes it possible for the algorithm to identify these potentially highly informative but uncommon features. In order to overcome this AdaGrad constraint, AdaSVRF (Combettes, Spiegel, and Pokutta 2020) integrates the SVRF method with AdaGrad. Specifically, AdaGrad can be computationally expensive at each iteration when used to minimize an objective across a limited set. The suggested technique seeks to resolve this problem while maintaining the desired descent direction features by including Frank-Wolfe into the optimization procedure.

In particular, AdaSVRF applies $K$ iterations of SVRF to

$$\min_{x\in\Omega} Q_t(x) = f(x_t) + \langle \nabla ef(x_t), x - x_t\rangle + \frac{1}{2\eta_t}\|x - x_t\|_{H_t}^2 \quad (7)$$

where $\eta_t > 0$ is a time-varying learning rate, $H_t$ is any diagonal matrix with positive entries, and $\|x\|_{H_t} = \sqrt{\langle x, xH_t\rangle}$. Equation (7) represents the same sub-problem that AdaGrad needs to solve at each iteration. Additionally, entries of $H_t$ are clipped to ensure $\sup_{t\in\mathbb{N}} \lambda_{\max}(H_t) < +\infty$.

At every iteration $t = s_k$, $k \in \mathbb{N}$, the algorithm computes the exact gradient of the iterate, saves it into memory, then builds the gradient estimator $\nabla ef(x_t)$ 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:

$$H_t = \delta + \nu uu^T X_t \sum_{s=0}^{\infty}(\nabla ef(x_s))^2 \quad (8)$$

where $\delta = 10^{-8}$ is a constant useful to prevent division by zero.

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

$$s_k = \frac{2k + k_0 - 2}{k_0}$$
$$K = 5$$
$$b_t = 8(2k_0 + 1 + 1)(K + 3)$$
$$\gamma_t = \frac{2}{t + 2}$$
$$\eta_t = \frac{\lambda_t^-}{L}$$

where $k_0 \in \mathbb{N}^+$ is a constant, useful in practice to prevent computing exact gradients too often in the early rounds ($k_0 = 4$ in the proposed implementation), and $L$ is the Lipschitz constant, which in the suggested implementation is set at 0.001.

## VII. Computational Experiments

The aforementioned algorithms have been tested using real-world data on the following regression and classification models: Linear Regression, L1-Regularized Logistic Regression and Shallow Feed-Forward Neural Network. For each model we compared both the algorithms on two different datasets. AdaSVRF uses the learning rates described in its respective section. SFW's learning rates were set respectively as:

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

## VIII. Convex Objectives

The assignments using Convex Objectives are explained in this section. The duality gap, which is calculated with regard to CPU Time and Epochs, has been used to compare the methods on a convex optimization problem. The definition of the duality gap, which acts as a measure of convergence, is

$$\text{dualitygap}(\bar{x}_t) = \ |\nabla f(\bar{x}_t), x_t - \text{lmo}(\nabla f(\bar{x}_t)\ | \quad (9)$$

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

## IX. Linear Regression

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data. The goal is to find the best-fitting line that minimizes the sum of the squared differences between the observed and predicted values of the dependent variable. We use a constraint that induces sparsity via the L1-norm. The objective function to minimize for the task, the Mean-Squared Error (MSE), is shown in the following equation.

$$f(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i^T w)^2$$

(10)

s.t. $\|x\|_1 < 100$

The complete gradient and the mini-batch gradient needed to execute the algorithms for this particular task are defined by equations 11 and 12, respectively:

$$\nabla f(w) = -2\frac{n}{X^T}(y - Xw) \tag{11}$$

$$\nabla f_i(w) = -2|idx|_{i \in idx} x_i^T (y_i - x_i^T w) \tag{12}$$

The radius $r$ for the $\ell_1$-ball was set to 100, and the algorithms were executed for 100 epochs. The Table I/III displays the MSE values that were determined by the algorithms, whereas the Table II/IV illustrates how every algorithm respects the restriction. The behavior of the duality gap for the Frank-Wolfe algorithms is illustrated in Figures 1/2.

### A. *Boston Dataset*

The goal is to forecast the median value of owner-occupied housing $(y_1, \ldots, y_n)$ in the Boston, Massachusetts area. The dataset comprises 404 samples with 13 characteristics.

TABLE I: Mean Squared Error (MSE)

| Method | MSE |
|---|---|
| SFW | 726.92 |
| AdaSVRF | 824.54 |

TABLE II: L1-Norm of Parametersn

| Algorithm | L1-Norm |
|---|---|
| SFW | 29.26 |
| AdaSVRF | 56.53 |

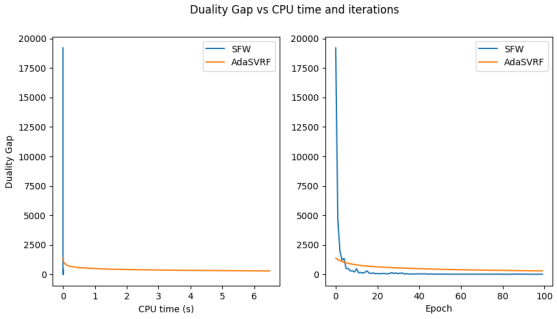


Fig. 1: Duality Gap vs. CPU time and iteration

### B. *California Housing Dataset*

We utilized another dataset to gain a deeper understanding of the behavior of both algorithms: the California Housing dataset. This dataset contains information about houses in a given California district, along with summary statistics based on the 1990 census data. It consists of 16.512 samples and 8 characteristics.

TABLE III: Mean Squared Error (MSE)

| Method | MSE |
|---|---|
| SFW | 5.15 |
| AdaSVRF | 645.3 |

TABLE IV: L1-Norm of Parameters

| Algorithm | L1-Norm |
|---|---|
| SFW | 1.29 |
| AdaSVRF | 44.95 |

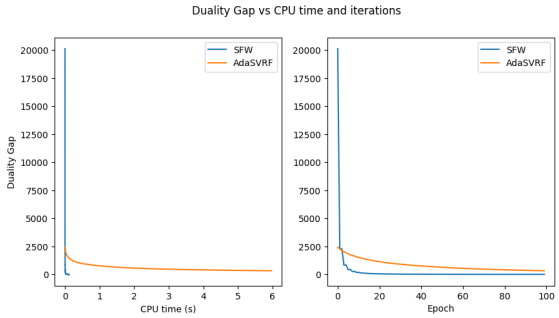


Fig. 2: Duality Gap vs. CPU time and iteration

From the experiment carried out of the two datasets, it is evident that AdaSVRF yields the worst score and we can notice a huge difference in term of MSE. It can be also noticed that both the algorithms respect the constrain and that AdaSVRF takes the longest time to execute, which is expected due to the computational complexity of its iterations. Both the algorithms appear to converge, but it is worth noticing that the SFW algorithm achieves significantly higher duality gap scores in the initial iterations.

## X. Nonconvex Goals

The classification tasks with nonconvex objectives are the main topic of this section. In particular, the following architectures are used to compare the algorithms' performances:

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

We also plot the metrics versus CPU Time and Epochs.

## XI. Logistic Regression Regularized (Lasso, L1-Norm penalty)

A regularized form of the logistic regression that includes the L1-Norm penalty term on the weights $w$ is called the Lasso Logistic Regression. By precisely shrinking the coefficients to zero, feature selection is carried out. Equation 13 illustrates the Regularized Cross Entropy as the objective function to be minimized:

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i x_i^T w)) + \lambda \|w\|_1 \tag{13}$$

s.t $\|x\|_1 \leq 100$
$\lambda > 0$ regularization hyperparameter

The full gradient and the mini-batch gradient used to run the algorithms for this specific task are shown, respectively, in Equation 14 and Equation 15:

$$\nabla f_i(w) = -\frac{1}{n}\sum_{i=1}^{n} x_i^T y_i \frac{\exp(-y_i(x_i^T w))}{1 + \exp(-y_i(x_i^T w))} + \text{sign}(w)\lambda \tag{14}$$

$$\nabla f_i(w) = -\frac{1}{|idx|}\sum_{i\in idx} x_i^T y_i \frac{\exp(-y_i(x_i^T w))}{1+\exp(-y_i(x_i^T w))} + \text{sign}(w)\lambda \tag{15}$$

To perform this task, the algorithms ran for 100 epochs. The radius for the $\ell^1$-ball was set to 100, to represent the constraint imposed on the problem. The Table V/VII displays the accuracy scores obtained by the algorithms , whereas the Table VI/VIII illustrates how every algorithm respects the restriction. The behavior of the duality gap for the Frank-Wolfe algorithms is illustrated in Figures 3/4.

### A. *California housing Dataset*

We utilized another time the California Housing dataset, but in this case we set a treshold based on the mean of the samples to advantage the binary classification done by the Lasso Logistic Regression. This dataset contains information about houses in a given California district, along with summary statistics based on the 1990 census data. It consists of 16.512 samples and 8 characteristics.

TABLE V: Accuracy score

| Method | Accuracy |
|---|---|
| SFW | 0.56 |
| AdaSVRF | 0.52 |

TABLE VI: L1-Norm of Parameters

| Algorithm | L1-Norm |
|---|---|
| SFW | 2.12 |
| AdaSVRF | 27.12 |

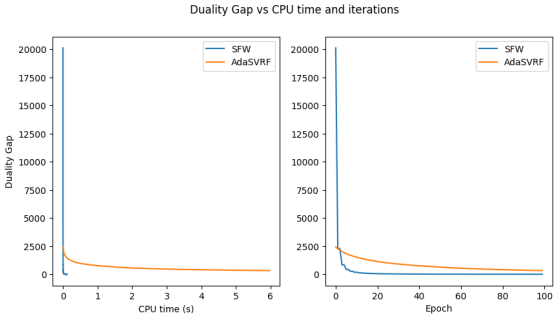


Fig. 3: Duality Gap Vs CPU time and iteration

### B. *Breast Cancer Dataset*

We tested the two algoritem also in the Breast Cancer dataset. It contains features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. The goal is to classify tumors as either malignant (cancerous) or benign (non-cancerous). It consists of 455 samples and 30 characteristics.

TABLE VII: Accuracy score

| Method | Accuracy |
|---|---|
| SFW | 0.88 |
| AdaSVRF | 0.12 |

TABLE VIII: L1-Norm of Parameters

| Algorithm | L1-Norm |
|---|---|
| SFW | 37.25 |
| AdaSVRF | 78.43 |

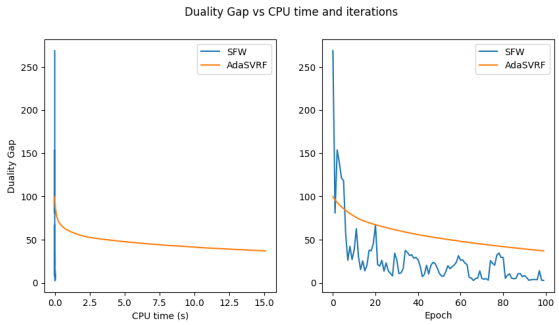


Fig. 4: Duality Gap Vs CPU time and iteration

From the tables and the plots, it is evident that the SWF outperforms the AdaSVRF, the first one obtain a high accuracy in both the datasets. Notably, in the Brest Cancer Dataset the gap between the accuracy of the two algorithms is really huge. Furthermore, it is also clear that both the algoritems satisfy the constraint. Notably, SFW, which achieves the best accuracy result, produces a solution with a lower L1-norm. Regarding the behavior of the two algorithms which is shown in the two plots we can see that in the first dataset both the algorithms have a smooth shape and converge, instead in the second dataset we can see an up-down behaviour of the SFW but it still seems to converge.

## XII. Shallow Feed-Forward Neural Network

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

**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 $X$:

$$Z1 = W_1 X^T + b_1$$

$$A1 = \text{ReLU}(Z1)$$

$$Z2 = W_2 A1 + b_2$$

$$Prediction : A2 = \sigma(Z2)$$

**Objective Function**
Cross Entropy, which is shown in Eqution 16.

$$J(W) = -\frac{1}{n}\sum_{i=1}^{n} y_{\text{train}}^{(i)} \log(A2^{(i)}) + (1 - y_{\text{train}}^{(i)}) \log(1 - A2^{(i)}) \tag{16}$$

**Backward Propagation**
Used to compute the gradients in the train phase. Takes as input $X_{\text{train}_i}$, $y_{\text{train}_i}$:

$$\frac{dZ2}{dA2} = A2 - y_{\text{train}}$$

$$\frac{dW_2}{dt} = \frac{1}{n}\frac{dZ2A1^T}{dt}$$

$$\frac{db2}{dt} = \frac{1}{n}\sum_{i=1}^{n}\frac{dZ2,i}{dt}$$

$$\frac{dZ1}{dt} = W_2^T \frac{dZ2}{dt}\text{ReLU}'(Z1)$$

$$\frac{dW_1}{dt} = \frac{1}{n}\frac{dZ1X_{\text{train}}}{dt}$$

$$\frac{db1}{dt} = \frac{1}{n}\sum_{i=1}^{n}\frac{dZ1,i}{dt}$$

**Experiment Description**
The implemented architecture has 2 neurons and is constrained using radius $r = 1$. To perform this task, the algorithms ran for 300 epochs. The Table IX/X displays the accuracy scores obtained by the algorithm, the behavior of the Loss in relation with the CPU time and the iterations is illustrated in Figures 5/7, and the behavior of the Accuracy in relation with the CPU time and the iterations is illustrated in Figures 6/8. Since, in this task, the behaviour and the score of the two algorithms were mostly identical for both the datasets, we have decided to advantage the binary classification to better capture the different behaviours of the algorithms.

### A. *Diabetes dataset*

The Diabetes Dataset can be used to explore various aspects of diabetes prediction, the target variable is binary wich indicate the presence or absence of diabetes. This dataset was originally obtained from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to use diagnostic measurements to establish whether or not a patient has diabetes. It consists of 455 samples and 30 characteristics.

TABLE IX: Accuracy score

| Method | Accuracy |
|---|---|
| SFW | 0.94 |
| AdaSVRF | 0.95 |

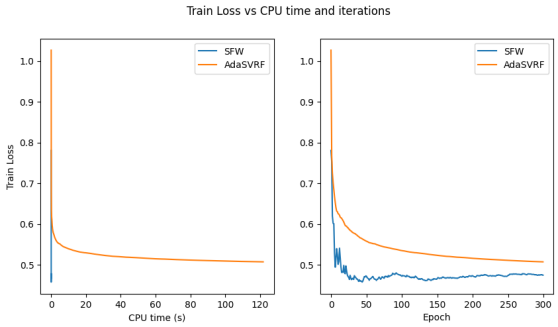


Fig. 5: Train Loss Vs CPU time and iteration

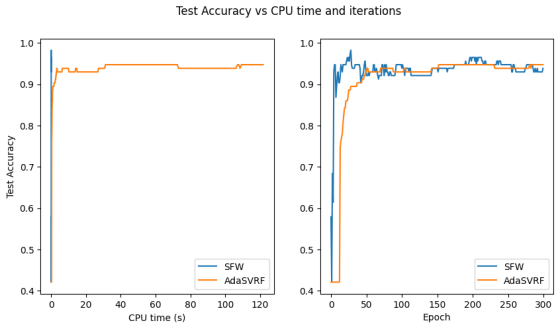


Fig. 6: Accuracy Vs CPU time and iteration

### B. *Breast Cancer Dataset*

Additionally, we tested the two algoritem also in the Breast Cancer dataset. It contains features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. The goal is to classify tumors as either malignant (cancerous) or benign (non-cancerous). It consists of 455 samples and 30 characteristics.

TABLE X: Accuracy score

| Method | Accuracy |
|---|---|
| SFW | 0.97 |
| AdaSVRF | 0.98 |

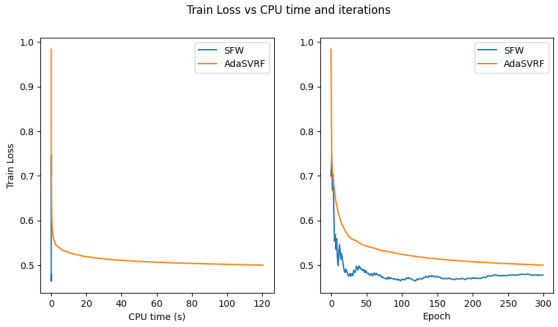


Fig. 7: Train Loss Vs CPU time and iteration

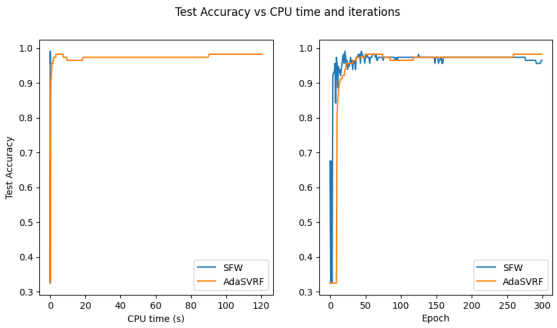


Fig. 8: Accuracy Vs CPU time and iteration

From the results on the different datasets, the algorithms seem to converge and their behavior is so similar, and we can notice that SFW is the fastest algorithm in terms of convergence but, in terms of accuracy, its performance is a bit lower than the AdaSVRF, which reaches high accuracy values but takes longer time.

## XIII. Conclusions

The suggested experiments have demonstrated how each method behaves on a range of problems and architectures in a constrained optimization framework. It might be important to note that any algorithm may behave differently when applied to distinct datasets or when using somewhat different topologies of models. From the experiments done, the SFW stands out as the best, both because it can achieve satisfactory performance despite being limited and especially because it takes much less time to converge. AdaSVRF may perform poorly in most of the suggested tests because it is built for robustness in large-scale situations, where it should perform exceptionally well. In fact, when the complexity of the model increases (as in the shallow neural network) the AdaSVRF performs better than the SFW and these improvements should be more accentuated when the complexity of the global problem increases. It could be interesting to see the behaviour of the AdaSVRF in a large-scale or computationally demanding optimization problem but, following the guidelines in paragraph 4.2 of (Combettes, Spiegel, and Pokutta 2020), it was decided to not try this experiment because variance reduction can be ineffective in the training of deep neural networks and also due to the AdaSVRF's lengthy execution time, it was not practical to assess it given the available resources.

## XIV. Refrences

Combettes, C. W.; and Pokutta, S. 2021. Complexity of lin ear 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 optimiza tion. arXiv preprint arXiv:2009.14114.

Luo, L.; Xiong, Y.; Liu, Y.; and Sun, X. 2019. Adaptive gra dient 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 An nual Allerton Conference on Communication, Control, and Computing (Allerton).

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