Adaptive stochastic Frank-Wolfe

Authors: Shab Cesare Akira Pompeiano and Antoni Valls Cifre
Dipartimento di Matematica, Università degli Studi di Padova

Abstract: The Frank-Wolfe method, a prominent first-order optimization algorithm, has gained traction due to its edge over projected gradient methods, especially in machine learning. However, its application in non-convex scenarios remains less explored. We delve into two foundational papers: Reddi et. al.'s Stochastic Frank-Wolfe algorithm (SFW) and Combettes et. al.'s AdaSVRF. Our computational experiments, using Iris and Obesity datasets, demonstrate AdaSVRF's superior performance over SFW, leveraging AdaGrad's adaptive learning rates and variance reduction. Such findings highlight AdaSVRF's potential in optimizing sparse problems with computational efficiency.

I. INTRODUCTION

The Frank-Wolfe method (also called conditional gradient method or reduced gradient method) is an iterative first-order optimization algorithm. First proposed by Marguerite Frank and Philip Wolfe from Princeton University in 1956, it has seen an impressive revival recently due to its nice properties compared to projected gradient methods, in particular for machine learning applications. Advantages of the Frank-Wolfe algorithm include that it is simple to implement; it results in a projection-free computation; and its ability to exploit structured constraints [1].

Algorithm 1 Frank-Wolfe method

```
1: Choose a point $x_1 \in C$
2: for k = 1, ... do
3:     Set $\hat{x}_k = \arg\min_{x\in C} \nabla f(x_k)^\top (x - x_k)$
4:     If $\hat{x}_k$ satisfies some specific condition, then STOP
5:     Set $x_{k+1} = x_k + \alpha_k(\hat{x}_k - x_k)$, with $\alpha_k \in (0, 1]$ suitably chosen stepsize
6: end for
```

The Frank-Wolfe method operates by iteratively approximating the initial constrained problem through the resolution of a sequence of linear subproblems. During each iteration, the algorithm determines the direction of the minimum linear approximation of the objective function within the constraints of the feasible set. This iterative approach gradually improves the solution until convergence is achieved or specific stopping conditions are met.

At each iteration of the algorithm it solves the problem:

$$\hat{x}_k = \arg\min_{x\in C} \nabla f(x_k)^\top (x - x_k). \tag{1}$$

The fundamental theorem of linear programming states that in a linear program with a non-empty feasible region (polytope), at least one optimal solution exists at an extreme point (vertex) of the polytope. This property is essential as it enables us to focus the search for the optimal solution solely on the vertices, rather than exploring the entire feasible region. The Frank-Wolfe algorithm leverages this property by iteratively moving towards the optimal solution using linear approximations of the objective function within the feasible region. It identifies the best current vertex and advances towards a new vertex that maximizes improvement in the objective function. This process continues until convergence [2].

Despite the extraordinary success of FW approaches in the convex setting, little research has been conducted to explore their relevance in the context of non-convex cases.

We now present the two papers that have been fundamental pillars of our work.

A. Stochastic Frank-Wolfe Methods for Non-convex Optimization [3]

This paper studies Frank-Wolfe methods for non-convex stochastic and finite-sum optimization problems, *i.e.* problems of the form:

$$\min_{x\in\Omega} F(x) = \begin{cases} \mathbb{E}_z[f(x,z)], & \text{(stochastic)} \\ \frac{1}{n}\sum_i f_i(x), & \text{(finite-sum)} \end{cases} \tag{2}$$

The Frank-Wolfe algorithm can be very handy. Specifically, when considering the the stochastic case, which is the one that will command our special attention. Calculating the stochastic approximation to $\nabla(F)$ is usually not expensive while on the other hand the cost projecting onto the constrained set can be very high in practice. Frank-Wolfe can avoid this expensive step by means of a linear oracle.

Despite the importance of non-convex machine learning models such as deep learning models, known variants of Frank Wolfe do not perform well on non-convex optimization problems.

Reddi et. al. [3] proposes the Stochastic Frank-Wolfe algorithm (SFW) which is proven to be faster than existing Frank-Wolfe methods. Two variants are also proposed, which aim to improve the convergence rate of SFW by implementing variance reduction techniques: SVFW and SagaFW, based respectively on variance reduction algorithms SVRG and SAGA.

In the stochastic case we assume that F is L-smooth and that function f is G-Lipschitz, *i.e.* $\max_{x\in\Omega, z\in\Xi} \|\nabla f(x,z)\| \leq G$. The criteria used to determine convergence is the Frank-Wolfe gap:

$$\mathcal{G}(x) = \max_{v\in\Omega}\langle v - x, -\nabla F(x)\rangle \qquad (3)$$

For constrained problems we use this quantity for the convergence analysis instead of $\|\nabla(F)\| = 0$ (as in unconstrained problems). For nonconvex functions $\mathcal{G}(x) = 0$ if and only if x is a stationary point.

The authors use three different oracle models in order to compare the convergence speed of different algorithms:

- Stochastic First-Order Oracle (SFO) [4]
- Incremental First-Order Oracle (IFO) [5]
- Linear Optimization Oracle (LO)

In order to calculate the approximation to the $\nabla(F)$, the authors take inspiration from the Robins-Monro approximation which is widely used in convex optimization [6].

Algorithm 2 Non-convex SFW

```
Input: Start point $x_0 \in C$, step-sizes $\{\gamma_i\}_{i=0}^{T-1}$,
       batch-sizes $\{b_i\}_{i=0}^{T-1}$.
1: for t=0 to T-1 do
2:    Uniformly randomly pick i.i.d samples $\{z_1^t, \dots, z_{b_t}^t\}$ according to the distribution $\mathcal{P}$.
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
```

The SFW for non-convex optimization (Algorithm 2) differs from the classic Frank-Wolfe (Algorithm 1) in two parts. First of all, it is possible to get an unbiased approximation of $\nabla(F)$ sampling independently from distribution $\mathcal{P}$ samples z for the mini-batch. The output after T iterations is then randomly selected among all iterates.

The study is however limited as the step size and the mini-batch sizes are selected arbitrarily depending on the specific case.

What the authors find is that the rates of SFO and LO complexity of SFW seem somehow worse than those obtained with Stochastic Gradient Descent for non-convex optimization, however the convergence criteria differ and thus the need to explore the difference in convergence remains. Nonetheless, the convergence rates seem to be only slightly worse than SFW in the convex setting.

B. Projection-Free Adaptive Gradients for Large-Scale Optimitzation [7]

This second paper marks the pioneering effort in using adaptive gradients to enhance the efficiency of SFW. It is the first instance of such an approach being employed.

Algorithm 3 Adaptive Gradient (AdaGrad)

```
Input: Start point $x_0 \in C$, offset $\delta > 0$, learning rate $\eta > 0$.
1: for t=0 to T-1 do
2:    Update the gradient estimator $\tilde{\nabla} f(x_t)$
3:    $H_t \leftarrow \mathrm{diag}\left(\delta 1 + \sqrt{\sum_{s=0}^{t}\tilde{\nabla} f(x_s)^2}\right)$
4:    $x_{t+1} \leftarrow \arg\min_{x\in C}\eta\langle\tilde{\nabla} f(x_t), x\rangle + \frac{1}{2}\|x - x_t\|_{H_t}^2$
5: end for
```

The specialty of AdaGrad (Algorithm 3) [8, 9] is how the algorithm uses different learning rates for different features. The features which are most influencing the gradient descent direction will have a smaller learning rate compared to features which seldom contribute to the gradient direction. This behaviour makes AdaGrad suitable for sparse problems, where the useful contribution of infrequent features would be otherwise lost. This is done by accumulating the squared approximation of the gradient for each feature, which is used to inversely scale and thus adapt the learning rate for each feature.

In Algorithm 3, the matrix $H_t \in \mathbb{R}^{n\times n}$ is diagonal and the default value of the offset hyperparameter is $\delta \leftarrow 10^{-8}$, which prevents from dividing by zero.

Each iteration of AdaGrad can be very expensive because the minimization problem to solve would be a non-euclidean projection, thus the authors suggest to avoid the projection and solve the AdaGrad problem partially with Frank-Wolfe until some accuracy is reached [10]. Although this simplification would retain only a fraction of the information from the AdaGrad, the authors claim that performing a fixed small number of iterations (*e.g.* $K \sim 5$) on the subproblems would be sufficient in practice.

Combining Stochastic Frank-Wolfe and AdaGrad, the authors propose a Frank-Wolfe with adaptive gradients. The algorithm maintains the computational efficiency of Frank-Wolfe while integrating AdaGrad's adaptive learning rates. By approximating AdaGrad's non-Euclidean projection with the Frank-Wolfe algorithm, computational intensity is reduced, making it a suitable approach for sparse problems. With this combination, the algorithm efficiently adapts to the importance of different features, notably optimizing problems with sparse feature contributions.

Some derived algorithms are presented: AdaSFW, AdaSVRF and AdaCSFW. In our project, our attention is directed towards the second approach, which notably mitigates gradient estimator variance through the application of variance reduction techniques.

At every iteration $t = s_k$, $k \in \mathbb{N}$, AdaSVRF (Algorithm 4) computes the exact gradient and use it in the following iterations from this snapshot in order to compute the gradient estimator $\tilde{\nabla} f(x_t)$. In order to ensure that the eigenvalues of H_t do not diverge (key aspect of the convergence analysis that they do), they propose to clip the entries of H_t [11] . The variance

Algorithm 4 AdaSVRF

```
Input: Start point $x_0 \in C$, snapshot times $s_k < s_{k+1}$ with $s_0 = 0$, batch-sizes $b_t \in \mathbb{N}\backslash\{0\}$, bounds $0 < \lambda_t^- \leq \lambda_{t+1}^- \leq \lambda_{t+1}^+ \leq \lambda_t^+$, number of iterations $K \in \mathbb{N}\backslash\{0\}$, learning rates $\eta_t > 0$, step-size bounds $\gamma_t \in [0, 1]$
 1: for t=0 to T-1 do
 2:     if t$\in \{s_k | k \in \mathbb{N}\}$ then
 3:         $\tilde{x}_t \leftarrow x_t$
 4:         $\tilde{\nabla} f(x_t) \leftarrow \nabla f(\tilde{x}_t)$
 5:     else
 6:         $\tilde{x}_t \leftarrow x_{t-1}$
 7:         $i_1, \dots, i_{b_t} \sim \mathcal{U}[1, m]$
 8:         $\tilde{\nabla} f(x_t) \leftarrow \nabla f(\tilde{x}_t) + \frac{1}{b_t} \sum_{i=i_1}^{i_{b_t}} (\nabla f_i(x_t) - \nabla f_i(\tilde{x}_t))$
 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:         $\nabla Q_t(y_k^{(t)}) \leftarrow \tilde{\nabla} f(x_t) + \frac{1}{\eta_t} H_t(y_k^{(t)} - x_t)$
14:         $v_k^{(t)} \leftarrow \arg\min_{v \in C} \langle \nabla Q_t(y_k^{(t)}), v \rangle$
15:         $\gamma_k^{(t)} \min \{\eta_t \frac{\langle \nabla Q_t(y_k^{(t)}, y_k^{(t)} - v_k^{(t)} \rangle}{\|y_k^{(t)} - v_k^{(t)}\|_{H_t}^2}, \gamma_t\}$
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
```

$\mathbb{E}[\|\tilde{\nabla} f(x_t) - \nabla f(x_t)\|_2^2$ is significantly reduced.

II. COMPUTATIONAL EXPERIMENTS

In this section we describe our experiment and present the performance results of our implementation of SFW and AdaSVRF algorithms on the two aforementioned datasets: Iris and Obesity. Whereas performance is measured during training on the test set, in terms of: Loss over Epoch, Loss over CPU Time (s), Accuracy over Epoch and Accuracy over CPU Time (s).

We implement Stochastic Frank-Wolfe and AdaSVRF following the papers from Combettes et. al. [7] and Reddi et. al. [3].

Both neural networks have one fully-connected hidden layer of 64 unit and ReLU activation functions. Each layer is constrained into a l_∞-ball with l_2-diameter equal to 40 times the expected l_2-norm of the Glorot uniform initialized values. For both implementations we initialize the weights of the neural networks using the Glorot (or Xavier) uniform initialization. The Glorot initialization helps preventing training issues like vanishing/exploding gradients, accelerates convergence, and promotes better information flow through layers by initializing weights in a balanced way [12]. The weights are drawn from a distribution with zero mean and a specific variance. For the uniform distribution, the weights are drawn from a range $[-\text{limit}, \text{limit}]$, where limit is $\sqrt{6/\text{fan\_in} + \text{fan\_out}}$, with fan_in representing the number of input units in the weight tensor and fan_out is the number of output units.

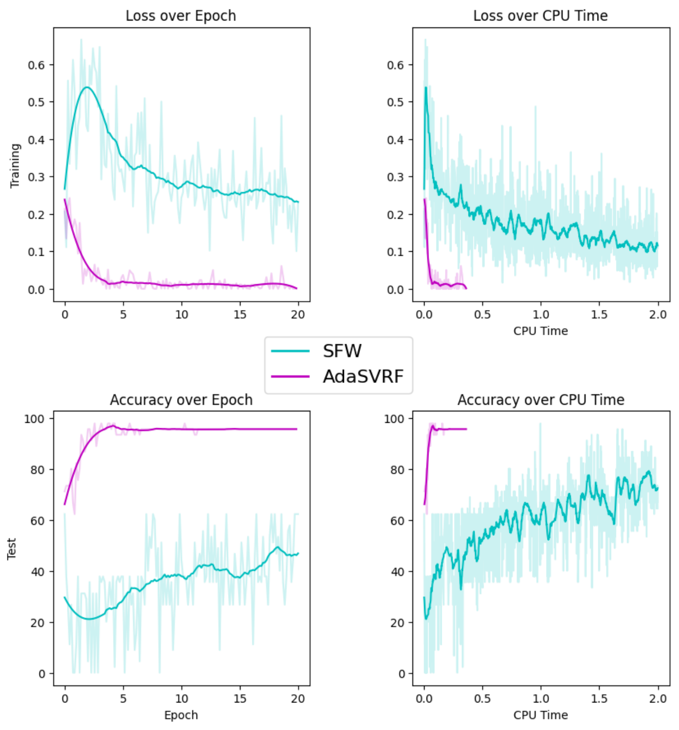

FIG. 1: Neural network with one fully-connected hidden layer on the Iris dataset.

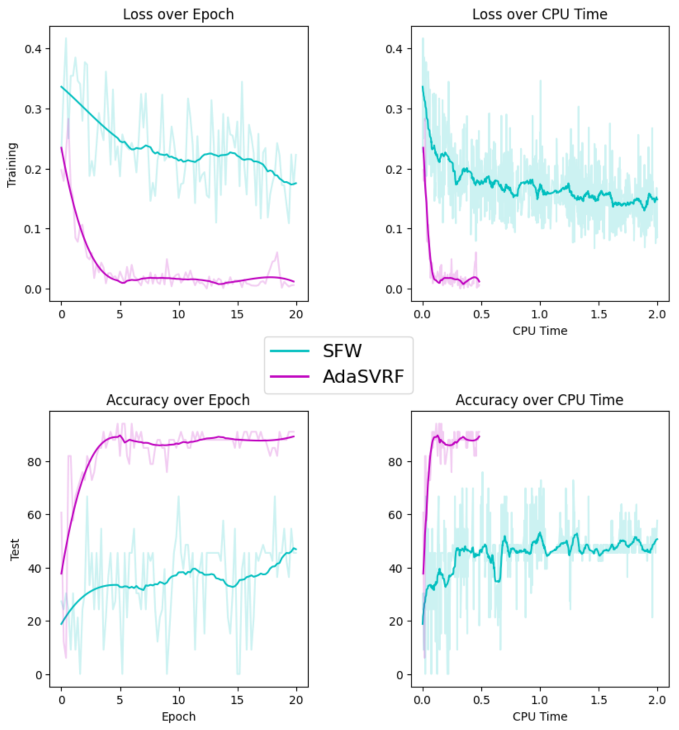

FIG. 2: Neural network with one fully-connected hidden layer on the Obesity dataset.

Calculating the expected l_2-norm of this uniform distribution over the interval [-limit, limit] is equivalent to calculating the root mean square (RMS) of the uniform distribution, which can be obtained with the formula:

$$\text{RMS} = \sqrt{a^2 + b^2 + ab/3} \tag{4}$$

where a and b represent the interval of the uniform distribution. So for Glorot uniform initialization, a is -limit and b is limit.

So, the expected l_2-norm of Glorot uniform initialized values would be $\sqrt{2 \times \text{limit}^2/3}$.

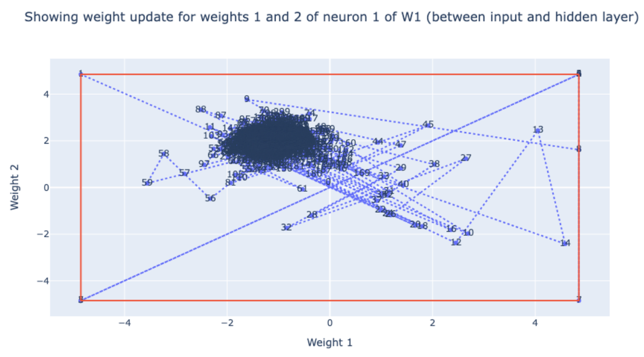

FIG. 3: Weight update SFW.

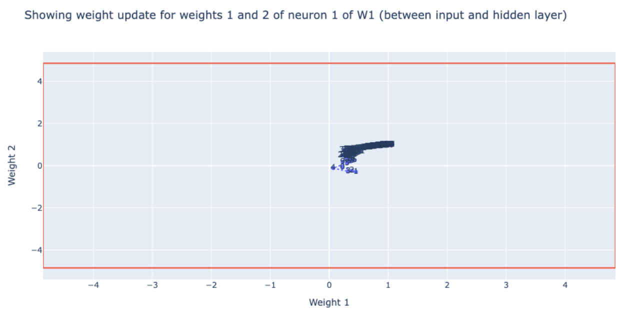

FIG. 4: Weight update AdaSVRF.

In order to test the correct execution of the implementations, we monitor the weight updates inside the constraints of different neurons. For an example see Fig. 3. As we commented before, what the Frank-Wolfe method is doing here is taking advantage of the geometry of the space in which the weights of the neural network are constrained. Bounding them into a l_∞-ball means that the minimum linear approximation of the objective function will be on one of the vertex of the feasible set.

A. Iris dataset

We train our neural networks on the well-known iris classification dataset [13, 14]. It consists of 150 samples of iris flowers from three different species (Setosa, Versicolor, and Virginica). Each sample includes measurements of four features: sepal length, sepal width, petal length, and petal width. We set *batch size* $\leftarrow 12$ in SFW and *batch size* $\leftarrow 12$, $K \leftarrow 5$, and $\eta \leftarrow 10^{-1/2}$ in AdaSVRF. The results of both algorithms are presented in FIG. 1.

B. Obesity dataset

We retrain our neural networks on the Obesity dataset [15]. It consists of 110 samples of medical information collected from a variety of sources, including medical records, surveys and self-reported data. Each sample includes the following features: age, gender, height, weight and BMI. The neural network is the same we used for the iris dataset, as we get outstanding performances using the same configuration. We keep *batch size* $\leftarrow 12$ in SFW and *batch size* $\leftarrow 12$, $K \leftarrow 5$, and $\eta \leftarrow 10^{-1/2}$ in AdaSVRF. The results of both algorithms are presented in FIG. 2.

III. DISCUSSION

From the findings presented in Fig. 1 and 2, a striking observation is the superior performance of AdaSVRF over SFW across both datasets. By exploiting the advantages of sparse gradients through AdaGrad together with variance reduction, AdaSVRF significantly enhances the efficacy of the SFW algorithm. Specifically, in the Iris dataset (as depicted in Fig. 1), AdaSVRF achieves a loss nearing 0 and an accuracy rate close to 95%. This is in strong contrast to SFW, which records a loss of approximately 0.1 and an accuracy rate around 70%. Notably, AdaSVRF also attains the optimal solution with greater speed.

This superior performance of AdaSVRF is further confirmed by the results from the Obesity dataset, as illustrated in Fig. 2. Furthermore, the employment of variance reduction in AdaSVRF is evident in its smoother plotted lines compared to its counterpart.

Finally, a comparative analysis of weight updates between Fig. 3 and 4 underscores the precision of AdaSVRF. The adaptive learning rates introduced by AdaGrad enable AdaSVRF to execute weight updates with increased precision in terms of learning rate adjustments.

IV. CONCLUSION

We have implemented two Frank-Wolfe algorithms, namely SFW and AdaSVRF, and compared their efficiency and accuracy across multiple datasets. The findings indicated clear superiority of AdaSVRF across both databases.

The integration of sparse gradients via AdaGrad combined with variance reduction, was critical in assessing AdaSVRF's capabilities. Not only did this combination result in the achievement of near-perfect loss and accuracy metrics, but it also enabled the algorithm to outpace SFW in converging to the optimal solution. Nonetheless, the results demonstrate a consistency between different datasets.

However, our study comes with limitations. Namely, as future work we suggest training the algorithms on more datasets, specially larger and with more complexity (e.g. IMDB 8k subword representation or CIFAR).

In summary, this research highlights the enhancements brought about by AdaSVRF, a variant of SFW, specifically in the realm of constrained non-convex optimization challenges. We have illustrated the applicability of both AdaSVRF and SFW in addressing these complexities within a neural network framework. Notably, AdaSVRF consistently delivers on its claims of superior efficiency and performance.

[1] T. B. Adeline Tse, Jeff Cheung, "Frank-wolfe," 2021.
[2] Y. Nesterov, "Introductory lectures on convex optimization: A basic course," Vol. 87. Springer Science Business Media, 2013.
[3] S. J. Reddi, S. Sra, B. Poczos, and A. Smola, "Stochastic frank-wolfe methods for nonconvex optimization," 2016.
[4] A. Nemirovski and D. Yudin, *Problem Complexity and Method Efficiency in Optimization.* New York: John Wiley, 1983.
[5] A. Agarwal and L. Bottou, "A lower bound for the optimization of finite sums," 2015.
[6] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400 – 407, 1951.
[7] C. W. Combettes, C. Spiegel, and S. Pokutta, "Projection-free adaptive gradients for large-scale optimization," 2021.
[8] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011.
[9] H. B. McMahan and M. J. Streeter, "Adaptive bound optimization for online convex optimization," *CoRR*, vol. abs/1002.4908, 2010.
[10] G. Lan and Y. Zhou, "Conditional gradient sliding for convex optimization," *SIAM Journal on Optimization*, vol. 26, no. 2, pp. 1379–1409, 2016.
[11] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," 2019.
[12] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 249–256, PMLR, 13–15 May 2010.
[13] D. F. Andrews and A. M. Herzberg, *Iris Data*, pp. 5–8. New York, NY: Springer New York, 1985.
[14] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals Eugenics*, vol. 7, pp. 179–188, 1936.
[15] S. K. Mandala, "Obesity classification dataset," 2023.