Frank Wolfe algorithms for Deep Neural Networks

Ivan Kulazhenkov
Ivan.Kulazhenkov@studenti.unipd.it

Rodrigo Arriaza
rodrigoalejandro.arriazaalonzo@studenti.unipd.it

Andres Aranguren
andres.aranguren@studenti.unipd.it

Sebastian Rojas
josesebastian.rojasardila@studenti.unipd.it

1. Introduction

Since the beginning of Neural Network use within data science the optimization algorithm of choice has been Stochastic Gradient Descent(SGD). It has proved both well performing and transferable to many problems that exist in the field of machine learning. Consequently, we would like to explore new possibilities and analyze the performance of 4 variants of the Frank-Wolfe (FW) algorithm when optimizing the parameters of a deep neural network.

Usually when implementing SGD it is done under the assumption that the parameter space in which the weights of the Neural Network lie is unconstrained. However, similar to what is done in regularization, we can impose a constraint to the parameter space. This we will achieve through the use of the $\mathcal{L}^1$-ball as a parameter constraint. Normally with other methods this would require a projection function so that the gradient is projected into the constrained space. The strength of FW algorithms though, lie in the fact that they are a family of projection free algorithms. Rather than using the projection step of SGD algorithms, Frank Wolfe utilizes a linear minimization oracle to determine the lower gradient and update the weights according to the update rule.

The most basic steps of the algorithm work according to the formulas found below. For a function f where we have defined a constrained feasible region $\mathcal{P}$ we have:

$$v_t = \arg\min_{x \in \mathcal{P}} \langle \nabla f(x_{t-1}), x \rangle \tag{1}$$

And the subsequent update is given as:

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

For the step size α_t we will use $\frac{2}{k+2}$, for $k = 1, 2, ..., n$.

With this base optimization step we will implement the following variants utilizing this strategy: Stochastic Frank Wolfe (SFW), Stochastic Frank Wolfe with Momentum (MSFW), Online Stochastic Recursive Gradient based Frank Wolfe (ORGFW) and the Stochastic Variance Reduced Frank Wolfe (SVRF) algorithms. We will test and evaluate the effectiveness of all 4 algorithms for use with a custom neural network and using the $\mathcal{L}^1$-ball constraint for each one. We will compare and contrast the performance of each algorithm when doing classification for 2 data-sets.

2. Related work

For the implementation of this project we based our approach on 4 papers described below, that provide different analysis and implementation details for some of the Frank Wolfe variations.

2.1. Deep Neural Network Training with Frank-Wolfe

The main focus of the above paper published by Sebastian Pokutta et al [4] is to analyse the efficiency of projection free, conditional gradient, first-order methods for the optimization of deep neural networks. In particular the two algorithms on which this paper focuses are the Stochastic Frank Wolfe and its momentum boosted variant. Though the implementation of the above algorithms for use in deep neural networks remains the focal point of the article; there are significant comparisons made to the other papers that we discuss in our related work section. Experiments of particular interest to the authors are those involving SVRF, SPIDER-FW and ORGFW algorithms. By way of using constrained optimization algorithms for neural network optimization the authors needed to define and use a single feasible region for their neural network problem. In their case the point was to prove the viability for such an operation. They did so through the use of the single feasible region and an $\mathcal{L}^p$-norm bound for constrains on both the weights and bias parameters within each layer. To give an example of this they showed a $\mathcal{L}^2$-norm bound would be given by the following equation:

$$\mathbb{E}(\|x\|_2) = \frac{n\sigma\Gamma(n/2 + 1/2)}{\sqrt{2}\Gamma(n/2 + 1)} \tag{3}$$

As such the diameter of the feasible region would be given as

$$\mathcal{D}(\mathcal{C}) = 2w\,\mathbb{E}(\|x\|_2) \tag{4}$$

where $\mathcal{C}$ is the feasible region, w is a fixed width and $w > 0$. Using these constraints alongside SFW the authors were able to match the performance of SGD with weight decay and improve it in the case of normal SGD.

Critical discoveries in this work are the strength of momentum based SFW in terms of general performance and when compared to the more complex algorithms analyzed. For example, the SVRF and ORGFW algorithms though achieve marginally stronger performance are way harder to implement and require many more gradient computations to achieve the equivalent result. We also can see that with SFW the authors attained comparable results to deep neural networks optimized with SGD. Both $\mathcal{L}^2$-norm and $\mathcal{L}^\infty$-norm parameter constraints were tested with $\mathcal{L}^2$-norm performing the better of the two.

2.2. Variance-Reduced and Project-Free Stochastic Optimization

The following paper proposes two stochastic Frank-Wolf variants which improve algorithm's performance in terms of the number of stochastic gradient evaluations required to achieve a 1-e accuracy, this is obtained by utilizing Nesterovs's acceleration technique and the variance reduction idea which will be further explained. Initially we consider the most common optimization problem in machine learning:

$$\min_{w \epsilon \Omega} f(w) = \min_{w \epsilon \Omega} \frac{1}{n} \sum_{i=1}^{n} f_i(w) \tag{5}$$

More specifically the case where n corresponds to a large number of training examples and the domain admits fast linear optimization, an ideal scenario for multiclass classification problems.

2.2.1 Stochastic variance reduced gradients

The variance reduced algorithm at a point w with a snapshot w_0 is defined by:

$$\nabla f(w; w_o) = \nabla f_i(w) - (\nabla f_i(w_0) - \nabla f(w_0)) \tag{6}$$

Where i consists on a random selected point from 1,..,n, being n the number of examples. W_0 is called the snapshot which is a decision point form a previous iteration, its gradient is described by $\nabla f(w_0)$. Therefore the computation of $\nabla f(w; w_0)$ requires only two stochastic gradient evaluations. Gradient at some point $\nabla f(w)$ and gradient of snapshot $\nabla f(w_0)$. It is important to add that the term $\nabla f_i(w_0) - \nabla f(w_0)$ is the correction term used to reduce the variance of the stochastic gradient. Compared to the standard Frank-Wolfe the proposed algorithm replaces the exact gradient with the average a mini-batch of stochastic gradients, taking snapshots at certain moments of the iterations. The convergence rate of SVRF is given by the following theorem:

Theorem 2.1.

$$\gamma = \frac{2}{k+1}, m_k = 96(k+1), N_t = 2^{t+3} - 2 \tag{7}$$

Where γ consists on the stepsize or adjusted learning rate of the algorithm, m_k the number of samples to compute the average gradient ∇_k of the minibatch, and finally N_t being the.....

While the required number of linear optimization or oracle calls remains the same as previous literature, the improvement of this algorithm is evidenced in the number of stochastic gradient evaluations. For smooth function there's an improvement from $O(\frac{1}{\epsilon^2})$ to $O(\frac{1}{\epsilon^{1.5}})$ and for smooth and strongly convex functions there's an improvement from $O(\frac{1}{\epsilon})$ to $O(ln\frac{1}{\epsilon})$

2.2.2 Stochastic variance-reduced conditional gradient sliding

The second algorithm applies the previously described variance reduction technique to the conditional gradient method CGS, which is a method for convex optimization that calls a linear optimization (LO) oracle to minimize a series of linear functions over a feasible set. This method differs from the classic conditional gradient, since the conditional gradient sliding proposed herein allows to skip the computation of gradients at certain times, hence it can reach the optimal complexity bounds not only in terms of the number of times the oracle LO is called but also in the number of gradient evaluations. Accordingly, as described in the previous algorithm of section 2.2.1, the stochastic gradient is replaced with the average of a mini.batch of variance reduced stochastic gradients taking snapshots at certain moments.

2.3. Efficient Projection-Free online methods with Stochastic Recursive Gradient

The paper deals with algorithms for solving online convex optimization problems OCO which can be divided in two categories: (i) projection-based methods, which can be computationally expensive and (ii) projection free methods, such as Frank-Wolfe type methods which are more suitable more high dimensional applications, since in contrast to the projection methods, only require to solve linear optimization problems over the constraint set. Nonetheless, this methods suffer from an intense trade off between regret and computational complexity, since currently there are no projection free methods with both an optimal regret bound and

a low computational cost. In order to overcome this difficulty the paper proposes two new methods, Online stochastic Recursive Gradient-based Frank-Wolfe (ORGFW) and Meta-ORGFW (MORGFW), in the stochastic and adversarial setting respectively, both settings regrets are described as follows:

Adversarial setting

$$R_t = \sum_{t=1}^{T} f_i(x_t) - \min_{x \epsilon C} \sum_{t=!}^{T} f_t(x) \tag{8}$$

Stochastic setting

$$SR_t = \sum_{t=1}^{T} (f_t(x_t) - f_t x^*) \tag{9}$$

The last one being very similar to the problem of stochastic optimization, however in OCO problems we are interested in producing a sequence of decision variables with low regret, on the other side in the stochastic optimization aims to find a minimizer of the loss function, where the performance is measure by the convergence rate.

2.3.1 Online Stochastic Recursive Gradient-Based Frank-Wolfe

2.3.2 Stochastic setting

In the stochastic setting the ORFGW algorithm receives through the iterations the loss $f_t(x)$ as well as the stochastic gradients $\nabla F_T(x_{t-1}, \epsilon_t)$ and $\nabla F_T(x_t, \epsilon_t)$, where ϵ_t is a random variable following a distribution P_t. The gradient in this model is calculated using a stochastic recursive estimator defined as:

$$d_t = \nabla F_t(x_t, \epsilon_t) + (1 - p_t)(d_{t-1} - \nabla F_t(X_{t-1}, \epsilon_t)) \tag{10}$$

Where d_1 equals the gradient of F when t=1. Once d_t is updated the algorithm finds a solution V_t to the linear optimization problem given by $argmin_{v \epsilon C}(d_t, v)$, and finally updates X_{t+1} in the direction $v_t - x_t$.

2.3.3 Adversarial setting

In the adversarial setting the recursive the algorithm MORGFW is inspired by the Meta-Frank Wolfe method, using the recursive estimator. This method relies on the outputs of base Online Linear optimization OLO. Each iteration the algorithm simulate a K-step Frank Wolfe subrouting calculating stochastic gradients of f:t and OLO algorithms. Successively the method take K Frank Wolfe type update steps where the update direction V_t is given by the base algorithm ϵ^k, later it receives the final iteration $x_t^{(K+1)}$ as the prediction in the t round, the loss and the stochastic gradient oracle.

2.3.4 Analysis of ORGFW

In order to obtain high probability results with the method ORGFW the following assumptions must be met.

1. The compact convex set C has diameter D meaning:

$$|x - y| < D \tag{11}$$

2. The stochastic gradient is unbiased and is L-Lipschitz continuous over the constraint set C

When met this assumption, the algorithm ORGFW achieves a nearly optimal $O(\sqrt{T})$regret bount for OCO problems.

2.3.5 Analysis of MORGFW

In order to obtain high probability results with the method MORGFW the following assumptions must be met.

1. The distance between the stochastic gradient and the exact gradient is bounded over the constraints set C wehre exists a $\sigma < \infty$, such taht

$$|\nabla F_t(x, \epsilon_t - \epsilon f_t(x)|^2 < \sigma \tag{12}$$

When the assumptions are met, the MORGFW is bounded from above by $\emptyset(\sqrt{T})$+R_T where R_T

2.4. Stochastic Frank Wolfe Methods for Nonconvex Optimization [5]

This paper analyzes in detail the case for Nonconvex optimization. As discussed in the other articles, Frank Wolfe algorithms have gained a lot of interest because of its projection-free characteristic. However there has not been much studies made regarding nonconvex scenarios. Therefore this paper presents a stochastic Frank Wolfe algorithm, and two variance reduced algorithms: SVFW and SagaFW. Finally they present their results which demonstrate that these variants attain faster convergence rates than the normal FW with a factor of $n^{1/3}$ and $n^{2/3}$ respectively.

Since the analysis is done for the nonconvex case, it is not possible to use the norm of the gradient to measure convergence. Thus the criterion used is the Frank-Wolfe gap:

$$G(x) = \max_{v \epsilon \Omega}(v - x, -\nabla F(x)) \tag{13}$$

$G(x) = 0$ for nonconvex functions if and only if x is a stationary point.

Furthermore, for the convergence analysis they used these oracles: Stochastic First-Order Oracle (SFO), Incremental First-Order Oracle (IFO) and Linear Optimization Oracle (LO), and then they refer to complexity for these oracles as the number of calls made by an algorithm.

2.4.1 SVFW

This algorithm is semi-stochastic since it computes the full gradient at the end of each epoch. This is done for controlling the variance of the stochastic gradients calculated later inside an inner loop. However the epoch size m is chosen in a way that the cost of calculating the full gradient is amortized. The key discovery for this algorithm is the following theorem:

Theorem 2.2.

$$\mathbb{E}[G(x_a)] <= \frac{2D}{\sqrt{T\beta}}\sqrt{L(F(x_0) - F(x*))}(1+\beta) \quad (14)$$

where x is an optimal solution of 13.*

Moreover the IFO and LO complexity for this algorithm are $O(n + n^{2/3}/\epsilon^2)$ and $O(1/\epsilon^2)$ respectively.

2.4.2 SagaFW

SagaFW avoids calculation of full gradients and instead updates the average gradient vector g_t at each iteration. The key results found for SafaFW is as follows:

Theorem 2.3.

$$\mathbb{E}[G(x_a)] <= \frac{2D}{\sqrt{T\beta}}\sqrt{L\theta(b,n,T)(F(x_0) - F(x*))}(1+\beta) \quad (15)$$

where x is an optimal solution of 13.*

Finally the IFO and LO complexity of this algorithm are $O(n + n^{1/3}/\epsilon^2)$ and $O(1/\epsilon^2)$ respectively.

3. Dataset

To conduct our experiment we utilized data sets which we did not see used in either of the related papers which we worked from. The objective was to test the 4 algorithms in an unbiased way and a new set of data was the best way to keep faithful to this objective. In the end we decided on two datasets coming from the UCI Machine Learning Repository [2]. These are the Breast Cancer Wisconsin and the Recognition of Handwritten Digits dataset. Both of these can be found on the UCI website: https://archive.ics.uci.edu/ml/datasets/.

To get the data ready for our model we pre processed it by combining data samples with targets labels for both the train and test set. These we then combined into DataLoader objects to better synchronize with the pytorch framework that we used.

Breast Cancer Wisconsin

- 569 data samples (80/20 train/test split)
- Binary Classification Problem

Recognition of Handwritten Digits

- 1797 data samples (80/20 train/test split)
- Multi Class Classification Problem (10 Classes)

4. Method

In order to support the described theory, we compare the performance of the previously mentioned algorithms, Stochastic Frank wolf (SFW), MSFW, ORGW and SVRF. Since, the complexity of computing gradients in terms of performing linear optimization and projecting, varies according to the implemented algorithm, we measured the actual running time of each optimizer analyzing the loss decay.

To do this, we use the PyTorch framework, due to its flexibility to declare custom optimizers. These are python classes that inherit the default Optimizer class from pytorch and allow to change the default implementation of the step() function. Thereby we implemented 4 custom optimizers, one for each algorithm. Furthermore, we implemented a Deep Neural Network of 4 layers, and an LMO oracle with the L1 ball constraint added.

Finally we ran the experiments for both datasets and specifying the hyperparameters as follows:

For the digits dataset:

- 1000 epochs for all optimizers.
- Batch size of 32 for all optimizers.
- L1 ball radius to 4096 for all optimizers.
- Momentum of 0.5 for MSFW.
- Subset of 77 samples for SVRF internal loop.

and for the breast cancer dataset:

- 120 epochs for all optimizers.
- Batch size of 2 for all optimizers.
- L1 ball radius to 4096 for all optimizers.
- Momentum of 0.5 for MSFW.
- Subset of 77 samples for SVRF internal loop.

5. Experiments

Figure1, reports the loss through 150 epochs, for the breast cancer dataset implementing the four alogrithms

5.1. Dataset: Breast cancer dataset

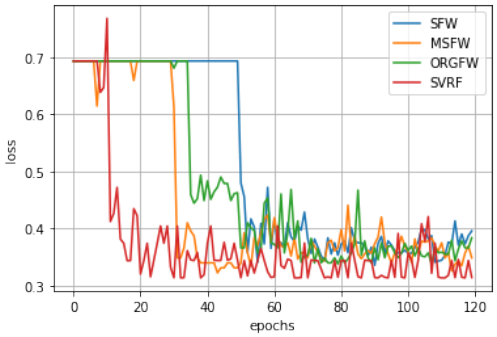

Figure 1. Training loss through epochs.

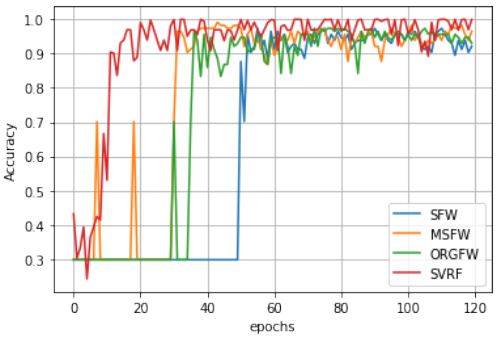

Figure 2. Training accuracy through epochs.

Optimizer	Time(s)
SFW	19.7
MSFW	21.2
ORGFW	24.0
SVRF	243.6

Table 1. Time comparison for 1000 epochs by algorithm for the breast cancer dataset

From 1 and 2 we can see that the variants of the classic SFW lead to better performance. We see for the first epochs that the SVRF and MSFW are the fastest to reduce the loss and increase the accuracy. However after more epochs the 4 algorithms reach very similar results. We see also in 1 that there is a considerable difference in execution times for SVRF when fixing the epoch number the same for all. This could be arguably different for the algorithm in question, since as seen in 2 SVRF reaches good results within much less epochs and does not change much after.

However it is worth noticing that, due to the small size of this dataset, these plots are very noisy. Therefore we perform the same analysis for the digits dataset which has much more samples.

5.2. Dataset: Digits recognition

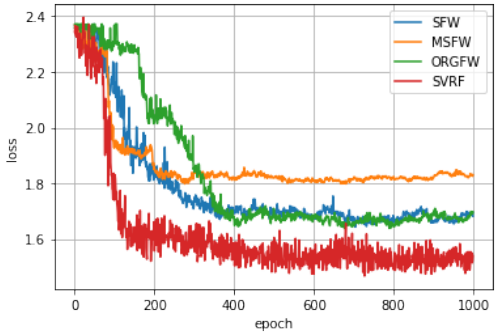

Figure 3. Training loss through epochs.

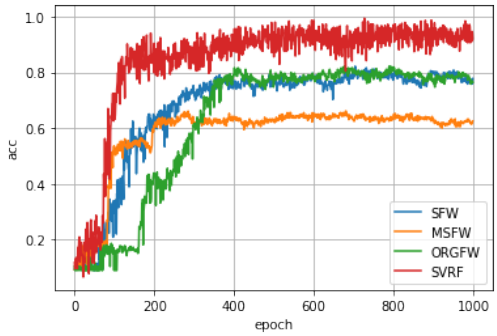

Figure 4. Training accuracy through epochs.

From 3 and 4 we see similar results as in the breast cancer dataset. However in this it is much clearer to see that again SVRF and MSFW have a better start than ORGFW and SFW. Nonetheless after more epochs MSFW seems to have reached the worst results overall, whereas SVRF reaches the best ones, leaving SFW and ORGWF in the middle with very similar results. Finally we compare their performance by measuring the time that these algorithms take to finish the 1000 epochs. As we see from 2, the first three algorithms have similar durations, but the SVRF takes almost ten times as long. This could be because SVRF performs another for loop for a small subset of the training set, for each epoch, to reduce the variance of the gradient. However as discussed for the previous dataset, it could be that SVRF does not require that many iterations as the others.

Optimizer	Time(s)
SFW	67.6
MSFW	72.5
ORGFW	79.0
SVRF	664.5

Table 2. Time comparison for 1000 epochs by algorithm

6. Conclusion

During the setting up and running of our experiments we can clearly see that the family of FW algorithms that we have tested are working very well for the problems presented in the data. Our custom optimizers are able to work well in the multi layer neural network we created. With relatively little parameter tuning we have reached very high accuracy scores for all four of our studied algorithms; especially in the case of the binary classification problem as can be seen in figure (2). This satisfies our initial curiosity with regards to the applicability of Frank Wolfe methods to neural network models.

We have additionally proven that the $\mathcal{L}^1$-ball constraint is feasible and applicable for neural network problems. As we have seen in table (2) the performance of the Stochastic and Momentum boosted Stochastic variants of our Frank Wolfe algorithms are particularly impressive. Though time consuming and incredibly costly the SVRF algorithm achieve the highest performance by far in terms of accuracy. This is something to be vary of for future experimentation; as improvements to the code could well reduce this computation cost to a manageable level.

By looking at both the accuracy reported in figure (2) and in figure (4) we can see that accuracy over time leaves a clear winner for convergence speed. We can see that the Momentum boosted SFW algorithm outstrips all the others in the initial epochs. Given enough time the other algorithms reach and outstrip it in later epochs of training.

The strength of our experiment lies in the reproducible and transferable nature of these Frank Wolfe optimization algorithms. With further tuning and experimentation we can match and perhaps exceed the unconstrained approach provided by Stochastic Gradient Descent.

7. Future Work

We have only been able to implement the basic Frank Wolfe algorithm and several of its variants as part of the study carried out to produce this paper. There are a number of studies of possible improvements to Frank Wolfe methods. An example is presented in the work carried out by Martin Jaggi et al [3]. Here the authors attempt to optimize the algorithms convergence for solutions that lie at the boundary which is a useful feature to have as it could well lead to linear convergence rates. Another set of interesting modifications and improvements possible for the family of Frank Wolfe algorithms are those that have been made by Lijun Ding et al in the following paper [1]. Here we observe modifications made to the algorithm to make it more suitable for large scale optimization. This is something which is critical during the rise of huge data sets and the necessity for efficient optimization methods. The improvements to the LMO and the computation of the gradient could be interesting to study in a practical setting. Finally one of the most recommended improvements for this work would be to implement the Frank Wolfe Gap 13, so as to not depend on a certain number of epochs. This could be specially important for the SVRF, since it has the higher computational duration per epoch, but it reaches better results with less epochs. We can summarize by saying that there many improvements to be made to the algorithm. Whether it is tailoring it for a specific problem domain or reducing its computational cost; we can firmly say that there are many interesting avenues for us to pursue with regards to utilizing Frank Wolfe for machine learning.

References

[1] Lijun Ding and Madeleine Udell. Frank-wolfe style algorithms for large scale optimization, 2018.

[2] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[3] Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of frank-wolfe optimization variants. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[4] Sebastian Pokutta, Christoph Spiegel, and Max Zimmer. Deep neural network training with frank-wolfe, 2020.

[5] Sashank J. Reddi, Suvrit Sra, Barnabas Poczos, and Alex Smola. Stochastic frank-wolfe methods for nonconvex optimization, 2016.