Frank and Wolfe vs Coordinate Descent for Deep Neural Networks

Lam Olivier Quôc-Vinh, Zghonda Jalel, Laraki Rayane
Section of Data Science, EPFL, Switzerland

***Abstract*—This paper focuses on a practical quantitative comparison between two optimizers in terms of training performance and accuracy. We provide a strict experimental protocol for testing the effectiveness of two recent implementations of deep Frank-Wolfe (DFW) and block coordinate descent (BCD) for DNNs on a standard digit recognition task. The results on the MNIST dataset show a very fast convergence for the BCD algorithm while the DFW yields a better accuracy.**

I. Introduction

In the past few years, deep learning has become more and more popular, thanks to its successful outcome in many real-world applications. As of today, stochastic gradient descent (SGD) is one of the most commonly used optimization algorithms due to the efficiency of the backpropagation method.

Even though SGD can yield great performances in various learning tasks, many researches are conducted in finding other optimizers that could have better performance and/or other properties than the SGD algorithm. In this paper, we focus on two recent optimizers, the block coordinate descend (BCD) and the deep Frank-Wolfe (DFW) algorithms.

The BCD algorithm is often used when an initial problem is too complex to find the optimal solution and it is possible to split it into simpler sub-problems, with respect to a subset (block) of variables [3][4][5][6]. In our case, the sub-problems are such that a closed-form solution exists and theses solutions are computed by direct minimization and by proximal algorithms. Therefore, the used BCD algorithm in this paper is a proximal algorithm and one of its properties is that this method is able to avoid any vanishing gradient issues.

The DFW algorithm (Berrada et al., 2019)[1], on the otheer hand, can compute the optimal learning rate for each step by using the Frank-Wolfe algorithm (Frank and Wolfe, 1956)[2].

As the literature already shows the convergence,the performance and some great properties of the two optimization algorithms, our goal is to compare their performance within the same learning task and understand which one works better in which type of task. For that purpose, we use the MNIST dataset in order to simulate an image recognition and classification task.

II. Methodology

We conduct experiments using a standard deep neural net model with 3 hidden fully connected layers on the MNIST dataset with 60K training and 10k test samples. The model takes a 28x28 image representing a handwritten digit and outputs a classification 1-hot vector of 10 classes. 3 variations of the model are used depending on the size of the dense layers denoted here D (D in [500,1000,1500]).

All hidden layers have Relu activation functions that help the reformulation of the classification problem in each layer with considering the activation as a projection on $\mathbb{R}_+^d$.

The proposed task to solve is a simple digit recognition problem where we study the impact the choice of the optimizer has on the performance and the effectiveness of the model.

For both optimizers the same experimental protocol is applied. Data normalization and randomization is used, 10 rounds training in 20 epochs. Optimal hyper-parameters values were taken from previous related work as the same task is performed using an equivalent model for BCD. No relevant hyperparameters are used for DFW since we don't use momentum and weight decay except for learning rate which has been manually tuned and set to 1e-1.

Google Colab is used for training the models with GPU enabled for both models, which help provide accurate training time comparison between the optimizers. All measurements are taken after averaging the accuracy and training results over the 10 rounds.

We provide in the results section insights on the accuracy obtained in both scenarios in order to analyze the generalization and the convergence of the algorithms. We use adapted implementations taken from Berrada et al.,2019)[1] for Deep Frank-Wolfe algorithm (DFW) and (Zeng et al. 2019)[5] for Block Coordinate descent (BCD) that we describe below for the convenience of the reader.

A. Deep Frank-Wolfe algorithm

In (Berrada et al.,2019)[1] the following DFW algorithm is introduced. The variables and function used in the algorithm are defined as follows:

- $\forall i, \mathbf{z}_i$: momentum velocity
- $\forall i, \mathbf{b}_t^{(i)}$: dual direction
- δ_t: derivative of (smoothed) loss function
- $\mathbf{r}_t$: derivative of regularization
- γ_t: step-size
- $\mathbf{w}_t$: parameters
- $\forall(\bar{y}, y) \in \mathcal{Y}^2, \Delta(\bar{y}, y) = \begin{cases} 0 & \text{if } \bar{y} = y, \\ 1 & \text{otherwise} \end{cases}$

In our implementation, we choose not to consider the momentum velocity variable. The variables are updated in a forward pass and the details of the algorithm are shown below:

Algorithm 1 The Deep Frank-Wolfe Algorithm

```
Require: proximal coefficient $\eta$, initial point $\mathbf{w}_0 \in \mathbb{R}^p$, momentum coefficient $\mu$, number of epochs.
Initialization: $t = 0,\ \mathbf{z}_0 = 0$

for each epoch do
    for each mini-batch $\mathcal{B}$ do
        Receive data of mini-batch $(\mathbf{x}_i, y_i)_{i \in \mathcal{B}}$
        $\mathbf{b}_t^{(i)}(\mathbf{w}_t) = (f_{\mathbf{x}_i,\bar{y}}(\mathbf{w}_t) - f_{\mathbf{x}_i,y_i}(\mathbf{w}_t) + \Delta(\bar{y}, y_i))_{\bar{y} \in \mathcal{Y}}, \forall i \in \mathcal{B}$
        $\mathbf{s}_t^{(i)} \leftarrow \texttt{get\_s}(\mathbf{b}_t^{(i)}(\mathbf{w}_t)), \forall i \in \mathcal{B}$
        $\delta_t = \partial \left( \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (\mathbf{s}_t^{(i)})^\top \mathbf{b}_t^{(i)}(\mathbf{w}_t) \right) \Big|_{\mathbf{w}_t}$
        $\mathbf{r}_t = \partial \rho(\mathbf{w}) \big|_{\mathbf{w}_t}$
        $\gamma_t = -\eta \delta_t^\top \mathbf{r}_t + \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (\mathbf{s}_t^{(i)})^\top \mathbf{b}_t^{(i)}(\mathbf{w}_t) \frac{\mathbf{w}_t}{\eta \|\delta_t\|^2}$
        $\mathbf{z}_{t+1} = \mu \mathbf{z}_t - \eta \gamma_t (\mathbf{r}_t + \delta_t)$
        $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta[\mathbf{r}_t + \gamma_t \delta_t] + \mu \mathbf{z}_{t+1}$
        $t = t + 1$
```

B. Block Coordinate Descent algorithm

Zeng & al. purpose in [5] the following BCD algorithm 2. The notation for the variables is the following:

- N : Number of layer in the neural network
- $\boldsymbol{X} \in \mathbb{R}^{d_0 \times n}$: the input vector
- $\boldsymbol{Y} \in \mathbb{R}^{d_N \times n}$: the output vector
- $\boldsymbol{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}$: the weight matrix between the $(i-1)$-th layer and the i-th layer, where $i = 1 \ldots N$ and $d_i \in \mathbb{N}$ is the number of units in the i-th layer.
- $\boldsymbol{V}_i = \sigma_i(\boldsymbol{W}_i \boldsymbol{V}_{i-1})$: the output vector at the i-th layer
- $\boldsymbol{U}_i = \boldsymbol{W}_i \boldsymbol{V}_{i-1}$: the input vector at the (i)-th layer.

From the analysis of Lau & al. [3], Xu and Yin [4] and Zhang and Brand [6], Zeng & al. [5] succeed to reformulate the initial empirical risk into a three-splitting formulation by introducing the auxiliary variables $\boldsymbol{U}_i$. They suggested the following minimization problem :

$$\min_{\mathcal{W},\mathcal{V},\mathcal{U}} \bar{\mathcal{L}}(\mathcal{W},\mathcal{V},\mathcal{U}) := \mathcal{L}_0(\mathcal{W},\mathcal{V}) + \frac{\gamma}{2} \sum_{i=1}^{N} \|\boldsymbol{V}_i - \sigma_i(\boldsymbol{U}_i)\|_F^2 + \|\boldsymbol{U}_i - \boldsymbol{W}_i \boldsymbol{V}_{i-1}\|_F^2$$

where γ is a hyperparameter and $\mathcal{L}_0$ is the sum of the empirical risk with the regularization terms of the blocks of variables $\{\boldsymbol{V}_i\}$ and $\{\boldsymbol{W}_i\}$.

Zeng & al. [5] proved that the above minimization problem can be splitted into three sub-problems, one consisting of the $\boldsymbol{W}_i$ variables, the second consisting of the $\boldsymbol{V}_i$ variables and the last one consisting of the newly introduced $\boldsymbol{U}_i$ variables. They suggested an backward BCD algorithm for the update of the variables, i.e. the variables of the final layer are updated first until the variables of the first layer. The variables $\{\boldsymbol{V}_i\}, \{\boldsymbol{W}_i\}, \{\boldsymbol{U}_i\}$ are alternately updated in each iteration and in order to update the variables, proximal algorithms are used except for the auxiliary variables $\{\boldsymbol{U}_i\}$, which are updated by direct minimization. The details of the algorithm can be found below.

Algorithm 2 Three-splitting BCD for DNN training

```
Samples: $\boldsymbol{X} \in \mathbb{R}^{d_0 \times n}, \boldsymbol{Y} \in \mathbb{R}^{d_N \times n}$
Initialization: $\{\boldsymbol{W}_i^0, \boldsymbol{V}_i^0, \boldsymbol{U}_i^0\}_{i=1}^N, \boldsymbol{V}_0^k \equiv \boldsymbol{V}_0 := \boldsymbol{X}$
Parameters: $\gamma > 0, \alpha > 0$

for $k = 1, \ldots$ do
    $\boldsymbol{V}_N^k = \mathrm{argmin}_{\boldsymbol{V}_N} \{ s_N(\boldsymbol{V}_N) + \mathcal{R}_n(\boldsymbol{V}_N; \boldsymbol{Y}) + \frac{\gamma}{2} \|\boldsymbol{V}_N - \boldsymbol{U}_N^{k-1}\|_F^2 + \frac{\alpha}{2} \|\boldsymbol{V}_N - \boldsymbol{V}_N^{k-1}{}_F^2\| \}$
    $\boldsymbol{U}_N^k = \mathrm{argmin}_{\boldsymbol{U}_N} \{ \frac{\gamma}{2} \|\boldsymbol{V}_N^k - \boldsymbol{U}_N\|_F^2 + \frac{\gamma}{2} \|\boldsymbol{U}_N - \boldsymbol{W}_N^{k-1} \boldsymbol{V}_{N-1}^{k-1}\|_F^2 \}$
    $\boldsymbol{W}_N^k = \mathrm{argmin}_{\boldsymbol{W}_N} \{ r_N(\boldsymbol{W}_N) + \frac{\alpha}{2} \|\boldsymbol{W}_N - \boldsymbol{W}_N^{k-1}\|_F^2 + \frac{\gamma}{2} \|\boldsymbol{U}_N^k - \boldsymbol{W}_N \boldsymbol{V}_{N-1}^{k-1}\|_F^2 \}$
    for $i = N-1, \ldots, 1$ do
        $\boldsymbol{V}_i^k = \mathrm{argmin}_{\boldsymbol{V}_i} \{ s_i(\boldsymbol{V}_i) + \frac{\gamma}{2} \|\boldsymbol{V}_i - \sigma_i(\boldsymbol{U}_i^{k-1})\|_F^2 + \frac{\gamma}{2} \|\boldsymbol{U}_{i+1}^k - \boldsymbol{W}_{i+1}^k \boldsymbol{V}_i\|_F^2 \}$
        $\boldsymbol{U}_i^k = \mathrm{argmin}_{\boldsymbol{U}_i} \{ \frac{\gamma}{2} \|\boldsymbol{V}_i^k - \sigma_i(\boldsymbol{U}_i)\|_F^2 + \frac{\gamma}{2} \|\boldsymbol{U}_i - \boldsymbol{W}_i^{k-1} \boldsymbol{V}_{i-1}^{k-1}\|_F^2 + \frac{\alpha}{2} \|\boldsymbol{U}_i - \boldsymbol{U}_i^{k-1}\|_F^2 \}$
        $\boldsymbol{W}_i^k = \mathrm{argmin}_{\boldsymbol{W}_i} \{ r_i(\boldsymbol{W}_i) + \frac{\gamma}{2} \|\boldsymbol{U}_i^k - \boldsymbol{W}_i \boldsymbol{V}_{i-1}^{k-1}\|_F^2 + \frac{\alpha}{2} \|\boldsymbol{W}_i - \boldsymbol{W}_i^{k-1}\|_F^2 \}$
```

III. Results

In the section we discuss the results obtained with the proposed experimental protocol.

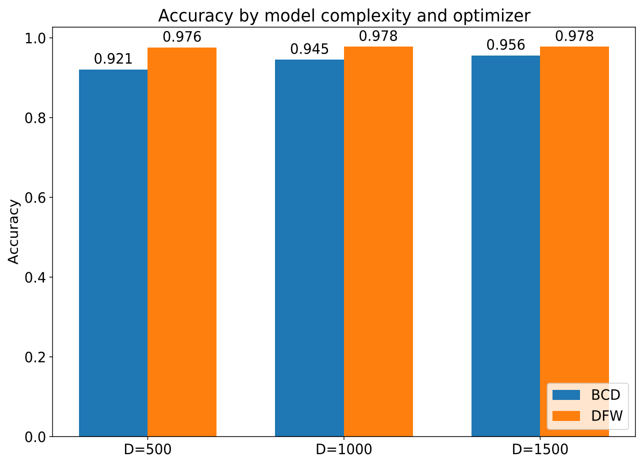

Fig. 1: Accuracy by model complexity and optimizer.

As shown in fig 1, in the best case scenario we reach an accuracy of 0.978 for DFW vs 0.956 for the model with hidden layers sizes of D=1500 units. We show as well that DFW is not sensitive to the model complexity and was able to perform the task no matter how much units we set per hidden layer. While BCD performed worse with a low number of units.

Overall in terms of accuracy DFW achieves significantly better results compared to BCD in this digit recognition problem.

For what concerns the training time presented in fig 2 without surprises BCD beats DFW in this comparison and trains substantially faster.

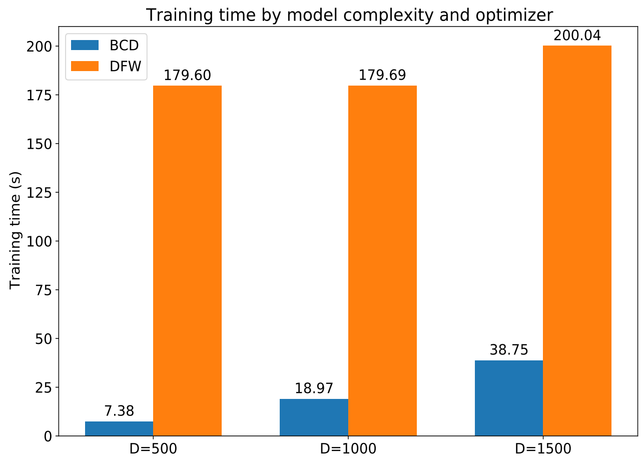

Fig. 2: Training time by model complexity and optimizer

Finally, a note on the convergence of the algorithms, we observe in fig 3 that DFW converges slowly and gradually towards the solution while BCD converges very quickly with achieving a solution in 2-3 epochs. The results are not surprising since the optimizer uses a gradient free solution.

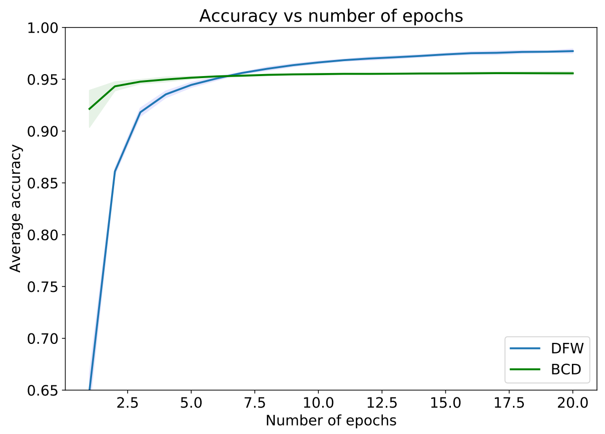

Fig. 3: Test accuracy evolution along the training epochs

IV. Conclusion

The insights on the training time were expected since we know that BCD has a cheaper iteration cost than DFW, even though we deactivated optional weight decay and momentum computation for the latter, we find that the algorithm has a substantial training cost. BCD with its gradient free solution provides as well a perfect convergence rate Nevertheless for achieving a better accuracy without the need of increasing the complexity of the model DFW is shown to be a good choice over BCD

References

[1] Leonard Berrada, Andrew Zisserman and M. Pawan Kumar. *Deep Frank-Wolfe for Neural Network Optimization*, 2019. https://arxiv.org/pdf/1811.07591.pdf

[2] Marguerite Frank, Philip Wolfe. *An algorithm for quadratic programming*, Naval Research Logistics Quarterly, 1956.

[3] Tim Tsz-Kit Lau, Jinshan Zeng, Baoyuan Wu, Yuan Yao. *A Proximal Block Coordinate Descent Algorithm for Deep Neural Network Training*, 2018. https://arxiv.org/pdf/1803.09082.pdf

[4] Yangyang Xu, Wotao Yin. A Block Coordinate Descend method for regularized multi-convex optimization with applications to non-negative tensor factorization and completion, SIAM Journal on Imaging Sciences, 2013. ftp://ftp.math.ucla.edu/pub/camreport/cam12-47.pdf

[5] Jinshan Zeng, Tim Tsz-Kit Lau, Shaobo Lin, Yuan Yao. *Global Convergence of Block Coordinate Descent in Deep Learning*, 2019. https://arxiv.org/pdf/1803.00225.pdf

[6] Ziming Zhang, Matthew Brand. *Convergent Block Coordinate Descent for Training Tikhonov Regularized Deep Neural Networks*, 2017. http://papers.neurips.cc/paper/6769-convergent-block-coordinate-descent-for-training-tikhonov-regularized-deep-neural-networks.pdf