Mini-Batch Proximal Block Coordinate Descent Method in Deep Learning

Haoming Lin, Junye Du, Zeyi Chen
École Polytechnique Fédérale de Lausanne (EPFL)
{haoming.lin, junye.du, zeyi.chen}@epfl.ch

***Abstract*—Deep learning continues to be the subject of widespread attention due to its remarkable empirical accomplishments, however, training DNNs is notably intricate due to the complexity associated with nonconvex optimization. Traditionally, the backpropagation algorithm has been utilized extensively as the cornerstone for first order gradient descent-oriented algorithms. Recent research has underscored the efficiency of some zero-order optimization methods in deep learning like block coordinate descent algorithm, which shows tremendous potential for training DNNs. In this project, we reimplement and simplify the BCD aglorithm on MNIST dataset and solve out the proximal iterations step under the condtion of ReLU activation, at the same time compare its performance with mainstream gradient algorithms.**

I. Introduction

Deep learning has gained immense popularity in recent years due to its remarkable ability to achieve state-of-the-art performance in extracting and learning from complex data. A major challenge for training deep neural networks (DNNs) springs from its essence of solving a high-dimensional nonconex optimization. Fundamentally, strategies employed for during the training of neural networks can be classified into three categories in terms of different orders being utilized in derivative, among which the first-order optimiztaion method like gradient descent (GD) is most commonly used and has made tremendous progress. Stochastic gradient descent (SGD), serving as the basic of first-order gradient methods, has engendered a multitude of variants like AdaGrad [1], RMSprop [2] and Adam [3]. Although these variants show adaptive performance when training deep neural networks, they may suffer from the inherent problem of vanishing gradient and saddle points; see, e.g., [4], [5]. In a gesture to solve this issue, approaches to approximate the objective function using function evaluations, namely zero-order methods, have been widely considered to skirt from explicit gradient information; e.g. alternating direction method of multipliers (ADMM) [6], the block coordinate descent (BCD) ([7], [8])

In this project, we focus on the proximal BCD method proposed in [9] from ICLR 2018. Apart from reimplementing the proximal BCD algorithm, we generalize the alrgorithm into a minibatch version such that we could have a more detailed comparison with current popular optimizers in terms of the convergence rate, accuracies and graphics memory usage in cuda.

II. BCD algorithm in deep learning

A. Notations and Problem Formulation

The following terminologies are adopted across the rest of this paper. We consider an N-layer feedforward neural network with $N-1$ hidden layers, with d_i denoting the number of hidden units in the ith hidden layer. Let $W_i \in \mathbb{R}^{d_{i-1}\times d_i}$ and $b_i \in \mathbb{R}^{d_i \times 1}$ denote the weight matrix and the bias vector between the $(i-1)$th and ith layer. The activation function after the ith layer is denoted by $\rho_i(\cdot)$. The collected data is $X = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^{d_0 \times n}$ and the labels are $Y = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^{d_N \times n}$. For the sake of representation, we further introduce two auxiliary variables denoting the output before and after activation at each layer, namely,

$$U_i = W_i^T V_{i-1} + b_i \in \mathbb{R}^{d_i \times 1} \; i = 1, 2, \ldots, N$$
$$V_i = \begin{cases} X & i = 0 \\ \sigma_i(U_i) & i = 1, 2, \ldots, N \end{cases} \tag{1}$$

We also denote $(V_i)_j$ to be the jth column of V_i. Finally, we define $\mathbf{W} := \{W_i\}_{i=1}^N$, $\mathbf{B} := \{b_i\}_{i=1}^N$ to be the collections of weight matrices and bias vectors, and $\mathbf{U} := \{U_i\}_{i=1}^N$, $\mathbf{V} := \{V_i\}_{i=0}^N$ to be the collections of the two auxiliary variables, respectively.

A multi-layer perceptron (MLP) consists of multiple layers with fully-connected neurons. By representing the empirical total risk as a sum of individual losses, the minimization problem to be solved in each iteration can then be formulated as

$$\min_{\mathbf{W},\mathbf{B}} \boldsymbol{\mathcal{R}}_n(\Phi(X; \mathbf{W}, \mathbf{B}), Y) := \frac{1}{n}\sum_{i=1}^{n} l[\Phi(x_i; W_i, b_i), y_i] \tag{2}$$

where $l(\cdot)$ an arbitrary loss function and was chosen to be the MSE-loss as our criterion. Meanwhile, Φ denotes the output of the associated MLP network which is with a composite form and can be simplified using (1) as

$$\Phi(x_i; W_i, b_i) := (V_N)_i. \tag{3}$$

This would give convenience for some relaxation of Problem 2 as to be shown later.

B. An Overview of the Implementation

The high dimension and the highly non-convexity natures often make (2) intractable from a computational perspective. When implementing the algorithm, we applied the variable splitting techniques (see, e.g., [10], [11], [12], etc.) which essentially relaxes Problem 2 and adapts to diverse machine learning applications. In a nutshell, addtional terms are introduced in the objective function to reduce the couplings between variables embedded in the nonlinear functions thus alleviating the complexity. In this project, we follow the 3-variable-splitting formulation in [12]. Specifically, we focus on the application of the 3-variable-splitting based proximal BCD on image classification, particularly on the MNIST dataset introduced in [13]. The updates in our algorithm were conducted in a ”backward” fashion in analogue to [12] and [11]. The activation function was chosen to be ReLU for its interpretability as a projection onto the closed upper half-space thus simplifying the procedure of explicitly solving the objective. This instead could have been difficult with other candidates such as tanh or sigmoid functions [9].

The details on the algorithms and the relaxed formulation of (2) are provided in Appendix for readers' references.

III. Experiments and comparisons

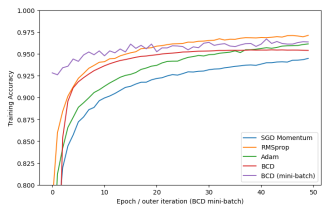
From the MNIST database, 60,000 images were selected for training and 10,000 for testing.

A. Comparison Between the Proximal BCD and Other Optimization Schemes

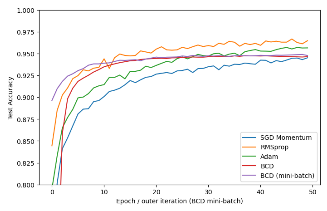
We examined the associated proximal BCD algorithms both in full-batch and mini-barch versions. Other candidates include the SGD with momentum, RMSProp and Adam, of which the learning rates are 0.1, 10^{-3} and 3×10^{-4} respectively. Four aspects were considered in our comparison of different optimization methods/schemes:

- Training and test accuracy.
- Convergence rate per iteration
- The duration time for training (measured in seconds).
- Memory occupied for CPU/GPUs (measured in MB).

The results from our expriments are summarized in Table I and the evolutions of training and testing accuracies per epoch are also displayed in Figure 1. Note that the mini-batch BCD was implemented using Algorithm 2 with batch size 4096.



(a) Training acc per iteration



(b) Test acc per iteration

Fig. 1: Evolutions of training & testing accuracies for candidate methods

According to Table I and Figure 1, it was discerned that the proximal BCD exhibited a significantly rapid convergence compared to other candidates during the initial epochs. However, it was observed to yield slightly inferior results concerning both training and test accuracy at the final stages, a characteristic synonymous with the inherent traits of BCD. As for the the minibatch BCD algorithm, it parallels the original BCD in terms of the rapid convergence rate yet demonstrates a propensity towards overfitting, which is indicative of the need for further refinements. Moreover, proximal BCD method harbors an inherent drawback: it necessitates a disproportionately substantial allocation of GPU memory, demanding approximately two hundred times the memory that the Stochastic Gradient Descent (SGD) method would require, even under circumstances where the minibatch size is minimized.

B. Influence of batchsize in mini-batch proximal BCD

In the course of empirical investigation, the hyperparameter *batchsize* emerged as a decisive factor in the training process of the minibatch proximal BCD model. Under conditions of a relatively minimal *batchsize*, the inner loop exhibited a remarkable convergence speed, resulting in overfitting within a scant approximated 10 iterations. Applying the foundational algorithm directly in a minibatch context resulted in a considerable decline in test accuracy, thereby presenting a significant accuracy disparity when compared to the training set. As a remedial strategy, we cyclically implemented the forward and backward functions, facilitating a more efficacious learning of the parameters by the model.

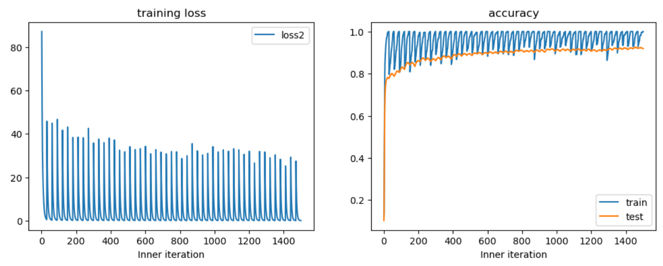


Fig. 2: Overfitting issue under low batchsize of inner loop

As is shown in Figure 2 and in Table II, even in scenarios where the minibatch size is relatively small, we are still able to control overfitting by alternating the updates between the backward and forward functions.

C. The Balance of Inner and Outer Iterations

This part serves to demonstrate the importance of selecting appropriate numbers of inner and outer iterations. In our setting, we controlled the product of the batch size, the number of inner iterations and the of outer iteratoins to be homogeneous. The final test accuracy and the approximate time to convergence were monitored. Empirical evidence showed that, as a trade-off between the learning accuracy and the efficiency, the number of inner iterations should be neither too large nor too small. Figure 3 gives the results when we employed a batch size of 128, and implies that frequent inner iterations would result in some overfitting problem since the model performs relatively poor though being extremely accurate on the training samples. Yet, an extremely small

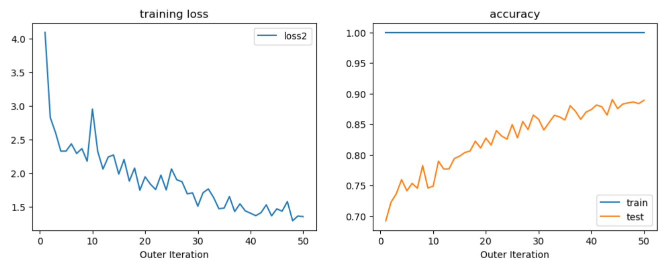
TABLE I: A comparison of Different Models' performances: Memory usage in MB, CPU indicates memeory after - before training; Training time does not include time consumed for validation during training.

Model	CPU	GPU Allocated	GPU Cached	Training Time	Final Test Acc	Iters to 0.94 Acc
BCD full batch	2386.0 - 562.4	2090.3	5128.0	∼76 s	0.946	15 (9.0e5 data)
BCD b4096	2553.2 - 562.5	166.0	794.0	∼117 s	0.948	13x12 (6.4e5 data)
SGD momentum 0.9	2255.6 - 537.9	0.7	2.0	∼58 s	0.945	41 (2.4e5 data)
RMSProp	2258.7 - 539.5	0.7	2.0	∼61 s	0.965	11 (1.1e5 data)
Adam	2257.2 - 539.5	0.9	2.0	∼69 s	0.957	23 (2.3e5 data)

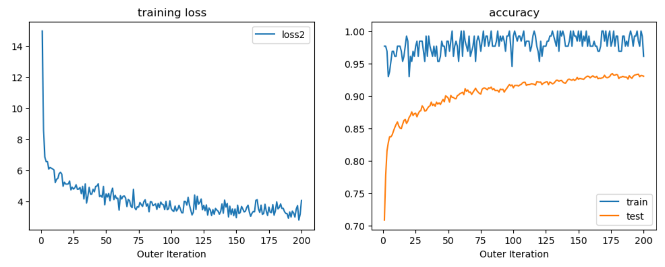
batch size	niter inner	niter outer	test accuracy
128	50	60	0.889
128	100	30	0.915
128	200	15	**0.933**
128	300	10	0.931
1024	5	75	0.889
1024	25	15	**0.934**
1024	75	5	0.919

TABLE II: Effect of inner and outer iteration numbers

number of iterations would also be disadvantageous since it procrastinates the learning procedure, which can be witnessed in the experiment summarized in Figure 4 and in Table II with the batch size set to be 1024.



(a) Training loss and test accuracy against the number of outer iterations (batch size = 128, niter-outer = 50, niter-inner = 60)

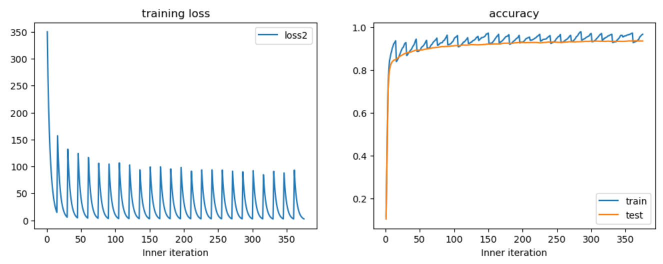


(b) Training loss and test accuracy against the number of outer iterations (batch size = 128, niter-outer = 200, niter-inner = 15)

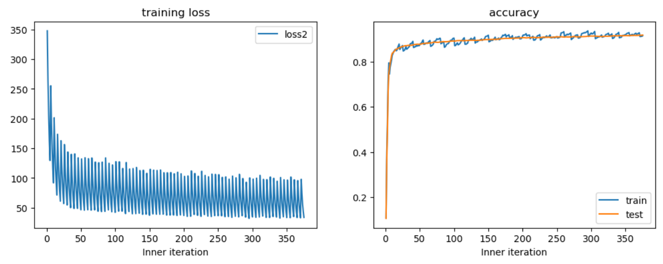
Fig. 3: Training loss/ Test accuracy V.S. the number of iterations: large nitr-inner causes overfitting

IV. Convergence rate and limitation

According to the literature works [12], there exists a global convergence to some critical point at a rate of $O(1/k)$ by incorporating several innovative components into the Kurdyka-Łojasiewicz inequality framework, thereby facilitating the execution of a comprehensive convergence analysis of BCD within the broad spectrum of deep learning contexts. However, the potential of this method is somewhat limited. Despite its prevails over other methods during the first a few epoches, it offers marginal improvements during later iterations and does not perform comparably well as SGD, Adam, and the like in the long term. Additionally, our discussion has been restricted to the ReLU activation function, and as mentioned in Section II, it could be challenging to derive explicit expressions for other activation functions which are left to be a future work. Consequently, in typical neural network training, SGD and its variants prove more effective and consume less space.



(a) Training loss and test accuracy against the number of inner iterations (batch size = 1024, niter-outer = 25, niter-inner = 15)



(b) Training loss and test accuracy against the number of inner iterations (batch size = 1024, niter-outer = 75, niter-inner = 5)

Fig. 4: Training loss/ Test accuracy V.S. the number of inner iterations: small nitr-inner causes slow learn

V. Summary

In conclusion, we reimplement the proximal BCD algorithm, fathom out the explicit expression of the iteration under ReLU activation function and generalize it into mini-batch version consisting of inner and outer loop. Through iterative testing and balancing adjustments of hyperparameters such as batchsize and the number of inner/outer iterations, we sought to examine their influence on the training outcomes. Moreover, a detailed comparison between proximal BCD and SGD-like algorithms was carried out to provide a comprehensive understanding of their relative efficacy.

REFERENCES

[1] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization.” *Journal of machine learning research*, vol. 12, no. 7, 2011.

[2] G. Hinton, N. Srivastava, and K. Swersky, “Neural networks for machine learning lecture 6a overview of mini-batch gradient descent,” *Cited on*, vol. 14, no. 8, p. 2, 2012.

[3] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[4] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[5] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization,” *Advances in neural information processing systems*, vol. 27, 2014.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[7] P. Tseng, “Convergence of a block coordinate descent method for nondifferentiable minimization,” *Journal of optimization theory and applications*, vol. 109, no. 3, p. 475, 2001.

[8] A. Beck and L. Tetruashvili, “On the convergence of block coordinate descent type methods,” *SIAM journal on Optimization*, vol. 23, no. 4, pp. 2037–2060, 2013.

[9] T. T.-K. Lau, J. Zeng, B. Wu, and Y. Yao, “A proximal block coordinate descent algorithm for deep neural network training,” *arXiv preprint arXiv:1803.09082*, 2018.

[10] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein, “Training neural networks without gradients: A scalable admm approach,” in *International conference on machine learning*. PMLR, 2016, pp. 2722–2731.

[11] Z. Zhang and M. Brand, “Convergent block coordinate descent for training tikhonov regularized deep neural networks,” *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[12] J. Zeng, T. T.-K. Lau, S. Lin, and Y. Yao, “Global convergence of block coordinate descent in deep learning,” in *International conference on machine learning*. PMLR, 2019, pp. 7313–7323.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

VI. APPENDIX

A. Relaxation of Problem (2) *and the Proximal BCD Algorithm*

Algorithm 1: Three-Splitting based Proximal BCD for MNIST under ReLU activation

```
Data: The data $X \in \mathbb{R}^{d_0 \times n}$ and the labels
      $Y \in \mathbb{R}^{d_N \times n}$ from MNIST; the parameters
      $\gamma \in \mathbb{R}_+$, $\alpha \in \mathbb{R}_+$, $\rho \in \mathbb{R}_+$; the activation
      functions $\sigma_i(\cdot)$; the number of iterations $T$;
      (optional) the weight $\mathbf{W}$ and the bias $\mathbf{B}$.
Result: $\mathbf{W}$, $\mathbf{B}$, $\mathbf{U}$, $\mathbf{V}$
Initialize (if not provided) $\mathbf{W} := \{W_i\}_{i=1}^N$ and
  $\mathbf{B} := \{b_i\}_{i=1}^N$;
Compute $\mathbf{U} := \{U_i\}_{i=1}^N$ and $\mathbf{V} := \{V_i\}_{i=0}^N$ according
  to (1);
for $k \leftarrow 1$ to $T$ do
    $V_N^k \leftarrow (Y + \gamma U_N + \alpha V_N)/(1 + \alpha + \gamma)$;
    $U_N^k \leftarrow (\gamma V_N + \rho W_N^T V_{N-1} + b_N)/(\gamma + \rho)$;
    $W_N^k \leftarrow$
      $(\alpha W_N + \rho(V_N - b_N)V_N^T)(\gamma V_N + \rho(W_N^T(V_N - b_N))$;
    $b_N^k \leftarrow (\alpha b_N + \rho(V_N - W_N^T V_{N-1})\mathbf{1}^T)/(\rho N + \alpha)$;
    for $i = N-1, N-2, \ldots, 1$ do
        $V_i^k \leftarrow$
          $(\rho W_i^T W_i + \gamma I_d)^{-1})(\rho W^T(U_i - b_i) + \gamma U_{i-1})$;
        $U_i^k \leftarrow \arg\min_{U_i}\{\frac{\gamma}{2}||V_i^k - \sigma_i(U_i)||_F^2 + \frac{\gamma}{2}||U_i -$
          $(W_i^{k-1})^T V_{i-1}^{k-1}||_F^2 + \frac{\alpha}{2}||U_i - U_i^k||_F^2\}$;
        $W_i^k \leftarrow$
          $(\alpha W_i + \rho(V_i - b_i)V_i^T)(\gamma V_i + \rho(W_i^T(V_i - b_i))$;
        $b_i^k \leftarrow (\alpha b_i + \rho(V_i - W_i^T V_{i-1})\mathbf{1}^T)/(\rho N + \alpha)$;
    end
end
```

To reduce the couplings between variables embedded in the nonlinear functions, Problem 2 was firstly extended to

$$\min_{\mathbf{W},\mathbf{B},\mathbf{U},\mathbf{V}} \mathcal{L}_0(\mathbf{W},\mathbf{B},\mathbf{V}) := \frac{1}{n}\sum_{i=1}^{n} l[(V_N)_i, y_i] + \sum_{i=1}^{N}[r_i(W_i) + s_i(V_i)] \tag{4}$$

subject to *Costraint 1 is satisfied.*

where r_i and s_i are some extended-real-valued nonnegative functions, depending on the problem scenario. Such a generalization contains high flexibility and enjoys a broad range of applications in machine learning; for instance, when considering the regularized DNN models, an assumption of convexity can be imposed on r_i and s_i (e.g. l_1-norm). Problem 4 can then be further relaxed to an unconstrained program by invoking some Lagrange multipliers (which are set to be identical for all variables as suggested in literature; see [12])

$$\min_{\mathbf{W},\mathbf{B},\mathbf{U},\mathbf{V}} \tilde{\mathcal{L}}(\mathbf{W},\mathbf{B},\mathbf{U},\mathbf{V}) := \mathcal{L}_0(\mathbf{W},\mathbf{B},\mathbf{V}) + \frac{\gamma}{2}\sum_{i=1}^{N}[||V_i - \rho_i(U_i)||_F^2 + ||U_i - W_i V_{i-1}||_F^2] \tag{5}$$

where $||\cdot||_F$ is the Frobenius norm. The procedure of the employed 3-variable-splitting based proximal BCD method is indicated in Algorithm 1 in details.

Algorithm 1 preassumingly exploits the full-batch information for the optimization in each iteration. In fact, one can easily modify it to a mini-batch algorithm by feeding it with only sampled data and embed it in some outer iteration. The modified mini-batch algorithm, which is established on Algorithm 1, is presented in Algorithm 2. The modified algorithm was used along in our mini-batch expriments.

Algorithm 2: Three-Splitting based Proximal BCD for MNIST under ReLU activation **(Mini-batch Version)**

```
Data: The data $X_0 \in \mathbb{R}^{d_0 \times n}$ and the labels
      $Y_0 \in \mathbb{R}^{d_N \times n}$ from MNIST; the batchsize $m$; the
      parameters $\gamma \in \mathbb{R}_+$, $\alpha \in \mathbb{R}_+$, $\rho \in \mathbb{R}_+$; the
      activation functions $\sigma_i(\cdot)$; the number of outer
      iterations $T_{\text{out}}$ and the number of inner iterations
      $T_{\text{in}}$.
Result: $\mathbf{W}$, $\mathbf{B}$
Initialize $\mathbf{W} := \{W_i\}_{i=1}^N$ and $\mathbf{B} := \{b_i\}_{i=1}^N$;
for $k \leftarrow 1$ to $T_{out}$ do
    $X, Y \leftarrow \text{sample}((X_0, Y_0), m)$;
    $\mathbf{W}, \mathbf{B} \leftarrow \text{Algorithm } 1(X, Y, T_{\text{in}}, \mathbf{W}, \mathbf{B})$;
end
```