Convergent Block Coordinate Descent for Training Tikhonov Regularized Deep Neural Networks

Implementation Track
4 Team Members

Abstract

Deep learning has become very popular due to its wide success in solving complex problems. However Deep Neural Networks (DNN's) tend to be highly nonconvex, so finding the global optimum is difficult. Using gradient-based methods such as stochastic gradient descent (SGD) to solve DNN's is prone to various problems, one of which is vanishing gradients, where the many layers of the DNN drives the gradients of the loss function to zero. State-of-the-art approaches to tackle this problem includes using non-gradient-based based algorithms, as well as using regularization functions. This paper is a re-implementation of an example to this approach; Zhang et.al's[16] "Convergent Block Coordinate Descent (BCD) for Training Tikhonov Regularized Deep Neural Networks". We present related works, methods for training the DNN, experiments and results with the MNIST and MNIST Fashion datasets, and the evaluation of our re-implemented algorithm against another BCD algorithm; Proximal BCD[9] as well as the SGD with Dropout regularization. Despite being bounded by computational limitations, our results support the superiority of the BCD algorithm and the Tikhonov regularization through faster convergence, and higher test accuracy.

1 Introduction

In this project we explore Zhang et.al [16] work on "Convergent Block Coordinate Descent (BCD) for Training Tikhonov Regularized Deep Neural Networks (DNNs)". This work tries to address the problem of nonconvexity of the DNN objective function, specifically the "vanishing gradient problem." It offers a solution through an updated, higher dimensional rectified linear unit (ReLU) activation function with Tikhonov regularization, and the adoption of a novel BCD as opposed to stochastic gradient descent (SGD) as the learning method. As an extension to our principal paper we test this algorithm on different datasets such as Fashion MNIST. We also expand our evaluation with an investigation into a newer upgrade to our principal paper, Proximal BCD. Moreover we compare results with Dropout regularized DNNs trained using SGD and alternating direction method of multipliers (ADMM) optimizer.

Our goal through this project is to deliver our search on related work around the problem (Section 2), the method to reimplement the presented methods (Section 3), the design of the experiments (Section 4), and the evaluation of experimental results (Section 5).

2 Related Work

One way to prevent over fitting problem in DNNs is to come up with a regularization that has a penalizing effect as studied in EECS 545. To this end, there are several papers that propose novel regularization methods. These can be broadly divided into three categories.

- Stochastic regularization(SR): The basic idea behind this approach is to add random variables to the network, so that the model obtained is robust. Dropout [13], is a popular SR method that randomly drops nodes of the DNN during training. Some prominent variations of the dropout techniques are *Dropconnect* [10] and *Fast dropout* [14]. *Dropconnect* is a technique which instead of randomly setting a subset of neurons to zero, sets a randomly selected subset of weights and biases to zero. This method was reported to have better regularization on some networks compared to the standard dropout method. *Fast dropout* is a technique that approximates the input to a neuron by a Gaussian distribution, which decreases number of computations, and yields faster convergence.
- Local regularization (LR): The idea behind local regularization is to define a local entropy based objective function that favours generalizable solutions in the flat regions of energy [5]. In a way this objective tries to avoid sharp minima that overfit the data.
- Tikhonov regularization(TR): TR smoothens the non-convex objective explictly (a closed form expression) and globally unlike SR and LR which do it implicitly. It is also to be noted the TR term in the objective is data dependent.

Other methods than BCD exist for optimizing the weights in DNNs. Two alternate potential methods are explored and evaluated for advantages and disadvantages in this application.

- Stochastic gradient descent (SGD): Stochastic gradient descent is the most popular method for training deep neural networks. Basically, this method involves iteratively updating the weights in the model according to the gradient with respect to randomly selected samples of the data. Bounds for weak convergence have been proven under certain conditions [7], but BCD has been proven to converge globally to stationary points in nonconvex optimization and is therefore advantageous for the application in this paper.
- Alternating direction method of multipliers (ADMM): ADMM is an algorithm which is well-suited to distributed convex optimization. It breaks a large problem into many local parts to which the solutions can be found more easily. These solutions can then be used to find the solution to the larger problem. It has been proven to have good convergence in some certain cases[11] [15], but no such proof exists for the application of training DNNs.

Among these methods, BCD and ADMM are superior to SGD for tackling the vanishing gradient problem, since they are gradient-free methods. Zeng et. al.[8] surveys and categorizes current BCD algorithms, and proposes a generic method to asses global convergence of a given BCD algorithm for DNNs. They categorize BCD algorithms as either *two splitting formulation* or *three splitting formulation*, based on the decomposition approach of the objective function. Our implemented paper[16] is in the first category, along with another BCD approach by Perpinan et. al[4], while Taylor et.al[6]. and Lau et. al.[9] adopt BCD approach of the second category. They propose a Kurdyka-Lojasiewicz (KL) inequality based framework[2], with a set of assumptions involving Lipschitz contiuity, to find a global guaranteed convergence to a critical point with O(1/k) general time complexity, for k number of iterations. They reported their analysis to work for a broad range of activation, loss and regularization functions.

In terms of recent advancement in this topic, lau et al. [9] proposed a proximal block coordinate descent for training DNNs. Authors defined a unique loss function using the quadratic penalty method [12]. So as to have a loss function as block convex, authors further simplified the loss function by projecting activation functions(ReLU) onto nonempty closed convex sets[2]. To further expand our testing, we have compared there method against our principal algorithm. To keep problem statement consistent across both algorithms we have used the same three layer DNN without any bias.

3 Method

The following subsections provide a summary of the derivation of Zhang et.al [16] method for training DNNs, as well as our specialization of the technique for the chosen loss function.

3.1 Tikhonov Regularization for DNNs

Table 1: Key Notations

Name	Description
$\mathbf{x}_i$	i-th training data $\in \mathbb{R}^{d_0}$
y_i	Class label for i-th training data $\in \mathcal{Y}$
$\mathcal{Y}$	Label set
$\mathbf{u}_{i,n}$	Output feature for $\mathbf{x}_i$ from the n-th $(1 \le n \le N)$ hidden layer $\in \mathbb{R}^{d_n}$
$\mathbf{W}_{n,m}$	Weight matrix between the n-th and m-th hidden layers $\in \mathbb{R}^{d_n \times d_m}$
$\mathcal{M}_n$	Input layer index set for the n-th hidden layer
$\mathbf{V}$	Weight matrix between the last hidden layer and the output layer $\in \mathbb{R}^{d_{N+1} \times d_N}$
$\ell(\cdot,\cdot)$	Convex loss function

3.1.1 Problem Setup

Modern DNNs generally employ ReLU as the activation functions, so in this work we will only be considering DNNs with the ReLU Activation function. In order to provide short paths through DNN, multi-input ReLU units (i.e. ReLU can take input from multiple previous layers as illustrated in Figure 1) are used.

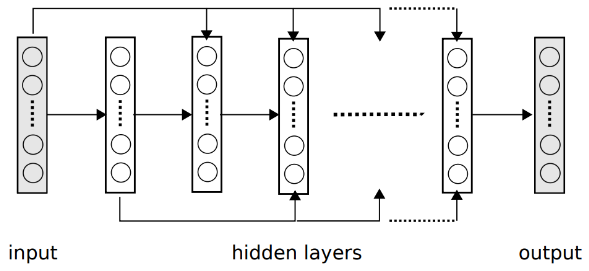

Figure 1: DNN Architecture

Mathematically, multi-input ReLU can be defined as:

$$\mathbf{u}_{i,n} = \begin{cases} \mathbf{x}_i, & \text{if } n = 0 \\ \max\left\{\mathbf{0}, \sum_{m \in \mathcal{M}_n} \mathbf{W}_{n,m}\mathbf{u}_{i,m}\right\}, & \text{otherwise} \end{cases} \tag{1}$$

Conventional Objective Function for Training DNNs with ReLU General objective function is as follows:

$$\min_{\mathbf{V}\in\mathcal{V}, \tilde{\mathcal{W}}\subseteq\mathcal{W}} \sum_i \ell\left(y_i, \mathbf{V}\mathbf{u}_{i,N}\right), \text{ s.t. } \mathbf{u}_{i,n} = \max\left\{\mathbf{0}, \sum_{m\in\mathcal{M}_n} \mathbf{W}_{n,m}\mathbf{u}_{i,m}\right\}, \mathbf{u}_{i,0} = \mathbf{x}_i, \forall i, \forall n \tag{2}$$

Here, $\tilde{\mathcal{W}} = \{\mathbf{W}_{n,m}\}$. The network architecture used in this work is mainly for extracting features and on top of which a classifier could be learned. To optimize this objective function a novel BCD based algorithm introduced in this paper using Tikhonov regularization is employed.

3.1.2 ReLU Reinterpretation

ReLU can be considered as projection on a convex set(POCS) [3] and can be rewritten as follows:

$$\max\{\mathbf{0}, \mathbf{x}\} \equiv \arg\min_{\mathbf{u}\in\mathcal{U}} \|\mathbf{u} - \mathbf{x}\|_2^2 \tag{3}$$

This reinterpretation of ReLU will help us form the basis of our lifted objective function.

3.1.3 Modified Tikhonov Objective Function

Using Equation 3, we will unroll our general objection function in Equation 2:

$$\min_{\tilde{\mathcal{U}}\subseteq\mathcal{U},\mathbf{V}\in\mathcal{V},\tilde{\mathcal{W}}\subseteq\mathcal{W}} f(\tilde{\mathcal{U}},\mathbf{V},\tilde{\mathcal{W}}) \sum_i \ell\left(y_i, \mathbf{V}\mathbf{u}_{i,N}\right) + \sum_{i,n} \frac{\gamma_n}{2} \left\| \mathbf{u}_{i,n} - \sum_{m\in\mathcal{M}_n} \mathbf{W}_{n,m}\mathbf{u}_{i,m} \right\|_2^2 \quad (4)$$
$$\text{s.t.} \quad \mathbf{u}_{i,n} \geq \mathbf{0}, \mathbf{u}_{i,0} = \mathbf{x}_i, \forall i, \forall n \geq 1$$

here $\tilde{\mathcal{U}} = \{\mathbf{u}_{i,n}\}$ and $\gamma_n \geq 0, \forall n$ denote our regularization constants. Larger γ_n values force $\mathbf{u}_{i,n}, \forall i$ to more closely approximate the output of ReLU at the n-th hidden layer. Further simplifying the above equation and rearranging the $\mathbf{u}$ and γ terms into a matrix $\mathbf{Q}$, the modified Tikhonov objective function is formed.

$$\min_{\tilde{\mathcal{U}}\subseteq\mathcal{U},\mathbf{V}\in\mathcal{V},\tilde{\mathcal{W}}\subseteq\mathcal{W}} f(\tilde{\mathcal{U}},\mathbf{V},\tilde{\mathcal{W}}) \equiv \sum_i \left\{ \ell\left(y_i, \mathbf{V}\mathbf{P}\mathbf{u}_i\right) + \frac{1}{2}\mathbf{u}_i^T \mathbf{Q}(\tilde{\mathcal{W}})\mathbf{u}_i \right\} \quad (5)$$

3.2 Block Coordinate Descent

To guarantee convergence, a novel BCD algorithm is used. In this algorithm, the objective function in Equation 4 is divided into three convex sub-problems. The BCD algorithm solves these convex subproblems sequentially, also with an additional quadratic term. These two features together ensure global convergence. The subproblems are presented in Equations 6-8. Equation 6 is the Tikhonov regularized inverse problem. Equation 7 is a classification problem, and Equation 8 is a least square regression problem.

$$\min_{\mathbf{u_i}\in\mathcal{U}} \ell\left(y_i, \mathbf{V}\mathbf{P}\mathbf{u}_i\right) + \frac{1}{2}\mathbf{u}_i^T \mathbf{Q}(\tilde{\mathcal{W}})\mathbf{u}_i, \forall i \quad (6)$$

$$\min_{\mathbf{V}\in\mathcal{V}} \ell\left(y_i, \mathbf{V}\mathbf{P}\mathbf{u}_i\right) \quad (7)$$

$$\min_{\forall\mathbf{W_{n,m}}\in\tilde{\mathcal{W}}} \sum_i \frac{\gamma_n}{2} \left\| \mathbf{u}_{i,n} - \sum_{m\in\mathcal{M}_n} \mathbf{W}_{n,m}\mathbf{u}_{i,m} \right\|_2^2 \quad (8)$$

Algorithm 1 demonstrates the novel BCD algorithm using these subproblems, for training the DNN. The input is the training data $(\mathbf{x}_i, \mathbf{y}_i)$ and regularization parameters γ_n, and the output is the weights $\tilde{W}$.

Algorithm 1 Block Coordinate Descent (BCD) Algorithm for Training DNNs

```
Input  : training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}$ and regularization parameters $\{\gamma_n\}$
Output : network weights $\tilde{\mathcal{W}}$
Randomly initialize $\tilde{\mathcal{U}}^{(0)} \subseteq \mathcal{U}, \mathbf{V}^{(0)} \in \mathcal{V}, \tilde{\mathcal{W}}^{(0)} \subseteq \mathcal{W}$;
Set sequence $\{\theta_t\}_{t=1}^{\infty}$ so that $0 \leq \theta_t \leq 1, \forall t$ and sequence $\left\{\sum_{k=t}^{\infty} \frac{\theta_k}{1-\theta_k}\right\}_{t=1}^{\infty}$ converges to zero, e.g. $\theta_t = \frac{1}{t^2}$;
for $t = 1, 2, \cdots$ do
    $\mathbf{u}_i^* \leftarrow \arg\min_{\mathbf{u}_i\in\mathcal{U}} \ell(y_i, \mathbf{V}^{(t-1)}\mathbf{P}\mathbf{u}_i) + \frac{1}{2}\mathbf{u}_i^T\mathbf{Q}(\tilde{\mathcal{W}}^{(t-1)})\mathbf{u}_i + \frac{1}{2}(1-\theta_t)^2\|\mathbf{u}_i - \mathbf{u}_i^{(t-1)}\|_2^2, \forall i$;
    $\mathbf{u}_i^{(t)} \leftarrow \mathbf{u}_i^{(t-1)} + \theta_t(\mathbf{u}_i^* - \mathbf{u}_i^{(t-1)}), \forall i$;
    $\mathbf{V}^* \leftarrow \arg\min_{\mathbf{V}\in\mathcal{V}} \sum_i \ell(y_i, \mathbf{V}\mathbf{P}\mathbf{u}_i^{(t)}) + \frac{1}{2}(1-\theta_t)^2\|\mathbf{V} - \mathbf{V}^{(t-1)}\|_F^2$;
    $\mathbf{V}^{(t)} \leftarrow \mathbf{V}^{(t-1)} + \theta_t(\mathbf{V}^* - \mathbf{V}^{(t-1)})$;
    $\tilde{\mathcal{W}}^* \leftarrow \arg\min_{\tilde{\mathcal{W}}\subseteq\mathcal{W}} \sum_i \frac{1}{2}[\mathbf{u}_i^{(t)}]^T\mathbf{Q}(\tilde{\mathcal{W}})\mathbf{u}_i^{(t)} + \frac{1}{2}(1-\theta_t)^2\sum_n\sum_{m\in\mathcal{M}_n}\|\mathbf{W}_{n,m} - \mathbf{W}_{n,m}^{(t-1)}\|_F^2$
    $\mathbf{W}_{n,m}^{(t)} \leftarrow \mathbf{W}_{n,m}^{(t-1)} + \theta_t(\mathbf{W}_{n,m}^* - \mathbf{W}_{n,m}^{(t-1)}), \forall n, \forall m \in \mathcal{M}_n, \mathbf{W}_{n,m}^* \in \tilde{\mathcal{W}}^*$;
end
return $\tilde{\mathcal{W}}$;
```

Our solutions to each of the three minimization problems are given in equations Equation 9, Equation 10, and Equation 11. These were determined by substituting the chosen loss function, squared error (SE), to each objective, and solving for the value which makes the gradient zero. The solutions to Equation 6, Equation 7, and Equation 8 are given in Equation 9, Equation 10, and Equation 11, respectively:

$$\mathbf{u_i'^*} = \left(\sum_n \gamma_n \left(\mathbf{P'}^T - \mathbf{A'}^T\mathbf{W'_n}A'\right)(\mathbf{P'} - \mathbf{W'_n}\mathbf{A'}) + \mathbf{P}^T\mathbf{V}^T\mathbf{V}\mathbf{P} + (1 + (1-\theta_t)^2)\mathbf{I}\right)^{-1}$$
$$\times \left(\sum_n \gamma_n \left(\mathbf{P'}^T - \mathbf{A'}^T\mathbf{W'_n}^T\right)\mathbf{W_{n,1}}\mathbf{u_{i,1}} + \mathbf{P}\mathbf{V^T}\mathbf{y_i} + (1-\theta_t)^2\mathbf{u_i}^{t-1}\right) \tag{9}$$

$$\mathbf{V}^* = \left(\sum_i \mathbf{y_i}\mathbf{u_i}^T\,\mathbf{P}^T(1-\theta_t)^2\mathbf{V}^{t-1}\right)\left(\mathbf{P}\sum_i \mathbf{u_i}\mathbf{u_i}^T\,\mathbf{P}^T + (1 + (1-\theta_t)^2)\mathbf{I}\right)^{-1} \tag{10}$$

$$\mathbf{W_n^*} = \left(\gamma_\mathbf{n}\mathbf{P}\sum_i \mathbf{u_i}\mathbf{u_i}^T\mathbf{A}^T + (1-\theta_\mathbf{t})^2\mathbf{W_n}^{t-1}\right)\left(\gamma_\mathbf{n}\mathbf{A}\sum_i \mathbf{u_i}\mathbf{u_i}^T\mathbf{A}^T + (1 + (1-\theta_\mathbf{t})^2)\mathbf{I}\right)^{-1} \tag{11}$$

where,

$$\begin{aligned}
\mathbf{u_i} &= \left[\mathbf{u_{i,1}}^T \ldots \mathbf{u_{i,n}}^T\right]^T \\
\mathbf{u_i'} &= \left[\mathbf{u_{i,2}}^T \ldots \mathbf{u_{i,n}}^T\right]^T \\
\mathbf{A}\mathbf{u_i} &= \left[\mathbf{u_{i,1}}^T \ldots \mathbf{u_{i,n-1}}^T\right]^T \\
\mathbf{P}\mathbf{u_i} &= \mathbf{u_{i,n}} \\
\mathbf{A'}\mathbf{u_i'} &= \left[\mathbf{u_{i,2}}^T \ldots \mathbf{u_{i,n-1}}^T\right]^T \\
\mathbf{P'}\mathbf{u_i'} &= \mathbf{u_{i,n}} \\
\mathbf{W_n} &= [\mathbf{W_{n,1}} \ldots \mathbf{W_{n,n-1}}] \\
\mathbf{W_n'} &= [\mathbf{W_{n,2}} \ldots \mathbf{W_{n,n-1}}]
\end{aligned}$$

Note that all of the optimizations include a small modification: the identity term in the matrix inversion is multiplied by $(1 + (1-\theta_\mathbf{t})^2)$ instead of $(1-\theta_\mathbf{t})^2$. The purpose of these corrections is to avoid numerical problems with matrix inversion, as without the correction the matrix to be inverted is generally singular at the first iteration. Also, the $\mathbf{u_i}$ update as stated in [16] is replaced with a $\mathbf{u_i'}$ update, which includes only the hidden layer outputs and not the input data, $\mathbf{x}$. This is because the input data is not a controllable parameter.

4 Experiments

The Tikhonov regularized BCD algorithm was implemented in Python language using NumPy and CuPy libraries. We conducted our evaluation on two datasets namely MNIST and MNIST-Fashion. Both datasets have ten classes and sixty thousand training images and another test thousand images. These images are grayscaled and are of 28x28 size. Our principal paper has been already been tested on MNIST dataset and its evaluation on MNIST-Fashion has been presented in Evalaution section.

We tested our algorithm on three layer DNN, with all γ_n set to 1. Each layer is fully connected and takes flattened image of 784X1 size as input. We have kept this network consistent through each of our evaluation. We have also compared our principal paper with more recent algorithm ie Proximal BCD. For proximal BCD's implementation we have taken inspiration from [1]. We further expanded our evaluation by comparing against Dropout regularized network that was trained using SGD and ADMM. For consistency, similar to Zhang et. al., we ran each experiment for 100 iterations.

5 Evaluation

At each epoch of the optimization, we compute predicted labels given the learned parameters of that epoch, and from that compute the loss and accuracy compared to the ground truth labels. Top row of Figure 2 shows overall mean squared error loss for the train data over epochs, for each dataset. We can observe that the loss is exponentially decreasing and converging. Bottom row of Figure 2 on the other hand shows the train and test accuracy over epochs, for each dataset. We can observe that the accuracy sharply increases after multiple epochs, and converges. With the MNIST data, the train accuracy converges to 87 percent, whereas test accuracy converges to 73 percent. While the performance is good, the lower test accuracy indicates presence of overfitting in our model. the reason that we are achieving less test accuracy than Zhang et. al's reported value of 94 percent can be attributed to two main reasons. First is due to computational limitations, we only used a subset of the data being 10,000 train and 200 test images, whereas Zhang et. al. use the full MNIST data containing 60,000 train and 10,000 test images. The second attribute is, we omitted exchanging the last layer of weights $\mathbf{V}$ with a linear support vector machine (SVM), as authors didn't provide enough detail regarding the implementation of this part, and multi-class classification with SVM is beyond the scope of our knowledge. Comparing the performance of the two datasets, we can observe that MNIST Fashion performed better than MNIST in both train and test performance.

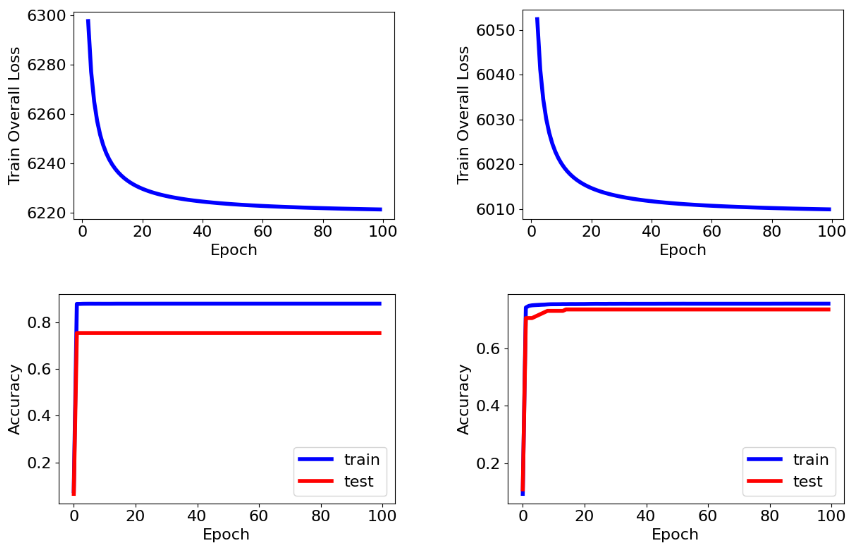

Figure 2: Tikhonov Regularized BCD implementation performance on MNIST (left) and MNIST Fashion (right) datasets.

To expand our evaluation, we took the open source implementation of Lau et. al's Proximal BCD algorithms, and tweaked some parameters to achieve similar experiment conditions. The training parameters used for this method were $\gamma = 1$, $\rho = 1$, $\alpha = 5$. We applied both the MNIST and MNIST fashion datasets for consistency, though were able to use the full datasets (60,000 train images and 10,000 test images) rather than a subset. With similar evaluation approach, Top row of Figure 3 represents the train squared loss function over epochs for the datasets, whereas the bottom row of Figure 3 represents the train and test accuracy over epochs. The test accuracy with the MNIST data is around 93-94 percent, which is very similar to that of Zhang et. al's. On contrary to our results, the MNIST fashion dataset performed more poorly compared to MNIST. Since DNN's are "black-box" models, it is difficult to pinpoint why one model classifies digits better over clothes and vice versa.

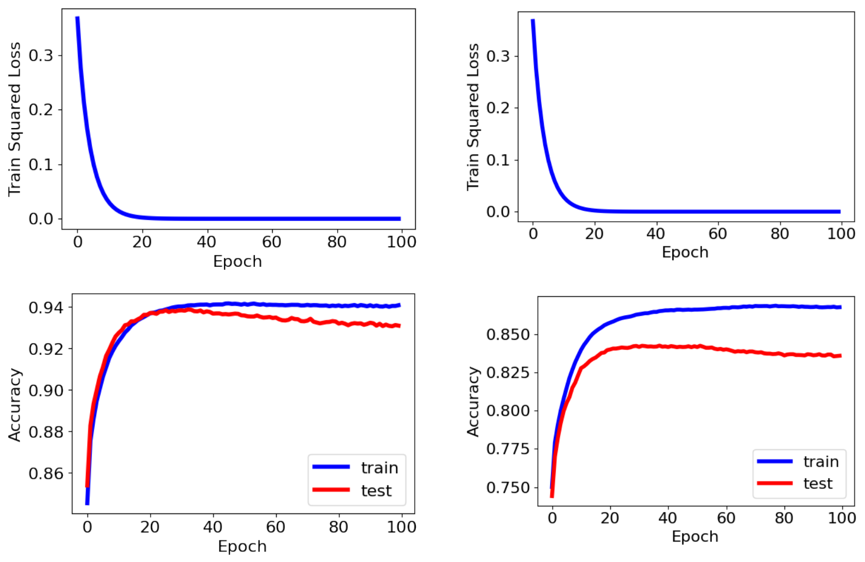

Figure 3: Proximal BCD performance on MNIST (left) and MNIST Fashion (right) datasets.

For expanding the scope of the project, and additional means for discussion, we have implemented a vanilla SGD and vanilla ADMM, both with vanilla Dropout. We did a similar performance evaluation using one of the datasets, using all available test and train instances. The parameters for SGD include a learning rate of 0.1 and momentum of 0.9, and the parameters for ADMM include a learning rate of 0.002. From Figure 4 we observe a test accuracy of 78 percent on SGD method, and a test accuracy of 86 percent in ADMM method. ADMM converges faster, and performs better under given number of iterations. Comparing test accuracy for all methods from least to highest is SGD, ADMM, BCD Proximal, BCD with Tikhanov. We can attribute the slowness of the SGD algorithm, especially in the earlier layers, to the vanishing gradients problem. Since ADMM and BCD is not gradient-based, we don't observe this slowness.

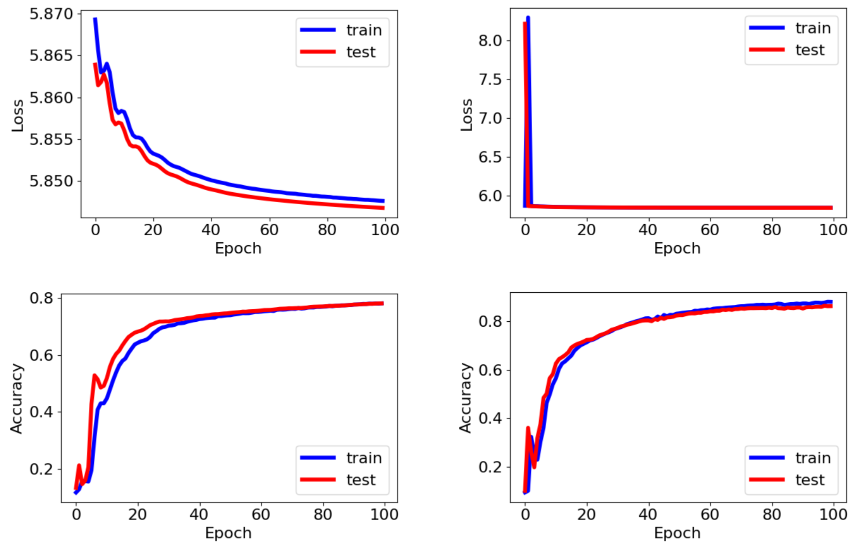

Figure 4: SGD with Dropout performance (left) vs. ADMM with Dropout (right) on MNIST Fashion Dataset.

6 Conclusion

In this work we re-implemented Zhang et al. work on a novel BCD algorithm for training DNNs. We have reevaluated their work on two datasets ie MNIST and MNIST-Fashion. We expanded our evaluation by comparing results with newer work in this field ie Proximal BCD [9]. We also implemented a Dropout regularized network and trained it using SGD and ADMM and mentioned their results in above sections. Three major advantages of BCD include: (a) high per epoch efficiency at early stages (observed in Figure 1), i.e. the training and test accuracies of BCD grow much faster than SGD in terms of epoch at the early stage; (b) good scalability, i.e. BCD can be implemented in a distributed and parallel manner via data parallelism on multi-core CPUs; (c) gradient free, i.e. gradient computations are not used for the block coordinate updates. One flaw of the BCD methods is that they generally require more memory than SGD method.

7 Contribution by Group Members:

Authors started with theoretical analysis of the Zhang et al. BCD algorithm with the aim to get close loop equation of objective functions. Author 1 took the lead in implementing the algorithm, while other authors supported in parts. Author 1 and Author 2 collaborated for theoretical analysis and implementation of the objective function V. Author 3 and Author 1 collaborated for objective function W. Author 4 and Author 3 ventured to further into recent advancements to this algorithm and theoretical analysis on Proximal BCD. Author 4 and Author 3 also collaborated to implement Proximal BCD. Author 3 and Author 2 implemented the Dropout regularized DNN and evaluated its performance. Author 1 took the lead to stitch all the parts together. Author 2 and Author 4 wrote the Abstract, and Related works and evaluation. Author 1 wrote the Methodology section and Author 3 wrote the experiments section.

References

[1] Proximal bcd. `https://github.com/timlautk/BCD-for-DNNs-PyTorch`.

[2] Hedy Attouch, Jérôme Bolte, and Benar Fux Svaiter. Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized gauss–seidel methods. *Mathematical Programming*, 137(1):91–129, Feb 2013.

[3] Heinz H. Bauschke, Jonathan, and M. Borwein. On projection algorithms for solving convex feasibility problems. *SIAM Review*, 38:367–426, 1996.

[4] Miguel A. Carreira-Perpinan and Weiran Wang. Distributed optimization of deeply nested systems. *International Conference on Artificial Intelligence and Statistics*, 2014.

[5] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys, 2017.

[6] Zheng Xu Bharat Singh Ankit Patel Gavin Taylor, Ryan Burmeister and Tom Goldstein. Training neural networks without gradients: A scalable admm approach. *International Conference on Machine Learning*, 2016.

[7] Saeed Ghadimi and Guanghui Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming, 2013.

[8] Shao-Bo Lin Jinshan Zeng, Tim Tsz-Kit Lau and Yuan Yao. Global convergence of block coordinate descent in deep learning. *International Conference on Machine Learning*, 2019.

[9] Tim Tsz-Kit Lau, Jinshan Zeng, Baoyuan Wu, and Yuan Yao. A proximal block coordinate descent algorithm for deep neural network training, 2018.

[10] Sixin Zhang Yann Le Cun Li Wan, Matthew Zeiler and Rob Fergus. Regularization of neural networks using dropconnect. *Proceedings of Machine Learning Research*, 2013.

[11] Robert Nishihara, Laurent Lessard, Ben Recht, Andrew Packard, and Michael Jordan. A general analysis of the convergence of admm. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 343–352, Lille, France, 07–09 Jul 2015. PMLR.

[12] Jorge Nocedal and Stephen J. Wright. Numerical optimization, 1999.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[14] Sida I. Wang and Christopher D. Manning. Fast dropout training. *International Conference on Machine Learning*, 2013.

[15] Yu Wang, Wotao Yin, and Jinshan Zeng. Global convergence of admm in nonconvex nonsmooth optimization, 2018.

[16] Ziming Zhang and Matthew Brand. Convergent block coordinate descent for training tikhonov regularized deep neural networks, 2017.