# Convergent Block Coordinate Descent for Training Tikhonov Regularized Deep Neural Networks

**Implementation Track**
4 Team Members

## 1 Introduction

In this project we are trying to explore Zhang et.al [7] 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 will testing this algorithm on different datasets such as Fashion MNIST.

Our goal through this proposal 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 time schedule and division of labor for these tasks (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 [5], is a popular SR method that randomly drops nodes of the DNN during training.
- 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 [2]. 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 [3], 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. Basically, 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[4] [6], but no such proof exists for the application of training DNNs.

# 3 Method

## 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 $(1 \leq n \leq N)$ 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.

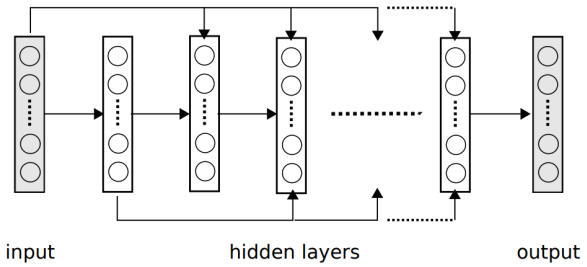


Figure 1: DNN Architecture

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 proejction on a convex set(POCS) [1] 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:

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

here $\tilde{\mathcal{U}} = \{\mathbf{u}_{i,n}\}$ and $\gamma_n \geq 0, \forall n$ denote our regularization constants. Larger $\gamma_n$ 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 $\gamma$ terms into a matrix $\mathbf{Q}$, the modified Tikhonov objective function is formed.

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

## 3.2 Block Coordinate Descent

**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** $\tilde{\mathcal{W}}$;

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 least square regression problem, and Equation 8 is a classification problem.

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

$$\min_{\forall\mathbf{W_{n,m}}\in\tilde{\mathcal{W}}} \sum_i \frac{\gamma_n}{2} \left\| \mathbf{u}_{i,n} - \sum_{m\in\mathcal{M}_n} \mathbf{W}_{n,m}\mathbf{u}_{i,m} \right\|_2^2 \tag{7}$$

$$\min_{\mathbf{V}\in\mathcal{V}} \ell\left(y_i, \mathbf{V}\mathbf{P}\mathbf{u}_i\right) \tag{8}$$

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 $\gamma_n$, and the output is the weights $\tilde{W}$.

## 4 Experiments

a. Implementation: Our implementation will in Python utlilizing libraries such as PyTorch, NumPy, and Pandas

b. Data set: In contrast to our principal paper we will using MNIST Fashion dataset for training and testing our network.

c. Evaluation: We will evaluating our algorithm on various levels. First we be testing on different complexities of DNNs such as 3 layer vs 10 layer DNN. Then compare our algorithm against various variants of SGD on multiple parameters such as convergence, testing error (Mean Squared Error) and computational time.

## 5 Plan of Project

a. Work division in the team is as follows

- Author 1: Lead the effort, share expertise, coordinate and assign tasks in implementing the novel BCD algorithm.
- Author 2: Test the algorithm on a small subset of the data and asses the results. Understand theory as to why the algorithm converges and explain it to the team.
- Author 3: Use the novel BCD algorithm to replicate the results in the paper and compare the results with a few variants of SGD as done in the paper.
- Author 4: Apply the novel BCD algorithm to a new data set and compare the results with a few variants of SGD (different from Author 3) as done in paper.

b. Author 1 has industry experience in working with Neural Networks, the rest don't have any prior experience apart from the content taught in EECS 545. Author 1 will lead the project and share his expertise to complete the project successfully.

c. As our team is working with Deep Neural Networks the training time is the biggest hurdle.

d. Milestones that the team set:

- Nov 5 - Nov 10, 2022: Every author has an in depth idea of how the algorithm works.
- Nov 11 - Nov 18, 2022: Code the algorithm and test it on a local setup with small subsets of data.
- Nov 18 - Nov 28, 2022: Reproduce the results from the paper and compare it with a few SGD variants as done in the paper.
- Nov 28 - Dec 10, 2022: Use a new data set to apply the algorithm and compare it with a few SGD variants as done in the paper.

e. If something goes wrong during the process we will stick to the steps described but work with a smaller data set and report results on that data set.

## References


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

[2] 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.

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

[4] 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.

[5] 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.

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

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