Toward Model Parallelism for Deep Neural Network based on Gradient-free ADMM Framework

Junxiang Wang
Emory University
jwan936@emory.edu

Zheng Chai,Yue Cheng
George Mason University
{zchai2,yuecheng}@gmu.edu

Liang Zhao*
Emory University
lzhao41@emory.edu
*corresponding author

***Abstract*—Alternating Direction Method of Multipliers (ADMM) has recently been proposed as a potential alternative optimizer to the Stochastic Gradient Descent(SGD) for deep learning problems. This is because ADMM can solve gradient vanishing and poor conditioning problems. Moreover, it has shown good scalability in many large-scale deep learning applications. However, there still lacks a parallel ADMM computational framework for deep neural networks because of layer dependency among variables. In this paper, we propose a novel parallel deep learning ADMM framework (pdADMM) to achieve layer parallelism: parameters in each layer of neural networks can be updated independently in parallel. The convergence of the proposed pdADMM to a critical point is theoretically proven under mild conditions. The convergence rate of the pdADMM is proven to be $o(1/k)$ where k is the number of iterations. Extensive experiments on six benchmark datasets demonstrated that our proposed pdADMM can lead to more than 10 times speedup for training large-scale deep neural networks, and outperformed most of the comparison methods. Our code is available at: *https://github.com/xianggebenben/pdADMM*.**

***Index Terms*—Model Parallelism, Deep Neural Network, Alternating Direction Method of Multipliers, Convergence**

I. Introduction

Due to wide applications and significant success in various applications, the training of deep neural network models has gained ever-increasing attention from the machine learning community. Gradient-based methods such as Stochastic Gradient Descent (SGD) and its variants have been considered as the state-of-the-art since the 1980s, mainly due to its superior performance. Despite the popularity, in recent years, the constant improvement of DNNs' performance is accompanied by a fast increase in models' complexity and size, which indicates a clear trend toward larger and deeper networks. Such a trend leads to severe challenges for large models to be fit into a single computing unit (e.g., GPU), and raises urgent demands for partitioning the model into different computing devices to parallelize training. However, the inherent bottleneck from backpropagation which prevents the gradients of different layers being calculated in parallel. This is because in backpropagation the gradient calculations of one layer tightly depend on and have to wait for the calculated results of all the previous layers, which prevents the gradients of different layers being calculated in parallel.

To work around the drawback from gradient-based methods, gradient-free methods have caught fast increasing attention in recent years, which aims to address the drawbacks such as strong dependency among layers, gradient vanishing (i.e. the error signal diminishes as the gradient is backpropagated), and poor conditioning (i.e. a small input can change the gradient dramatically). For example, Talyor et al. and Wang et al. presented an Alternating Direction Method of Multipliers (ADMM) algorithm to train neural network models [1], [2]. Moreover, extensive experiments have revealed that the ADMM outperformed most of SGD-related methods [2]. Amongst the gradient-free methods for deep learning optimization, ADMM-based methods are deemed to have great potential of parallelism of deep neural network training, due to its inherent nature, which can break an objective into multiple subproblems, each of which can be solved in parallel [3].

Despite the potential, a parallel algorithm based on ADMM for deep neural network training has rarely been explored, developed, and evaluated until now, due to the layer dependency among subproblems of ADMM. Even though the ADMM reduces the layer dependency compared with SGD, one subproblem of ADMM is dependent on its previous subproblem. Therefore, existing ADMM-based optimizers still update parameters sequentially.

To handle the difficulties of layer dependency, in this paper we propose a novel parallel deep learning Alternating Direction Method of Multipliers (pdADMM) optimization framework to train large-scale neural networks. Our contributions in this paper include:

- We propose a novel reformulation of the feed-forward neural network problem, which splits a neural network into independent layer partitions and allows for ADMM to achieve model parallelism.
- We present a model-parallelism version of the ADMM algorithm to train a feed-forward deep neural network. All parameters in each layer can be updated in parallel to speed up the training process significantly. All subproblems generated by the pdADMM algorithm are discussed in detail.
- We investigated the convergence properties of parallel ADMM in the common nonlinear activation functions such as the Rectified linear unit (Relu), and we prove that the pdADMM converges to a state-of-the-art critical point with a sublinear convergence rate $o(1/k)$.
- We conduct extensive experiments on six benchmark datasets to show the massive speedup of the proposed

2374-8486/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDM50108.2020.00068

pdADMM as well as its competitive performance with state-of-the-art optimizers.

The organization of this paper is shown as follows: In Section II, we summarize recent related research work to this paper. In Section III, we formulate the novel pdADMM algorithm to train a feed-forward neural network. In Section IV, the convergence guarantee of pdADMM to a critical point is provided. Extensive experiments on benchmark datasets to demonstrate the convergence, speedup and comparable performance of pdADMM are shown in Section V, and Section VI concludes this work.

II. Related Work

Distributed ADMM ADMM is one of the commonly applied techniques in distributed optimization. Overall, the previous works on distributed ADMM can be classified into two categories: synchronous problems and asynchronous problems. Synchronous problems usually require workers to optimize parameters in time before the master update the consensus variable, while asynchronous problems allow some workers to delay parameter updates. Most literature focused on the application of the distributed ADMM on synchronous problems. For example, Mota et al. utilized the distributed ADMM for the congestion control problem [4]; Makhdoumi and Ozdaglar studied the convergence properties of the distributed ADMM on the network communication problem. For more work, please refer to [5], [6], [7], [8], [9]. On the other hand, a handful of papers investigated how asynchronous problems can be addressed by distributed ADMM. For instance, Zhang et al., Wei et al., Chang et al and Hong proved the convergence of the distributed ADMM on asynchronous problems [10], [11], [12], [13], [14]. Kumar et al. discussed the application of the ADMM on multi-agent problems over heterogeneous networks [15]. However, there still lacks a general framework for ADMM to train deep neural networks in the distributed fashion.

Convergence analysis of nonconvex ADMM: Despite the outstanding performance of the nonconvex ADMM, its convergence theory is not well established due to the complexity of both coupled objectives and various (inequality and equality) constraints. Specifically, Magnusson et al. provided new convergence conditions of ADMM for a class of nonconvex structured optimization problems [16]; Li and Pong investigated the properties of the nonconvex ADMM on the composite optimization problem [17]; Wang et al. presented mild convergence conditions of the nonconvex ADMM where the objective function can be coupled and nonsmooth [18]; Hong et al. proved that the classic ADMM converges to stationary points provided that the penalty parameter is sufficiently large [19]; Wang et al. proved the convergence of multi-convex ADMM with inequality constraints [20]; Liu et al. proved the convergence properties of a parallel and linearized ADMM [21]. Wang and Zhao studied the convergence conditions of the nonconvex ADMM in the nonlinearly constrained equality problems [22]; Xie et al. proposed a deep-learning-based ADMM algorithm to study the constrained optimization problems [23]. Wang et al. gave the first convergence proof of ADMM in the nonconvex deep learning problems [2], [24]. For more work, please refer to [25], [26], [27], [28], [29], [30].

Distributed Deep Learning With the increased volume of data and layers of neural networks, there is a need to design distributed systems to train a deep neural network for large-scale applications. Most recent papers have proposed gradient-based distributed systems to train neural networks: For example, Wen et al. proposed Terngrad to accelerate distributed deep learning in data parallelism [31]; Sergeev et al. presented an open-source library Horovod to reduce communication overhead [32]. Other systems include SINGA [33] Mxnet[34], TicTac [35] and Poseidon [36].

Data and Model Parallelism Data parallelism focuses on distributing data across different processors, which can be implemented in parallel. Scaling SGD is one of the most common ways to reach data parallelism [37]. For example, the distributed architecture, Poseidon, is achieved by scaling SGD through overlapping communication and computation over networks. The recently proposed ADMM [1], [2] is another way of data parallelism: each subproblem generated by ADMM can be solved in parallel. However, data parallelism suffers from the bottleneck of a neural network: for SGD, the gradient should be transmitted through all processors; for ADMM, the parameters in one layer are subject to these in its previous layer. As a result, this leads to heavy communication cost and time delay. Model parallelism, however, can solve this challenge because model parallelism splits a neural network to many independent partitions. In this way, each partition can be optimized in parallel and hence reduce time delay. For instance, Parpas and Muir proposed a parallel-in-time method from the perspective of dynamic systems [38]; Huo et al. introduced a feature replay algorithm to achieve model parallelism [39]. Zhuang et al. broke layer dependency by introducing the delayed gradient [40]. However, to the best of our knowledge, there still lacks an exploration on how to achieve model parallelism via ADMM.

III. pdADMM Algorithm

We propose the pdADMM algorithm in this section. Specifically, Section III-A introduces the existing deep learning ADMM method, and reformulates the problem and presents the pdADMM algorithm in detail. Section III-B discusses all subproblems generated by pdADMM and the strategy to train a large-scale deep neural network via pdADMM.

A. Background

In this section, we introduce the formulation of the feed-forward neural network training problem and an existing deep learning ADMM method. The important notations of this paper are detailed in Table I.

The feed-forward neural network is formulated as follows [2]:

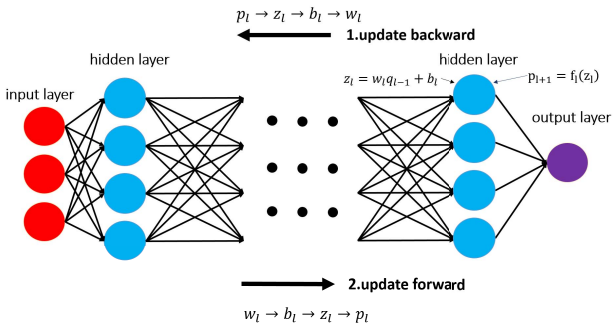


Fig. 1: The overview of existing dlADMM algorithm: parameters are updated in a sequential fashion.

Notations	Descriptions
L	Number of layers.
W_l	The weight matrix for the l-th layer.
b_l	The intercept vector for the l-th layer.
z_l	The auxiliary variable of the linear mapping for the l-th layer.
$f_l(z_l)$	The nonlinear activation function for the l-th layer.
p_l	The input for the l-th layer.
q_l	The output for the l-th layer.
x	The input matrix of the neural network.
y	The predefined label vector.
$R(z_L, y)$	The risk function for the l-th layer.
n_l	The number of neurons for the l-th layer.

TABLE I: Important Notations

Problem 1.

$$\min_{W_l, b_l, z_l, p_l} R(z_L; y)$$
$$s.t.\ z_l = W_l p_l + b_l,\ p_{l+1} = f_l(z_l)(l = 1, \cdots, L-1)$$

where $p_1 = x \in \mathbb{R}^{n_0}$ is the input of the deep neural network where n_0 is the number of feature dimensions, and y is a predefined label vector. p_l is the input for the l-th layer, also the output for the $(l-1)$-th layer. $R(z_L; y)$ is a risk function for the L-th layer, which is convex, continuous and proper. $z_l = W_l p_l + b_l$ and $p_{l+1} = f_l(z_l)$ are linear and nonlinear mappings for the l-th layer, respectively.

Problem 1 has been addressed by deep learning Alternating Direction Method of Multipliers (dlADMM) [2]. As shown in Figure 1, the dlADMM algorithm updates parameters from the final layer, and moves backward to the first layer, then updates parameters forward from the first layer to the final layer, in order to exchange information efficiently. However, parameters in one layer are dependent on its neighboring layers, and hence can not achieve parallelism. For example, the update of p_{l+1} on the $l+1$-th layer needs to wait before z_l on the l-th layer is updated. In order to address layer dependency, we relax Problem 1 to Problem 2 as follows:

Problem 2.

$$\min_{\boldsymbol{p}, \boldsymbol{W}, \boldsymbol{b}, \boldsymbol{z}, \boldsymbol{q}} F(\boldsymbol{p}, \boldsymbol{W}, \boldsymbol{b}, \boldsymbol{z}, \boldsymbol{q}) = R(z_L; y)$$
$$+ (\nu/2)(\sum_{l=1}^{L} \|z_l - W_l p_l - b_l\|_2^2 + \sum_{l=1}^{L-1} \|q_l - f_l(z_l)\|_2^2)$$
$$s.t.\ p_{l+1} = q_l$$

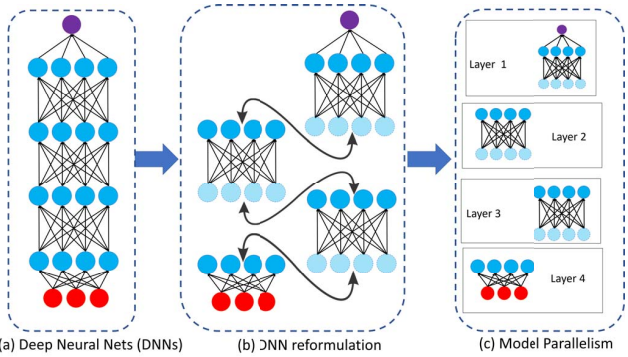


Fig. 2: The pdADMM optimization framework: an overview

where $\mathbf{p} = \{p_l\}_{l=1}^{L}$, $\mathbf{W} = \{W_l\}_{l=1}^{L}$, $\mathbf{b} = \{b_l\}_{l=1}^{L}$, $\mathbf{z} = \{z_l\}_{l=1}^{L}$, $\mathbf{q} = \{q_l\}_{l=1}^{L-1}$, and $\nu > 0$ is a tuning parameter. As $\nu \to \infty$, Problem 2 approaches Problem 1. We reduce layer dependency by splitting the output of the l-th layer and the input of the $l+1$-th layer into two variables p_{l+1} and q_l, respectively.

The high-level overview of the pdADMM algorithm is shown in Figure 2. Specifically, by breaking the whole neural network into multiple layers, each of which can be optimized by an independent worker. Therefore, the layerwise training can be implemented in parallel. Moreover, the gradient vanishing problem can be avoided in this way. This is because the accumulate gradient calculated by the backpropagation algorithm is split into layerwise components.

Now we follow the ADMM routine to solve Problem 2, the augmented Lagrangian function is formulated mathematically as follows:

$$L_\rho(\mathbf{p}, \mathbf{W}, \mathbf{b}, \mathbf{z}, \mathbf{q}, \mathbf{u})$$
$$= F(\mathbf{p}, \mathbf{W}, \mathbf{b}, \mathbf{z}, \mathbf{q}) + \sum_{l=1}^{L-1} (u_l^T (p_{l+1} - q_l) + (\rho/2)\|p_{l+1} - q_l\|_2^2)$$
$$= R(z_L; y) + \phi(p_1, W_1, b_1, z_1) + \sum_{l=2}^{L} \phi(p_l, W_l, b_l, z_l, q_{l-1}, u_{l-1})$$
$$+ (\nu/2) \sum_{l=1}^{L-1} \|q_l - f_l(z_l)\|_2^2$$

where $\phi(p_1, W_1, b_1, z_1) = (\nu/2)\|z_1 - W_1 p_1 - b_1\|_2^2$, $\phi(p_l, W_l, b_l, z_l, q_{l-1}, u_{l-1}) = (\nu/2)\|z_l - W_l p_l - b_l\|_2^2 + u_{l-1}^T (p_l - q_{l-1}) + (\rho/2)\|p_l - q_{l-1}\|_2^2$, $u_l (l = 1, \cdots, L-1)$ are dual variables, $\rho > 0$ is a parameter, and $\mathbf{u} = \{u_l\}_{l=1}^{L-1}$. The detail of the pdADMM is shown in Algorithm 1. Specifically, Lines 5-9 update primal variables $\mathbf{p}$, $\mathbf{W}$, $\mathbf{b}$, $\mathbf{z}$ and $\mathbf{q}$, respectively, while Line 11 updates the dual variable $\mathbf{u}$. the discussion on how to solve subproblems generated by pdADMM is detailed in the next section.

B. Solutions to All Subproblems

In this section, we discuss how to solve all subproblems generated by pdADMM in detail.

Algorithm 1 the pdADMM Algorithm

Require: y, $p_1 = x$, ρ, ν.
Ensure: $\mathbf{p}, \mathbf{W}, \mathbf{b}, \mathbf{z}, \mathbf{q}$.
Initialize $k = 0$.
while $\mathbf{p}^k, \mathbf{W}^k, \mathbf{b}^k, \mathbf{z}^k, \mathbf{q}^k$ not converged **do**
Update p_l^{k+1} of different l by Equation (1) in parallel.
Update W_l^{k+1} of different l by Equation (2) in parallel.
Update b_l^{k+1} of different l by Equation (3) in parallel.
Update z_l^{k+1} of different l by Equations (4) and (5) in parallel.
Update q_l^{k+1} of different l by Equation (6) in parallel.
$r_l^k \leftarrow p_{l+1}^{k+1} - q_l^{k+1} (l = 1, \cdots, L)$ in parallel # Compute residuals.
Update u_l^{k+1} of different l by Equation (7) in parallel.
$k \leftarrow k + 1$.
end while
Output $\mathbf{p}, \mathbf{W}, \mathbf{b}, \mathbf{z}, \mathbf{q}$.

1. Update $\mathbf{p}^{k+1}$

The variable $\mathbf{p}^{k+1}$ is updated as follows:

$$\begin{aligned} p_l^{k+1} &\leftarrow \arg\min_{p_l} L_\rho(\mathbf{p}, \mathbf{W}^k, \mathbf{b}^k, \mathbf{z}^k, \mathbf{q}^k, \mathbf{u}^k) \\ &= \phi(p_l, W_l^k, b_l^k, z_l^k, q_{l-1}^k, u_{l-1}^k) \end{aligned}$$

Because W_l and p_l are coupled in ϕ, solving p_l should require the time-consuming operation of matrix inversion of W_l. To handle this, we apply similar quadratic approximation techniques as used in dlADMM [2] as follows:

$$p_l^{k+1} \leftarrow \arg\min_{p_l} U_l(p_l; \tau_l^{k+1}) \tag{1}$$

where $U_l(p_l; \tau_l^{k+1}) = \phi(p_l^k, W_l^k, b_l^k, z_l^k, q_{l-1}^k, u_{l-1}^k) + (\nabla_{p_l^k}\phi(p_l^k, W_l^k, b_l^k, z_l^k, q_{l-1}^k, u_{l-1}^k)(p_l - p_l^k) + (\tau_l^{k+1}/2)\|p_l - p_l^k\|_2^2$, and $\tau_l^{k+1} > 0$ is a parameter. τ_l^{k+1} should satisfy $\phi(p_l^{k+1}, W_l^k, b_l^k, z_l^k, q_{l-1}^k, u_{l-1}^k) \le U_l(p_l^{k+1}; \tau_l^{k+1})$. The solution to Equation (1) is: $p_l^{k+1} \leftarrow p_l^k - \nabla_{p_l^k}\phi(p_l^k, W_l^k, b_l^k, z_l^k, q_{l-1}^k, u_{l-1}^k)/\tau_l^{k+1}$.

2. Update $\mathbf{W}^{k+1}$

The variable $\mathbf{W}^{k+1}$ is updated as follows:

$$\begin{aligned} W_l^{k+1} &\leftarrow \arg\min_{W_l} L_\rho(\mathbf{p}^{k+1}, \mathbf{W}, \mathbf{b}^k, \mathbf{z}^k, \mathbf{q}^k, \mathbf{u}^k) \\ &= \arg\min_{W_l} \begin{cases} \phi(p_1^{k+1}, W_1, b_1^k, z_1^k) & l = 1 \\ \phi(p_l^{k+1}, W_l, b_l^k, z_l^k, q_{l-1}^k, u_{l-1}^k) & 1 < l \le L \end{cases} \end{aligned}$$

Similar to updating p_l, the following subproblem should be solved instead:

$$W_l^{k+1} \leftarrow \arg\min_{W_l} V_l(W_l; \theta_l^{k+1}) \tag{2}$$

where

$$\begin{aligned} &V_1(W_1; \theta_1^{k+1}) = \phi(p_1^{k+1}, W_1^k, b_1^k, z_1^k) \\ &+ \nabla_{W_1^k}\phi^T(p_1^{k+1}, W_1^k, b_1^k, z_1^k)(W_1 - W_1^k) \\ &+ (\theta_l^{k+1}/2)\|W_1 - W_1^k\|_2^2 \\ &V_l(W_l; \theta_l^{k+1}) = \phi(p_l^{k+1}, W_l^k, b_l^k, z_l^k, q_{l-1}^k, u_{l-1}^k) \\ &+ \nabla_{W_l^k}\phi^T(p_l^{k+1}, W_l^k, b_l^k, z_l^k, q_{l-1}^k, u_{l-1}^k)(W_l - W_l^k) \\ &+ (\theta_l^{k+1}/2)\|W_l - W_l^k\|_2^2 \end{aligned}$$

and θ_l^{k+1} is a parameter, which should satisfy $\phi(p_1^{k+1}, W_1^{k+1}, b_1^k, z_1^k) \le V(W_1^{k+1}; \theta_1^{k+1})$ and $\phi(p_l^{k+1}, W_l^{k+1}, b_l^k, z_l^k, q_{l-1}^k, u_{l-1}^k) \le V(W_l^{k+1}; \theta_l^{k+1})(1 < l < L)$. The solution to Equation (2) is shown as follows:

$$W_l^{k+1} \leftarrow W_l^k - \begin{cases} \nabla_{W_1^k}\phi(p_1^{k+1}, W_1^k, b_1^k, z_1^k)/\theta_l^{k+1} & l = 1 \\ \nabla_{W_l^k}\phi(p_l^{k+1}, W_l^k, b_l^k, z_l^k, q_{l-1}^k, u_{l-1}^k)/\theta_l^{k+1} & 1 < l \le L \end{cases}$$

3. Update $\mathbf{b}^{k+1}$

The variable $\mathbf{b}^{k+1}$ is updated as follows:

$$\begin{aligned} b_l^{k+1} &\leftarrow \arg\min_{b_l} L_\rho(\mathbf{p}^{k+1}, \mathbf{W}^{k+1}, \mathbf{b}, \mathbf{z}^k, \mathbf{q}^k, \mathbf{u}^k) \\ &= \arg\min_{b_l} \begin{cases} \phi(p_1^{k+1}, W_1^{k+1}, b_1, z_1^k) & l = 1 \\ \phi(p_l^{k+1}, W_l^{k+1}, b_l, z_l^k, q_{l-1}^k, u_{l-1}^k) & 1 < l \le L \end{cases}. \end{aligned}$$

Similarly, we solve the following subproblems instead:

$$\begin{aligned} b_1^{k+1} &\leftarrow \arg\min_{b_1} \phi(p_1^{k+1}, W_1^{k+1}, b_1^k, z_1^k) \\ &+ \nabla_{b_1^k}\phi^T(p_1^{k+1}, W_1^{k+1}, b_1^k, z_1^k)(b_l - b_l^k) + (\nu/2)\|b_l - b_l^k\|_2^2 \end{aligned}$$

$$\begin{aligned} b_l^{k+1} &\leftarrow \arg\min_{b_l} \phi(p_l^{k+1}, W_l^{k+1}, b_l^k, z_l^k, q_{l-1}^k, u_{l-1}^k) \\ &+ \nabla_{b_l^k}\phi^T(p_l^{k+1}, W_l^{k+1}, b_l^k, z_l^k, q_{l-1}^k, u_{l-1}^k)(b_l - b_l^k) \\ &+ (\nu/2)\|b_l - b_l^k\|_2^2 (1 < l \le L) \end{aligned} \tag{3}$$

The solution to Equation (3) is:

$$b_l^{k+1} \leftarrow b_l^k - \begin{cases} \nabla_{b_1^k}\phi(p_1^{k+1}, W_1^{k+1}, b_1^k, z_1^k)/\nu & l = 1 \\ \nabla_{b_l^k}\phi(p_l^{k+1}, W_l^{k+1}, b_l^k, z_l^k, q_{l-1}^k, u_{l-1}^k)/\nu & 1 < l \le L \end{cases}$$

4. Update $\mathbf{z}^{k+1}$

The variable $\mathbf{z}^{k+1}$ is updated as follows:

$$\begin{aligned} z_l^{k+1} &\leftarrow \arg\min_{z_l} (\nu/2)\|z_l - W_l^{k+1}p_l^{k+1} - b_l^{k+1}\|_2^2 \\ &+ (\nu/2)\|q_l^k - f_l(z_l)\|_2^2 + (\nu/2)\|z_l - z_l^k\|_2^2 (l < L) \end{aligned} \tag{4}$$

$$z_L^{k+1} \leftarrow \arg\min_{z_l} R(z_L; y) + (\nu/2)\|z_L - W_L^{k+1}p_L^{k+1} - b_L^{k+1}\|_2^2 \tag{5}$$

where a quadratic term $(\nu/2)\|z_l - z_l^k\|_2^2$ is added in Equation (4) to control z_l^{k+1} to close to z_l^k. Equation (5) is convex, which can be solved by Fast Iterative Soft Thresholding Algorithm (FISTA) [41].

For Equation (4), nonsmooth activations usually lead to closed-form solutions [2], [22]. For example, for Relu $f_l(z_l) = \max(z_l, 0)$, the solution to Equation (4) is shown as follows:

$$z_l^{k+1} = \begin{cases} \min((W_l^{k+1}p_l^{k+1} + b_l^{k+1} + z_l^k)/2, 0) & z_l^{k+1} \le 0 \\ \max((W_l^{k+1}p_l^{k+1} + b_l^{k+1} + q_l^k + z_l^k)/3, 0) & z_l^{k+1} \ge 0 \end{cases}$$

For smooth activations such as tanh and sigmoid, a lookup-table is recommended [2].

5. Update $\mathbf{q}^{k+1}$

The variable $\mathbf{q}^{k+1}$ is updated as follows:

$$q_l^{k+1} \leftarrow \arg\min_{q_l} L_\rho(\mathbf{p}^{k+1}, \mathbf{W}^{k+1}, \mathbf{b}^{k+1}, \mathbf{z}^{k+1}, \mathbf{q}, \mathbf{u}^k) = \arg\min_{q_l} \phi(p_{l+1}^{k+1}, W_{l+1}^{k+1}, b_{l+1}^{k+1}, z_{l+1}^{k+1}, q_l, u_l^k). \quad (6)$$

Equation (6) has a closed-form solution as follows:

$$q_l^{k+1} \leftarrow p_{l+1}^{k+1} + u_l^k/\rho + f_l(z_l^{k+1}))/2$$

6. Update $\mathbf{u}^{k+1}$

The variable $\mathbf{u}^{k+1}$ is updated as follows:

$$u_l^{k+1} \leftarrow u_l^k + \rho(p_{l+1}^{k+1} - q_l^{k+1}) \quad (7)$$

Finally, Our proposed pdADMM can be efficient for training a deep feed-forward neural network. To achieve this, we begin from training a swallow neural network with the first few layers of the deep neural network, then more layers are added for training step by step until finally all layers are involved in the training process. The pdADMM can achieve good performance as well as reduce training cost by this strategy.

IV. Convergence Analysis

In this section, the theoretical convergence of the proposed pdADMM algorithm. Firstly, the Lipschitz continuity and coercivity are defined as follows:

Definition 1. *(Lipschitz Continuity) A function $g(x)$ is Lipschitz continuous if there exists a constant $D > 0$ such that $\forall x_1, x_2$, the following holds*

$$\|g(x_1) - g(x_2)\| \leq D\|x_1 - x_2\|.$$

Definition 2. *(Coercivity) A function $h(x)$ is coerce over the feasible set $\mathscr{F}$ means that $h(x) \to \infty$ if $x \in \mathscr{F}$ and $\|x\| \to \infty$.*

Then the following assumption is required for convergence analysis.

Assumption 1. *$f_l(z_l)$ is Lipschitz continuous with coefficient $S > 0$, and $F(\boldsymbol{p}, \boldsymbol{W}, \boldsymbol{b}, \boldsymbol{z}, \boldsymbol{q})$ is coercive. Moreover, $\partial f_l(z_l)$ is bounded, i.e. there exists $M > 0$ such that $\|\partial f_l(z_l)\| \leq M$.*

Assumption 1 is mild to satisfy: most common activation functions such as Relu and leaky Relu satisfy Assumption 1. No assumption is needed on the risk function $R(z_l; y)$, which shows that the convergence condition of our proposed pdADMM is milder than that of the dlADMM, which requires $R(z_l; y)$ to be Lipschitz differentiable [2]. Due to space limit, the detailed proofs are provided in the Appendix*. The technical proofs follow the similar routine as dlADMM [2]. The difference consists in the fact that the dual variable u_l is controlled by q_l and z_l (Lemma 5 in the Appendix), which holds under Assumption 1, while u_l can be controlled only by z_l in the convergence proof of dlADMM. The first lemma shows that the objective keeps decreasing when ρ is sufficently large.

*Proofs: https://github.com/xianggebenben/Junxiang_Wang/blob/master/supplementary_material/ICDM2020/pdADMM.pdf.

Lemma 1 (Decreasing Objective)**.** *If $\rho > \max(4\nu S^2, (\sqrt{17}+1)\nu/2)$, there exist $C_1 = \nu/2 - 2\nu^2 S^2/\rho > 0$ and $C_2 = \rho/2 - 2\nu^2/\rho - \nu/2 > 0$ such that it holds for any $k \in \mathbb{N}$ that*

$$\begin{aligned} &L_\rho(\boldsymbol{p}^k, \boldsymbol{W}^k, \boldsymbol{b}^k, \boldsymbol{z}^k, \boldsymbol{q}^k, \boldsymbol{u}^k) - L_\rho(\boldsymbol{p}^{k+1}, \boldsymbol{W}^{k+1}, \boldsymbol{b}^{k+1}, \boldsymbol{z}^{k+1}, \boldsymbol{q}^{k+1}, \boldsymbol{u}^{k+1}) \\ &\geq \sum_{l=2}^{L} (\tau_l^{k+1}/2)\|p_l^{k+1} - p_l^k\|_2^2 + \sum_{l=1}^{L} (\theta_l^{k+1}/2)\|W_l^{k+1} - W_l^k\|_2^2 \\ &+ \sum_{l=1}^{L} (\nu/2)\|b_l^{k+1} - b_l^k\|_2^2 + \sum_{l=1}^{L-1} C_1\|z_l^{k+1} - z_l^k\|_2^2 \\ &+ (\nu/2)\|z_L^{k+1} - z_L^k\|_2^2 + \sum_{l=1}^{L-1} C_2\|q_l^{k+1} - q_l^k\|_2^2 \end{aligned} \quad (8)$$

The second Lemma illustrates that the objective is bounded from below when ρ is large enough, and all variables are bounded.

Lemma 2 (Bounded Objective)**.** *If $\rho > \nu$, then $L_\rho(\boldsymbol{p}^k, \boldsymbol{W}^k, \boldsymbol{b}^k, \boldsymbol{z}^k, \boldsymbol{q}^k, \boldsymbol{u}^k)$ is lower bounded. Moreover, $\boldsymbol{p}^k, \boldsymbol{W}^k, \boldsymbol{b}^k, \boldsymbol{z}^k, \boldsymbol{q}^k$, and $\boldsymbol{u}^k$ are bounded, i.e. there exist $\mathbb{N}_{\boldsymbol{p}}$, $\mathbb{N}_{\boldsymbol{W}}$, $\mathbb{N}_{\boldsymbol{b}}$, $\mathbb{N}_{\boldsymbol{z}}$, $\mathbb{N}_{\boldsymbol{q}}$, and $\mathbb{N}_{\boldsymbol{u}} > 0$, such that $\|\boldsymbol{p}^k\| \leq \mathbb{N}_{\boldsymbol{p}}$, $\|\boldsymbol{W}^k\| \leq \mathbb{N}_{\boldsymbol{W}}$, $\|\boldsymbol{b}^k\| \leq \mathbb{N}_{\boldsymbol{b}}$, $\|\boldsymbol{z}^k\| \leq \mathbb{N}_{\boldsymbol{z}}$, $\|\boldsymbol{q}^k\| \leq \mathbb{N}_{\boldsymbol{q}}$, and $\|\boldsymbol{u}^k\| \leq \mathbb{N}_{\boldsymbol{u}}$.*

Based on Lemmas 1 and 2, the following theorem ensures that the objective is convergent.

Theorem 1 (Convergent Objective)**.** *If $\rho > \max(4\nu S^2, (\sqrt{17}+1)\nu/2)$, then $L_\rho(\boldsymbol{p}^k, \boldsymbol{W}^k, \boldsymbol{b}^k, \boldsymbol{z}^k, \boldsymbol{q}^k, \boldsymbol{u}^k)$ is convergent. Moreover, $\lim_{k\to\infty} \|\boldsymbol{p}^{k+1} - \boldsymbol{p}^k\|_2^2 = 0$, $\lim_{k\to\infty} \|\boldsymbol{W}^{k+1} - \boldsymbol{W}^k\|_2^2 = 0$, $\lim_{k\to\infty} \|\boldsymbol{b}^{k+1} - \boldsymbol{b}^k\|_2^2 = 0$, $\lim_{k\to\infty} \|\boldsymbol{z}^{k+1} - \boldsymbol{z}^k\|_2^2 = 0$, $\lim_{k\to\infty} \|\boldsymbol{q}^{k+1} - \boldsymbol{q}^k\|_2^2 = 0$, $\lim_{k\to\infty} \|\boldsymbol{u}^{k+1} - \boldsymbol{u}^k\|_2^2 = 0$.*

Proof. From Lemmas 1 and 2, we know that $L_\rho(\mathbf{p}^k, \mathbf{W}^k, \mathbf{b}^k, \mathbf{z}^k, \mathbf{q}^k, \mathbf{u}^k)$ is convergent because a monotone bounded sequence converges. Moreover, we take the limit on the both sides of Inequality (8) to obtain

$$\begin{aligned} 0 &= \lim_{k\to\infty} L_\rho(\mathbf{p}^k, \mathbf{W}^k, \mathbf{b}^k, \mathbf{z}^k, \mathbf{q}^k, \mathbf{u}^k) \\ &- \lim_{k\to\infty} L_\rho(\mathbf{p}^{k+1}, \mathbf{W}^{k+1}, \mathbf{b}^{k+1}, \mathbf{z}^{k+1}, \mathbf{q}^{k+1}, \mathbf{u}^{k+1}) \\ &\geq \lim_{k\to\infty} (\sum_{l=2}^{L} (\tau_l^{k+1}/2)\|p_l^{k+1} - p_l^k\|_2^2 \\ &+ \sum_{l=1}^{L} (\theta_l^{k+1}/2)\|W_l^{k+1} - W_l^k\|_2^2 + \sum_{l=1}^{L} (\nu/2)\|b_l^{k+1} - b_l^k\|_2^2 \\ &+ \sum_{l=1}^{L-1} C_1\|z_l^{k+1} - z_l^k\|_2^2 + (\nu/2)\|z_L^{k+1} - z_L^k\|_2^2 \\ &+ \sum_{l=1}^{L-1} C_2\|q_l^{k+1} - q_l^k\|_2^2) \geq 0 \end{aligned}$$

Because $L_\rho(\mathbf{p}^k, \mathbf{W}^k, \mathbf{b}^k, \mathbf{z}^k, \mathbf{q}^k, \mathbf{u}^k)$ is convergent, then $\lim_{k\to\infty} \|\mathbf{p}^{k+1} - \mathbf{p}^k\|_2^2 = 0$, $\lim_{k\to\infty} \|\mathbf{W}^{k+1} - \mathbf{W}^k\|_2^2 = 0$, $\lim_{k\to\infty} \|\mathbf{b}^{k+1} - \mathbf{b}^k\|_2^2 = 0$, $\lim_{k\to\infty} \|\mathbf{z}^{k+1} - \mathbf{z}^k\|_2^2 = 0$, and $\lim_{k\to\infty} \|\mathbf{q}^{k+1} - \mathbf{q}^k\|_2^2 = 0$. $\lim_{k\to\infty} \|\mathbf{u}^{k+1} - \mathbf{u}^k\|_2^2 = 0$ is derived from Lemma 5 in the Appendix. □

The third lemma guarantees that the subgradient of the objective is upper bounded, which is stated as follows:

Lemma 3 (Bounded Subgradient)**.** *There exists a constant* $C > 0$ *and* $g^{k+1} \in \partial L_\rho(\boldsymbol{p}^{k+1}, \boldsymbol{W}^{k+1}, \boldsymbol{b}^{k+1}, \boldsymbol{z}^{k+1}, \boldsymbol{q}^{k+1}, \boldsymbol{u}^{k+1})$ *such that*

$$\|g^{k+1}\| \leq C(\|\boldsymbol{p}^{k+1} - \boldsymbol{p}^k\| + \|\boldsymbol{W}^{k+1} - \boldsymbol{W}^k\| + \|\boldsymbol{b}^{k+1} - \boldsymbol{b}^k\| + \|\boldsymbol{z}^{k+1} - \boldsymbol{z}^k\| + \|\boldsymbol{q}^{k+1} - \boldsymbol{q}^k\| + \|\boldsymbol{u}^{k+1} - \boldsymbol{u}^k\|)$$

Now based on Theorem 1, and Lemma 3, the convergence of the pdADMM algorithm to a critical point is presented in the following theorem.

Theorem 2 (Convergence to a Critical Point)**.** *If* $\rho > \max(4\nu S^2, (\sqrt{17}+1)\nu/2)$*, then for the variables* $(\boldsymbol{p}, \boldsymbol{W}, \boldsymbol{b}, \boldsymbol{z}, \boldsymbol{q}, \boldsymbol{u})$ *in Problem 2, starting from any* $(\boldsymbol{p}^0, \boldsymbol{W}^0, \boldsymbol{b}^0, \boldsymbol{z}^0, \boldsymbol{q}^0, \boldsymbol{u}^0)$*,* $(\boldsymbol{p}^k, \boldsymbol{W}^k, \boldsymbol{b}^k, \boldsymbol{z}^k, \boldsymbol{q}^k, \boldsymbol{u}^k)$ *has at least a limit point* $(\boldsymbol{p}^*, \boldsymbol{W}^*, \boldsymbol{b}^*, \boldsymbol{z}^*, \boldsymbol{q}^*, \boldsymbol{u}^*)$*, and any limit point is a critical point of Problem 2. That is,* $0 \in \partial L_\rho(\boldsymbol{p}^*, \boldsymbol{W}^*, \boldsymbol{b}^*, \boldsymbol{z}^*, \boldsymbol{q}^*, \boldsymbol{u}^*)$*. In other words,*

$$\begin{aligned}
&p_{l+1}^* = q_l^*,\ \nabla_{\boldsymbol{p}^*} L_\rho(\boldsymbol{p}^*, \boldsymbol{W}^*, \boldsymbol{b}^*, \boldsymbol{z}^*, \boldsymbol{q}^*, \boldsymbol{u}^*) = 0,\\
&\nabla_{\boldsymbol{W}^*} L_\rho(\boldsymbol{p}^*, \boldsymbol{W}^*, \boldsymbol{b}^*, \boldsymbol{z}^*, \boldsymbol{q}^*, \boldsymbol{u}^*) = 0,\\
&\nabla_{\boldsymbol{b}^*} L_\rho(\boldsymbol{p}^*, \boldsymbol{W}^*, \boldsymbol{b}^*, \boldsymbol{z}^*, \boldsymbol{q}^*, \boldsymbol{u}^*) = 0,\\
&0 \in \partial_{\boldsymbol{z}^*} L_\rho(\boldsymbol{p}^*, \boldsymbol{W}^*, \boldsymbol{b}^*, \boldsymbol{z}^*, \boldsymbol{q}^*, \boldsymbol{u}^*),\\
&\nabla_{\boldsymbol{q}^*} L_\rho(\boldsymbol{p}^*, \boldsymbol{W}^*, \boldsymbol{b}^*, \boldsymbol{z}^*, \boldsymbol{q}^*, \boldsymbol{u}^*) = 0.
\end{aligned}$$

Proof. From Lemma 2, $(\mathbf{p}^k, \mathbf{W}^k, \mathbf{b}^k, \mathbf{z}^k, \mathbf{q}^k, \mathbf{u}^k)$ has at least a limit point $(\mathbf{p}^*, \mathbf{W}^*, \mathbf{b}^*, \mathbf{z}^*, \mathbf{q}^*, \mathbf{u}^*)$ because a bounded sequence has at least a limit point. From Lemma 3 and Theorem 1, $\|g^{k+1}\| \to 0$ as $k \to \infty$. According to the definition of general subgradient (Defintion 8.3 in [42]), we have $0 \in \partial L_\rho(\mathbf{p}^*, \mathbf{W}^*, \mathbf{b}^*, \mathbf{z}^*, \mathbf{q}^*, \mathbf{u}^*)$. In other words, every limit point $(\mathbf{p}^*, \mathbf{W}^*, \mathbf{b}^*, \mathbf{z}^*, \mathbf{q}^*, \mathbf{u}^*)$ is a critical point. □

Theorem 2 shows that our proposed pdADMM algorithm converges for sufficiently large ρ, which is consistent with previous literature [2]. Next, the following theorem ensures the sublinear convergence rate $o(1/k)$ of the proposed pdADMM algorithm, whose proof is at the end of this paper.

Theorem 3 (Convergence Rate)**.** *For a sequence* $(\boldsymbol{p}^k, \boldsymbol{W}^k, \boldsymbol{b}^k, \boldsymbol{z}^k, \boldsymbol{q}^k, \boldsymbol{u}^k)$*, define* $c_k = \min_{0 \leq i \leq k}(\sum_{l=2}^{L}(\tau_l^{i+1}/2)\|p_l^{i+1} - p_l^i\|_2^2 + \sum_{l=1}^{L}(\theta_l^{i+1}/2)\|W_l^{i+1} - W_l^i\|_2^2 + \sum_{l=1}^{L}(\nu/2)\|b_l^{i+1} - b_l^i\|_2^2 + \sum_{l=1}^{L-1} C_1\|z_l^{i+1} - z_l^i\|_2^2 + (\nu/2)\|z_L^{i+1} - z_L^i\|_2^2 + \sum_{l=1}^{L-1} C_2\|q_l^{i+1} - q_l^i\|_2^2)$ *where* $C_1 = \nu/2 - 2\nu^2 S^2/\rho > 0$ *and* $C_2 = \rho/2 - 2\nu^2/\rho - \nu/2 > 0$*, then the convergence rate of* c_k *is* $o(1/k)$*.*

V. Experiments

In this section, we evaluate the performance of the proposed pdADMM using six benchmark datasets. Speedup, convergence and accuracy performance are compared with several state-of-the-art optimizers. All experiments were conducted on 64-bit machine with Intel Xeon(R) silver 4114 Processor and 48GB RAM.

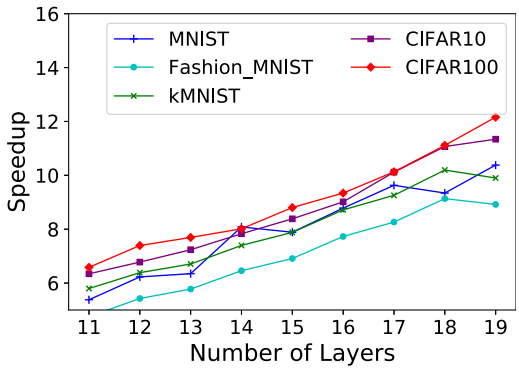


Fig. 3: The relationship between speedup and the number of layers: the speedup increases linearly with the number of layers.

A. Datasets

In this experiment, six benchmark datasets were used for performance evaluation:

1. MNIST [43]. The MNIST dataset has ten classes of handwritten-digit images, which was firstly introduced by Lecun et al. in 1998 [43]. It contains 55,000 training samples and 10,000 test samples with 196 features each, which is provided by the Keras library [44].
2. Fashion MNIST [45]. The Fashion MNIST dataset has ten classes of assortment images on the website of Zalando, which is Europes largest online fashion platform [45]. The Fashion-MNIST dataset consists of 60,000 training samples and 10,000 test samples with 196 features each.
3. kMNIST(Kuzushiji-MNIST) [46]. The kMNIST dataset has ten classes, each of which is a character to represent each of the 10 rows of Hiragana. The kMNIST dataset consists of 60,000 training samples and 10,000 test samples with 196 features each.
4. SVHN (Street View House Numbers) [47]. The SVHN dataset is obtained from house numbers in Google Street View images. It consists of ten classes of digits. In our experiments, we use three classes '0', '1' and '2'. The number of training data and test data are 24,446 and 9,248, respectively, with 768 features each.
5. CIFAR10 [48]. CIFAR10 is a collection of color images with 10 different classes. In our experiments, we use two classes '0' and '6'. The number of training data and test data are 12,000 and 2,000, respectively, with 768 features each.
6. CIFAR100 [48]. CIFAR100 is similar to CIFAR10 except that CIFAR100 has 100 classes. In our experiments, we use two classes '0' and '2'. The number of training data and test data are 5,000 and 1,000, respectively, with 768 features each.

B. Speedup

In this experiment, we investigate the speedup of the proposed pdADMM algorithm concerning the number of layers and the number of neurons on the large-scale deep neural networks. The activation function was set to the Rectified linear unit (Relu). The loss function was the cross-entropy loss. The running time per epoch was the average of 10 epochs. ρ

MNIST dataset			
Neurons#	Serial pdADMM (sec)	pdADMM(sec)	Speedup
1500	237.66	26.78	8.87
1600	348.70	31.78	10.97
1700	390.51	35.79	10.91
1800	475.60	41.37	11.50
1900	465.57	45.87	10.15
2000	570.90	50.70	11.26
2000	570.9	50.7	11.26
2100	570	54.91	10.38
2200	678.83	63.59	10.68
2300	710.3	70.36	10.10
2400	766.82	62.5	12.27
Fashion MNIST dataset			
Neurons#	Serial pdADMM (sec)	pdADMM(sec)	Speedup
1500	358.68	32.65	10.99
1600	407.71	37.90	10.76
1700	476.79	44.75	10.65
1800	539.51	50.50	10.68
1900	599.42	53.88	11.13
2000	645.87	58.68	11.01
2100	740.39	67.91	10.90
2200	818.58	74.17	11.03
kMNIST dataset			
Neurons#	Serial pdADMM (sec)	pdADMM(sec)	Speedup
1500	354.85	32.65	10.87
1600	407.73	37.11	10.99
1700	472.4648	42.58	11.10
1800	539.52	48.78	11.06
1900	596.84	55.56	10.74
2000	660.58	56.10	11.78
2100	737.78	66.95	11.02
2200	806.74	76.16	10.59
CIFAR10 dataset			
Neurons#	Serial pdADMM (sec)	pdADMM(sec)	Speedup
1500	326.62	25.00	13.06
1600	374.82	28.96	12.94
1700	433.46	33.99	12.75
1800	485.86	38.66	12.57
1900	544.11	43.10	12.62
2000	572.33	46.90	12.20
2100	602.65	55.25	10.91
2200	732.79	59.27	12.36
2300	784.87	56.26	13.95
2400	854.47	63.1	13.54
CIFAR100 dataset			
Neurons#	Serial pdADMM (sec)	pdADMM(sec)	Speedup
1500	334.55	25.39	13.18
1600	382.24	29.3	13.05
1700	445.23	34	13.09
1800	500.00	38.38	13.03
1900	549.77	43.25	12.71
2000	576.10	42.47	13.56
2100	666.06	47.43	14.04
2200	735.63	52.41	14.04
2300	793.03	56.73	13.98
2400	857.41	62.3	13.76

TABLE II: The relation between speedup and number of neurons on the MNIST, Fashion MNIST datasets, kMNIST, CIFAR10 and CIFAR100 datasets: the pdADMM runs 10 times faster than its serial version.

and ν were both set to 10^{-4}.

Firstly, we investigated the relationship between speedup and the number of layers. We set up a feed-forward neural network with different number of hidden layers, which ranges from 11 to 19. The number of neurons in each layer was fixed to 2,400. The SVHN dataset was not tried due to memory issues. Figure 3 shows that the speedup increases linearly with the number of layers. Specifically, the speedup reached 11 when 19 hidden layers were trained. This indicates that the deeper a neural network is, the more speedup our proposed pdADMM can gain.

Secondly, the relationship between speedup and the number of neurons was studied. Specifically, we test our proposed pdADMM algorithm on a feed-forward neural network with 19 hidden layers. The number of neurons in each layer ranges from 1,500 to 2,400. The speedup was shown in Table II on the MNIST and Fashion MNIST datasets. Specifically, the speedup remains stable around 10 no matter how many neurons were trained. This concludes that the speedup of the proposed pdADMM is independent of the number of neurons.

C. Convergence

To validate the convergence of the proposed pdADMM, we set up a feed-forward neural network with 9 hidden layers, each of which has 500 neurons. The Rectified linear unit (ReLU) was used for the activation function for both network structures. The loss function was set as the cross-entropy loss. The number of epoch was set to 100. ν and ρ was both set to 0.1. As shown in Figure 4, the objective keeps decreasing monotonically on all six datasets, and the residual converges sublinearly to 0, which are consistent with Theorems 2 and 3.

D. Accuracy

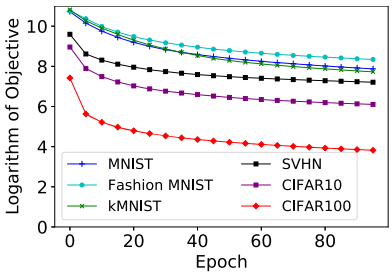
1) Experimental Settings: In order to evaluate accuracy, we used the same architecture as the previous section. ν and ρ were set to 10^{-4} in order to maximize the performance of training data. The number of epoch was set to 100. In this experiment, the full batch was used for training. As suggested by the training strategy in Section III-B, we firstly trained a feed-forward neural network with five hidden layers, and then all layers were involved in training.

2) Comparison Methods: SGD and its variants are state-of-the-art methods and hence were served as comparison methods. For SGD-based methods, the full batch dataset is used for training models. All parameters were chosen by maximizing the accuracy of training datasets. The baselines are described as follows:

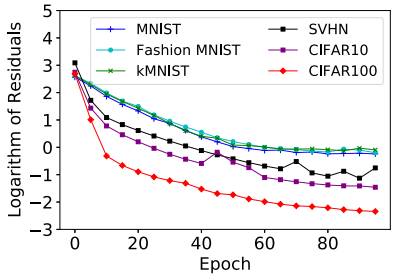
1. Gradient Descent (GD) [49]. The GD and its variants are the most popular deep learning optimizers, whose convergence has been studied extensively in the literature. The learning rate of GD was set to 0.01.

2. Adaptive learning rate method (Adadelta) [50]. The Adadelta is proposed to overcome the sensitivity to hyper-parameter selection. The learning rate of Adadelta was set to 1.

3. Adaptive gradient algorithm (Adagrad) [51]. Adagrad is

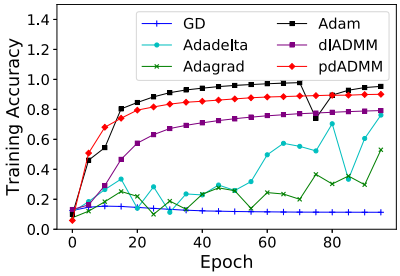


(a). Objective versus epoch

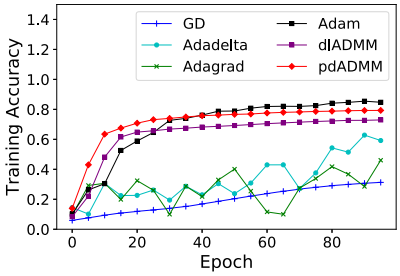


(b). Residual versus epoch

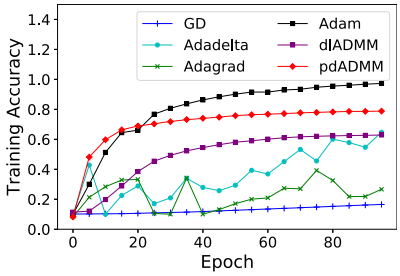
Fig. 4: The convergence of the proposed pdADMM: the objective decreases monotonously, and the residual converges to 0.



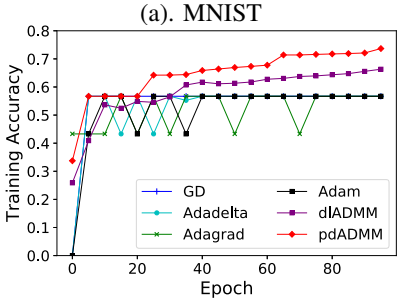
(a). MNIST



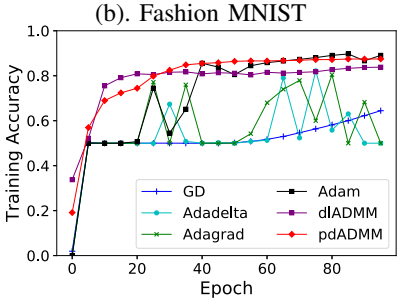
(b). Fashion MNIST



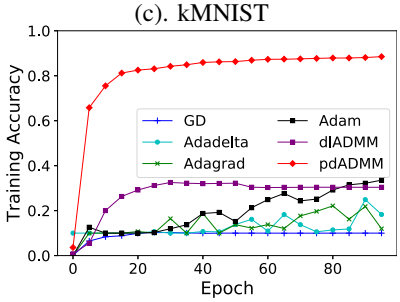
(c). kMNIST



(d). SVHN

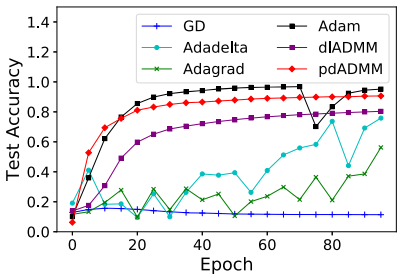


(e). CIFAR10

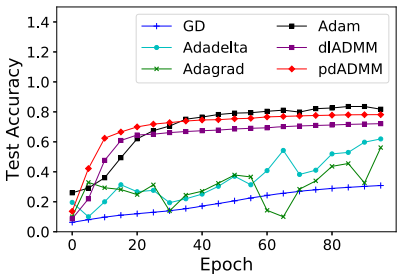


(f). CIFAR100

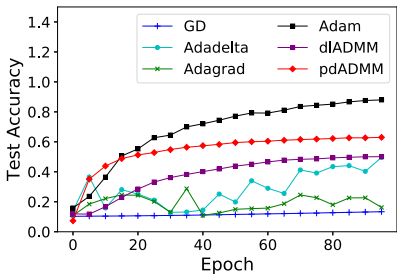
Fig. 5: Training Accuracy of all methods: pdADMM outperformed most comparison methods; SGD-type methods suffer from gradient vanishing problems.



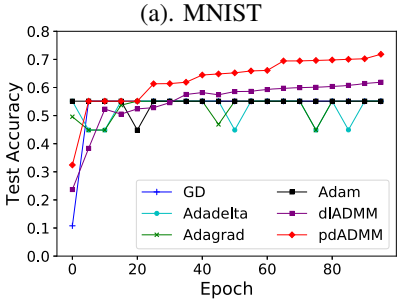
(a). MNIST



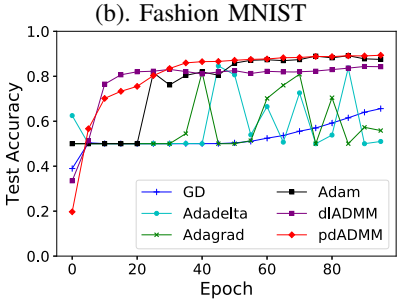
(b). Fashion MNIST



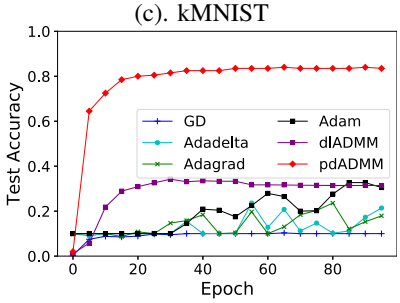
(c). kMNIST



(d). SVHN



(e). CIFAR10



(f). CIFAR100

Fig. 6: Test Accuracy of all methods: pdADMM outperformed most comparison methods.

an improved version of SGD: rather than fixing the learning rate during training, it adapts the learning rate to the hyper-parameter. The learning rate of Adagrad was set to 0.1.

4. Adaptive momentum estimation (Adam) [52]. Adam is the most popular optimization method for deep learning models. It estimates the first and second momentum to correct the biased gradient and thus makes convergence fast. The learning rate of Adam was set to 0.001.

5. Deep learning Alternaing Direction Method of Multipliers (dlADMM) [2]. The dlADMM is an improvement of the previous ADMM implementation [1]. It is guaranteed to converge to a critical point with a rate of $o(1/k)$. ρ and ν were both set to 10^{-6}.

3) Performance: In this section, the performance of the proposed pdADMM is analyzed against comparison methods. Figures 5 and 6 show the training and test accuracy of the proposed pdADMM against comparison methods on six datasets, respectively. X-axis and Y-axis represent epoch and training accuracy, respectively. Overall, the pdADMM outperformed most of comparison methods: it performed the best on the SVHN, CIFAR10 and CIFAR100 datasets, while was only secondary to Adam on the MNIST, Fashion MNIST and kMNIST datasets. The performance gap is particularly obvious on the CIFAR100 dataset. Most SGD-type methods suffered from gradient vanishing in the deep neural network, and struggled to find an optimum: for example, GD can only reach 10% training accuracy on the MNIST dataset; Adagrad and Adadelta performed better then GD, but took many epochs to escape saddle points. The dlADMM can avoid gradient vanishing problem, however, it performed worse than the proposed pdADMM on all datasets. Adam performed the best on three MNIST-like datasets, but performed worse than the proposed pdADMM on three other datasets, which are hard to train with high accuracy than three MNIST-like datasets. This indicates that the proposed pdADMM may be more suitable for training hard datasets then Adam.

VI. Conclusion

Alternating Direction Method of Multipliers (ADMM) is considered to be a good alternative to Stochastic gradient descent (SGD) for training deep neural networks. In this paper, we propose a novel parallel deep learning Alternating Direction Method of Multipliers (pdADMM) to achieve layer parallelism. The proposed pdADMM is guaranteed to converge to a critical solution under mild conditions. Experiments on benchmark datasets demonstrate that our proposed pdADMM can lead to a huge speedup when training a deep feed-forward neural network, it also outperformed others on six benchmark datasets.

Acknowledgement

This work was supported by the National Science Foundation (NSF) Grant No. 1755850, No. 1841520, No. 2007716, No. 2007976, No. 1942594, No. 1907805, a Jeffress Memorial Trust Award, NVIDIA GPU Grant, and Design Knowledge Company (subcontract number: 10827.002.120.04).

References

[1] 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*, 2016, pp. 2722–2731.

[2] J. Wang, F. Yu, X. Chen, and L. Zhao, "Admm for efficient deep learning with global convergence," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 111–119.

[3] 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.

[4] J. F. Mota, J. M. Xavier, P. M. Aguiar, and M. Püschel, "Distributed admm for model predictive control and congestion control," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 5110–5115.

[5] T.-H. Chang, "A proximal dual consensus admm method for multi-agent constrained optimization," *IEEE Transactions on Signal Processing*, vol. 64, no. 14, pp. 3719–3734, 2016.

[6] T.-H. Chang, M. Hong, and X. Wang, "Multi-agent distributed optimization via inexact consensus admm," *IEEE Transactions on Signal Processing*, vol. 63, no. 2, pp. 482–497, 2014.

[7] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, "On the linear convergence of the admm in decentralized consensus optimization," *IEEE Transactions on Signal Processing*, vol. 62, no. 7, pp. 1750–1761, 2014.

[8] Z. Xu, G. Taylor, H. Li, M. A. Figueiredo, X. Yuan, and T. Goldstein, "Adaptive consensus admm for distributed optimization," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3841–3850.

[9] S. Zhu, M. Hong, and B. Chen, "Quantized consensus admm for multi-agent distributed optimization," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 4134–4138.

[10] R. Zhang and J. Kwok, "Asynchronous distributed admm for consensus optimization," in *International Conference on Machine Learning*, 2014, pp. 1701–1709.

[11] E. Wei and A. Ozdaglar, "Distributed alternating direction method of multipliers," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 5445–5450.

[12] T.-H. Chang, M. Hong, W.-C. Liao, and X. Wang, "Asynchronous distributed admm for large-scale optimizationpart i: Algorithm and convergence analysis," *IEEE Transactions on Signal Processing*, vol. 64, no. 12, pp. 3118–3130, 2016.

[13] T.-H. Chang, W.-C. Liao, M. Hong, and X. Wang, "Asynchronous distributed admm for large-scale optimizationpart ii: Linear convergence analysis and numerical performance," *IEEE Transactions on Signal Processing*, vol. 64, no. 12, pp. 3131–3144, 2016.

[14] M. Hong, "A distributed, asynchronous, and incremental algorithm for nonconvex optimization: an admm approach," *IEEE Transactions on Control of Network Systems*, vol. 5, no. 3, pp. 935–945, 2017.

[15] S. Kumar, R. Jain, and K. Rajawat, "Asynchronous optimization over heterogeneous networks via consensus admm," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 3, no. 1, pp. 114–129, 2016.

[16] S. Magnússon, P. C. Weeraddana, M. G. Rabbat, and C. Fischione, "On the convergence of alternating direction lagrangian methods for nonconvex structured optimization problems," *IEEE Transactions on Control of Network Systems*, vol. 3, no. 3, pp. 296–309, 2015.

[17] G. Li and T. K. Pong, "Global convergence of splitting methods for nonconvex composite optimization," *SIAM Journal on Optimization*, vol. 25, no. 4, pp. 2434–2460, 2015.

[18] Y. Wang, W. Yin, and J. Zeng, "Global convergence of admm in nonconvex nonsmooth optimization," *Journal of Scientific Computing*, pp. 1–35, 2015.

[19] M. Hong, Z.-Q. Luo, and M. Razaviyayn, "Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems," *SIAM Journal on Optimization*, vol. 26, no. 1, pp. 337–364, 2016.

[20] J. Wang, L. Zhao, and L. Wu, "Multi-convex inequality-constrained alternating direction method of multipliers," *arXiv preprint arXiv:1902.10882*, 2019.

[21] Q. Liu, X. Shen, and Y. Gu, "Linearized admm for nonconvex nonsmooth optimization with convergence analysis," *IEEE Access*, vol. 7, pp. 76 131–76 144, 2019.

[22] J. Wang and L. Zhao, "Nonconvex generalization of admm for nonlinear equality constrained problems," *arXiv preprint arXiv:1705.03412*, 2017.

[23] X. Xie, J. Wu, G. Liu, Z. Zhong, and Z. Lin, "Differentiable linearized admm," in *International Conference on Machine Learning*, 2019, pp. 6902–6911.

[24] J. Wang and L. Zhao, "Accelerated gradient-free neural network training by multi-convex alternating optimization," 2019.

[25] R. Chartrand and B. Wohlberg, "A nonconvex admm algorithm for group sparsity with sparse groups," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 6009–6013.

[26] D. Hajinezhad and M. Hong, "Nonconvex alternating direction method of multipliers for distributed sparse principal component analysis," in *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2015, pp. 255–259.

[27] K. Guo, D. Han, D. Z. Wang, and T. Wu, "Convergence of admm for multi-block nonconvex separable optimization models," *Frontiers of Mathematics in China*, vol. 12, no. 5, pp. 1139–1162, 2017.

[28] A. Themelis and P. Patrinos, "Douglas–rachford splitting and admm for nonconvex optimization: Tight convergence results," *SIAM Journal on Optimization*, vol. 30, no. 1, pp. 149–181, 2020.

[29] J. Wang, Z. Chai, Y. Chen, and L. Zhao, "Tunable subnetwork splitting for model-parallelism of neural network training," in *ICML 2020 Workshop: Beyond First Order Methods in Machine Learning*, 2020.

[30] J. Wang and L. Zhao, "The application of multi-block admm on isotonic regression problems," *arXiv preprint arXiv:1903.01054*, 2019.

[31] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Advances in neural information processing systems*, 2017, pp. 1509–1519.

[32] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," 2018.

[33] B. C. Ooi, K.-L. Tan, S. Wang, W. Wang, Q. Cai, G. Chen, J. Gao, Z. Luo, A. K. Tung, Y. Wang *et al.*, "Singa: A distributed deep learning platform," in *Proceedings of the 23rd ACM international conference on Multimedia*. ACM, 2015, pp. 685–688.

[34] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015.

[35] S. H. Hashemi, S. A. Jyothi, and R. H. Campbell, "Tictac: Accelerating distributed deep learning with communication scheduling," in *Proceedings of the 2nd SysML Conference*, 2019.

[36] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on {GPU} clusters," in *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, 2017, pp. 181–193.

[37] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems*, 2010, pp. 2595–2603.

[38] P. Parpas and C. Muir, "Predict globally, correct locally: Parallel-in-time optimal control of neural networks," 2019.

[39] Z. Huo, B. Gu, and H. Huang, "Training neural networks using features replay," in *Advances in Neural Information Processing Systems*, 2018, pp. 6659–6668.

[40] H. Zhuang, Y. Wang, Q. Liu, and Z. Lin, "Fully decoupled neural network learning using delayed gradients," 2019.

[41] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM journal on imaging sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[42] R. T. Rockafellar and R. J.-B. Wets, *Variational analysis*. Springer Science & Business Media, 2009, vol. 317.

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

[44] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[45] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.

[46] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," in *NeurIPS 2018 Workshop on Machine Learning for Creativity and Design*, 2018.

[47] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[48] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[49] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[50] M. D. Zeiler, "Adadelta: an adaptive learning rate method," 2012.

[51] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[52] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[53] W. Deng, M.-J. Lai, Z. Peng, and W. Yin, "Parallel multi-block admm with o (1/k) convergence," *Journal of Scientific Computing*, vol. 71, no. 2, pp. 712–736, 2017.

Proof of Theorem 3

Proof. To prove this theorem, we will first show that c_k satisfies two conditions: (1). $c_k \geq c_{k+1}$. (2). $\sum_{k=0}^{\infty} c_k$ is bounded. Specifically, first, we have

$$
\begin{aligned}
c_k =& \min_{0\leq i\leq k}(\sum_{l=2}^{L}(\tau_l^{i+1}/2)\|p_l^{i+1}-p_l^i\|_2^2 \\
&+\sum_{l=1}^{L}(\theta_l^{i+1}/2)\|W_l^{i+1}-W_l^i\|_2^2+\sum_{l=1}^{L}(\nu/2)\|b_l^{i+1}-b_l^i\|_2^2 \\
&+\sum_{l=1}^{L-1}C_1\|z_l^{i+1}-z_l^i\|_2^2+(\nu/2)\|z_L^{i+1}-z_L^i\|_2^2 \\
&+\sum_{l=1}^{L-1}C_2\|q_l^{i+1}-q_l^i\|_2^2) \\
\geq& \min_{0\leq i\leq k+1}(\sum_{l=2}^{L}(\tau_l^{i+1}/2)\|p_l^{i+1}-p_l^i\|_2^2 \\
&+\sum_{l=1}^{L}(\theta_l^{i+1}/2)\|W_l^{i+1}-W_l^i\|_2^2+\sum_{l=1}^{L}(\nu/2)\|b_l^{i+1}-b_l^i\|_2^2 \\
&+\sum_{l=1}^{L-1}C_1\|z_l^{i+1}-z_l^i\|_2^2+(\nu/2)\|z_L^{i+1}-z_L^i\|_2^2 \\
&+\sum_{l=1}^{L-1}C_2\|q_l^{i+1}-q_l^i\|_2^2) \\
=& c_{k+1}
\end{aligned}
$$

Therefore c_k satisfies the first condition. Second,

$$
\begin{aligned}
&\sum_{k=0}^{\infty}c_k=\sum_{k=0}^{\infty}\min_{0\leq i\leq k}(\sum_{l=2}^{L}(\tau_l^{i+1}/2)\|p_l^{i+1}-p_l^i\|_2^2 \\
&+\sum_{l=1}^{L}(\theta_l^{i+1}/2)\|W_l^{i+1}-W_l^i\|_2^2+\sum_{l=1}^{L}(\nu/2)\|b_l^{i+1}-b_l^i\|_2^2 \\
&+\sum_{l=1}^{L-1}C_1\|z_l^{i+1}-z_l^i\|_2^2+(\nu/2)\|z_L^{i+1}-z_L^i\|_2^2 \\
&+\sum_{l=1}^{L-1}C_2\|q_l^{i+1}-q_l^i\|_2^2) \\
&\leq\sum_{k=0}^{\infty}(\sum_{l=2}^{L}(\tau_l^{k+1}/2)\|p_l^{k+1}-p_l^k\|_2^2 \\
&+\sum_{l=1}^{L}(\theta_l^{k+1}/2)\|W_l^{k+1}-W_l^k\|_2^2+\sum_{l=1}^{L}(\nu/2)\|b_l^{k+1}-b_l^k\|_2^2 \\
&+\sum_{l=1}^{L-1}C_1\|z_l^{k+1}-z_l^k\|_2^2+(\nu/2)\|z_L^{k+1}-z_L^k\|_2^2 \\
&+\sum_{l=1}^{L-1}C_2\|q_l^{k+1}-q_l^k\|_2^2) \\
&\leq L_\rho(\mathbf{p}^0,\mathbf{W}^0,\mathbf{b}^0,\mathbf{z}^0,\mathbf{q}^0,\mathbf{u}^0)-L_\rho(\mathbf{p}^*,\mathbf{W}^*,\mathbf{b}^*,\mathbf{z}^*,\mathbf{q}^*,\mathbf{u}^*) \\
&\text{(Lemma 1)}
\end{aligned}
$$

So c_k satisfies the second condition. Finally, since we have proved the first two conditions and the third one $c_k \geq 0$ is obvious, the convergence rate of $o(1/k)$ is proven (Lemma 1.2 in [53]). □