Training Neural Networks by Lifted Proximal Operator Machines

Jia Li, Mingqing Xiao, Cong Fang, Yue Dai, Chao Xu, and Zhouchen Lin, *Fellow, IEEE*

Abstract—We present the lifted proximal operator machine (LPOM) to train fully-connected feed-forward neural networks. LPOM represents the activation function as an equivalent proximal operator and adds the proximal operators to the objective function of a network as penalties. LPOM is block multi-convex in all layer-wise weights and activations. This allows us to develop a new block coordinate descent (BCD) method with convergence guarantee to solve it. Due to the novel formulation and solving method, LPOM only uses the activation function itself and does not require any gradient steps. Thus it avoids the gradient vanishing or exploding issues, which are often blamed in gradient-based methods. Also, it can handle various non-decreasing Lipschitz continuous activation functions. Additionally, LPOM is almost as memory-efficient as stochastic gradient descent and its parameter tuning is relatively easy. We further implement and analyze the parallel solution of LPOM. We first propose a general asynchronous-parallel BCD method with convergence guarantee. Then we use it to solve LPOM, resulting in asynchronous-parallel LPOM. For faster speed, we develop the synchronous-parallel LPOM. We validate the advantages of LPOM on various network architectures and datasets. We also apply synchronous-parallel LPOM to autoencoder training and demonstrate its fast convergence and superior performance.

Index Terms—Neural networks, lifted proximal operator machines, block multi-convex, block coordinate descent, parallel implementation.

1 Introduction

Neural networks (NNs) are powerful models that have achieved great success in many applications including image recognition [1], speech recognition [2], natural language understanding [3], and building the Go game learning system [4]. However, since their objective functions are highly non-convex, the training of NNs remains a challenge, which includes the ill-conditioned Hessian and the existence of many saddle points and local minima [5].

As the predominant method in NN training, the gradient-based methods mainly include the vanilla stochastic gradient descent (SGD) [6] and SGD with adaptive learning rates and momentum terms like Nesterov momentum [7], AdaGrad [8], RMSProp [9], Adam [10], and AMSGrad [11]. SGD and its variants use a small mini-batch training samples to estimate the full-batch gradient. So the weight update of each step is simple. Moreover, the estimated gradients have noise. This helps to escape saddle points [12]. Despite the great success of gradient-based methods, they have several critical drawbacks as well. The major flaw is that they suffer from the vanishing or exploding gradient issue, which slows down or destabilizes the training. Some approaches have been developed to remit such an issue, e.g., the introduction of rectified linear units (ReLUs), batch normalization (BN) [13], or residual networks (ResNets) [14]. However, the problem of computing the gradients of a nested objective function still persists. Also, their parameter tuning is difficult, e.g., setting the learning rates and stopping criteria [15]. Furthermore, they cannot handle non-differentiable activation functions directly (e.g., binarized neural networks [16]) and cannot update the weights across layers in parallel [15]. For more details on the limitations of SGD, we refer the readers to [17] and [15].

Due to the limitations of SGD discussed above, there is very active research in developing new training methods for NNs. A notable approach is to introduce auxiliary variables associated with the network activations and formulate the training of an NN as an equality constrained optimization problem [18]. Then this approach solves the resulted constrained problem with standard optimization methods. Another advantage of this approach is that it may be implemented in parallel, and thus may be capable of solving distributed large-scale problems. Several methods of this kind of approach have been proposed and they differ in how to relax the constraints and add penalties to the objective function. Carreira-Perpinan and Wang [18] used quadratic penalties to approximately enforce the equality constraints and solved a series of unconstrained minimization problems. Zeng et al. [19] also used quadratic penalties to approximately enforce the equality constraints but introduced one more block of auxiliary variables for each layer. Motivated by alternating direction method of multipliers

- *J. Li is with the Key Laboratory of Machine Perception (MOE), School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, P.R. China, and the School of Artificial Intelligence, Beijing Normal University, Beijing 100875, P.R. China. E-mail: jiali.gm@gmail.com.*
- *M. Xiao, C. Xu, and Z. Lin are with the Key Laboratory of Machine Perception (MOE), School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, P.R. China. E-mail: mingqing_xiao@pku.edu.cn; xuchao@cis.pku.edu.cn; zlin@pku.edu.cn (Corresponding author: Zhouchen Lin).*
- *C. Fang is with University of Pennsylvania, USA. E-mail: fangcong@pku.edu.cn.*
- *Y. Dai is with the College of Software, Beihang University, Beijing 100083, P.R. China. E-mail: daiyue@buaa.edu.cn.*

Manuscript received April 19, 2005; revised August 26, 2015.

TABLE 1
The property summary of LPOM as compared to SGD. * denotes that it has the vanishing or exploding gradient issues, which has to be addressed by extra empirical tricks.

Property	SGD	LPOM
Memory	Equal	
Gradient	Yes*	No
Parallelizability	No	Yes
Parameter tuning	Hard	Easy
Activation function	Differentiable and Non-saturating	Can be non-differentiable or saturating

(ADMM) [20], Taylor et al. [17] and Zhang et al. [21] adopted the augmented Lagrangian approach to obtain a sequence satisfying the nonlinear equality constraints. However, the introduced Lagrange multipliers require more memory and ADMM is not designed to deal with nonlinear equality constraints. Zhang and Brand [22] and Askari et al. [23] used the equivalent representations of the activation functions, which eliminate the nonlinear constraints. However, the method in [22] is limited to the ReLU activation function. While the formulation in [23] can handle general activation functions, its solving method needs to be tailored for each activation function and the presented solving method is for ReLU only. Also, the two methods cannot compete with SGD in speed or error rate. Moreover, in the test phase, they needed to solve optimization problems to predict the labels of new samples, which is computationally unaffordable. In comparison, SGD uses feed-forward passes to predict labels, i.e., propagating activations from the input to the output layer and making the prediction using the activation of the output layer. Recently, Gu et al. [24] presented a follow-up work of [23]. Their formulation (see (4) and (9) in [24]) is equivalent to ours (see (19) or (20)). However, their solving method is not as efficient and general as ours (see the last paragraph of Subsection 4.1).

The main contributions of this paper can be summarized as follows.

- We develop a new optimization method to train fully-connected feed-forward NNs, which we call the lifted proximal operator machine (LPOM)[1]. LPOM allows us to infer new samples using simple feed-forward passes, which is the same as SGD. Moreover, LPOM is block multi-convex, i.e., it is convex w.r.t. layer-wise weights or activations while keeping the remaining weights and activations fixed. In contrast, most of the existing NN training methods do not have this property. This is very beneficial for the training of NNs.
- Accordingly, we propose a new block coordinate descent (BCD) method to solve LPOM and prove its convergence. Our BCD solving method only uses the mapping of the activation function itself and does not use its derivative. Thus it avoids the gradient vanishing or exploding problem, which is well-known in gradient-based methods. Also, LPOM is applicable to various non-decreasing Lipschitz continuous activation functions, which can be saturating (e.g., sigmoid and tanh) and non-differentiable (e.g., ReLU and leaky ReLU). Moreover, LPOM needs no more auxiliary variables than layer-wise activations. So it requires almost the same memory as that of SGD, which also stores all activations for gradient calculations. Additionally, it is easy to tune the penalty parameters in LPOM.
- We implement and analyze the parallel solution of LPOM in both asynchronous and synchronous ways. We first propose a general asynchronous-parallel BCD method. We prove its convergence and use it to solve LPOM, resulting in asynchronous-parallel LPOM. The obtained conclusion is also the foundation of synchronous-parallel LPOM. Then we propose synchronous-parallel LPOM and show that it achieves satisfactory speedup over serial LPOM without suffering degradation in performance.

1. We filed a patent on the LPOM formulation in Nov. 2017, which corresponds to Section 3, and filed another patent on the solving method of LPOM in Oct. 2018, which corresponds to Section 4. The preliminary version of this work has been presented at AAAI 2019 [25].

TABLE 2
Summary of the main notations used in this paper.

Notation	Definition
$\mathbf{1}$	All-one column vector
$\mathbf{0}$	All-zero matrix
$\lvert a\rvert$	Absolute value of a scalar a
X^T	Transpose of vector or matrix X
f^{-1}	Inverse of function f
$f'(x)$	Derivative of univariate function f at x
∇f	Gradient of multivariate function f
$X^\dagger$	Pseudo-inverse of matrix X
$\lVert x\rVert_2$	l_2-norm of vector x
$\circ$	Element-wise multiplication
$\lvert X\rvert$	A matrix with the absolute values of X
$\lVert X\rVert_F$	Frobenius norm of matrix X
$\lVert X\rVert_\infty$	l_∞-norm of matrix X
$\lVert X\rVert_1$	l_1-norm of matrix X
$\lVert X\rVert_2$	l_2-norm (spectral norm) of matrix X
$\langle X, Y\rangle$	Inner product of X and Y

Since SGD is the commonly used NN training method, we use it as our main competitor. As shown in Table 1, LPOM exhibits some favorable properties as compared to SGD. Currently, we implement LPOM only on fully-connected NNs. Convolutional neural networks (CNNs) are the most popular feed-forward networks. However, since we have not interpreted pooling operators and skip-connections, we will implement LPOM on CNNs in the future. We note that most of the existing non-gradient based methods also focus on fully-connected NNs first [17], [18], [19], [21], [22], [23].

2 Related Work

In this section, we review some non-gradient based NN training methods, which are most related to our work. We summarize the main notations used throughout the paper in Table 2.

Consider a standard feed-forward NN with n layers for supervised learning, where the first layer is the input layer and the n-th layer is the output layer. Let $X^1 \in \mathbb{R}^{n_1 \times m}$ be a batch of training samples, where n_1 is the dimension of the training samples and also the number of neurons in the input layer, and m is the batch size. Let $D \in \mathbb{R}^{c\times m}$ be the labels of X^1, where c is the number of classes. Let W^i be the weight matrix between the i-th layer and the $(i+1)$-th layer, where we stack an additional column to W^i and an all-one row to X^i and omit the corresponding bias. Let $\phi(\cdot)$ be the element-wise activation function (e.g., sigmoid, tanh, and ReLU). Let $\ell(\cdot,\cdot)$ be the loss function (e.g., mean squared error (MSE) or cross-entropy). With the help of these notations, the batch training problem of the NN can be formulated as the following minimization:

$$\min_{\{W^i\}} \ell\left(\phi(W^{n-1}\phi(\cdots\phi(W^2\phi(W^1X^1))\cdots)), D\right). \tag{1}$$

In the above formulation, the objective function of the NN is nested, where the output of the i-th layer is the input of the $(i+1)$-th layer for $i=1,\cdots,n-1$. Problem (1) can be solved by SGD. It computes the gradients of $\{W^i\}_{i=1}^{n-1}$ using backpropagation and then updates $\{W^i\}_{i=1}^{n-1}$ by gradient descent.

To make problem (1) more computationally tractable, the most common approach is introducing the activation of each layer as a block of auxiliary variables. Then problem (1) can be equivalently rewritten as an equality constrained minimization problem [18]:

$$\begin{aligned}&\min_{\{W^i\},\{X^i\}} \ell(X^n, D)\\ &\text{s.t. } X^i = \phi(W^{i-1}X^{i-1}),\ i=2,3,\cdots,n,\end{aligned} \tag{2}$$

where X^i is the activation of the i-th layer, X^n is also the output of the NN, and the other notations are the same as those in (1). In comparison to problem (1), the objective function of problem (2) is not nested. So we may use more elegant optimization methods to solve it. We want to note that when using SGD to solve problem (1), it also needs to record the activations $\{X^i\}_{i=2}^n$ to compute the gradients.

Carreira-Perpinan and Wang [18] proposed the method of auxiliary coordinates (MAC) to approximately solve problem (2). Rather than directly solving (2), MAC relaxes the equality constraints by using quadratic penalties and solves unconstrained problems of the form:

$$\min_{\{W^i\},\{X^i\}} \ell(X^n, D) + \frac{\mu}{2}\sum_{i=2}^{n} \|X^i - \phi(W^{i-1}X^{i-1})\|_F^2, \tag{3}$$

where $\mu > 0$ gradually increases with the iterations. In order to decouple the nonlinear activations and obtain more efficient solving methods for subproblems, Zeng et al. [19] introduced a new block of auxiliary variables for each layer to problem (2) and suggested the following problem:

$$\begin{aligned}&\min_{\{W^i\},\{X^i\},\{U^i\}} \ell(X^n, D)\\ &\text{s.t. } U^i = W^{i-1}X^{i-1}, X^i = \phi(U^i),\ i=2,3,\cdots,n.\end{aligned} \tag{4}$$

Following the MAC method, instead of directly solving the above, they optimized the following problems of the form:

$$\begin{aligned}&\min_{\{W^i\},\{X^i\},\{U^i\}} \ell(X^n, D)\\ &+\frac{\mu}{2}\sum_{i=2}^{n}(\|U^i - W^{i-1}X^{i-1}\|_F^2 + \|X^i - \phi(U^i)\|_F^2).\end{aligned} \tag{5}$$

Taylor et al. [17] also tried to optimize problem (4). Inspired by ADMM [20], they only added a Lagrange multiplier to the constraint of the output layer, rather than adding a Lagrange multiplier to each equality constraint, which yields

$$\begin{aligned}&\min_{\{W^i\},\{X^i\},\{U^i\},M} \ell(U^n, D)\\ &+\langle U^n, M\rangle + \frac{\beta}{2}\left\|U^n - W^{n-1}X^{n-1}\right\|_F^2\\ &+\sum_{i=2}^{n-1}\frac{\mu_i}{2}(\|U^i - W^{i-1}X^{i-1}\|_F^2 + \|X^i - \phi(U^i)\|_F^2),\end{aligned} \tag{6}$$

where M is the Lagrange multiplier and $\beta > 0$ and $\mu_i > 0$ are constants. Note that the output layer uses the linear activation function (i.e., $\phi(x) = x$). So they only heuristically used the ADMM. Also inspired by ADMM, Zhang et al. [21] used an alternative variable splitting scheme:

$$\begin{aligned}&\min_{\{W^i\},\{X^i\},\{U^i\}} \ell(X^n, D)\\ &\text{s.t. } U^{i-1} = X^{i-1}, X^i = \phi(W^{i-1}U^{i-1}),\ i=2,3,\cdots,n.\end{aligned} \tag{7}$$

As in ADMM, they added a Lagrange multiplier for each constraint in (7). Then the augmented Lagrangian problem becomes:

$$\begin{aligned}&\min_{\{W^i\},\{X^i\},\{U^i\},\{A^i\},\{B^i\}} \ell(X^n, D)\\ &+\frac{\mu}{2}\sum_{i=2}^{n}\left(\left\|U^{i-1} - X^{i-1} + A^{i-1}\right\|_F^2\right.\\ &\qquad\left. + \left\|X^i - \phi(W^{i-1}U^{i-1}) + B^{i-1}\right\|_F^2\right),\end{aligned} \tag{8}$$

where A^i and B^i are the Lagrange multipliers. However, problem (7) contains nonlinear equality constraints and ADMM is designed to handle linearly constrained problems.

Zhang and Brand [22] developed a new approach only to solve problem (2) using the ReLU activation function. Most notably, they represented the ReLU activation function as a convex minimization problem. Namely, they reinterpreted the equality constraint in problem (2) using the ReLU activation function as the following problem:

$$\begin{aligned}X^i &= \phi(W^{i-1}X^{i-1})\\ &= \max(W^{i-1}X^{i-1}, \mathbf{0})\\ &= \operatorname*{argmin}_{U^i \geq \mathbf{0}} \|U^i - W^{i-1}X^{i-1}\|_F^2,\end{aligned} \tag{9}$$

where $\max$ is an element-wise maximum operator. According to the interpretation, they approximated problem (2) using the ReLU activation function as follows:

$$\begin{aligned}&\min_{\{W^i\},\{X^i\}} \ell(X^n, D) + \sum_{i=2}^{n}\frac{\mu_i}{2}\|X^i - W^{i-1}X^{i-1}\|_F^2\\ &\text{s.t. } X^i \geq \mathbf{0},\ i=2,3,\cdots,n.\end{aligned} \tag{10}$$

TABLE 3
The $f(x)$ and $g(x)$ of several commonly used activation functions. Note that $0<\alpha<1$ for the leaky ReLU function and $\alpha>0$ for the exponential linear unit (ELU) function. We use MATLAB's default functions `finverse` and `int` to compute the inverse and indefinite integral of a function, respectively. We only use $\phi(x)$ and do not use $\phi^{-1}(x)$, $f(x)$, and $g(x)$ in our computation.

function	$\phi(x)$	$\phi^{-1}(x)$	$f(x)$	$g(x)$
sigmoid	$\frac{1}{1+e^{-x}}$	$\log\frac{x}{1-x}$ $(0<x<1)$	$\begin{cases} x\log x+(1-x)\log(1-x)-\frac{x^2}{2}, & 0<x<1 \\ +\infty, & \text{otherwise}\end{cases}$	$\log(e^x+1)-\frac{x^2}{2}$
tanh	$\frac{e^x-e^{-x}}{e^x+e^{-x}}$	$\frac{1}{2}\log\frac{1+x}{1-x}$ $(-1<x<1)$	$\begin{cases} \frac{1}{2}[(1-x)\log(1-x) \\ +(1+x)\log(1+x))]-\frac{x^2}{2}, & -1<x<1 \\ +\infty, & \text{otherwise}\end{cases}$	$\log(\frac{e^x+e^{-x}}{2})-\frac{x^2}{2}$
ReLU	$\max(x,0)$	$\begin{cases} x, & x>0 \\ (-\infty,0), & x=0\end{cases}$	$\begin{cases} 0, & x\geq 0 \\ +\infty, & \text{otherwise}\end{cases}$	$\begin{cases} 0, & x\geq 0 \\ -\frac{1}{2}x^2, & x<0\end{cases}$
leaky ReLU	$\begin{cases} x, & x\geq 0 \\ \alpha x, & x<0\end{cases}$	$\begin{cases} x, & x\geq 0 \\ x/\alpha, & x<0\end{cases}$	$\begin{cases} 0, & x\geq 0 \\ \frac{1-\alpha}{2\alpha}x^2, & x<0\end{cases}$	$\begin{cases} 0, & x\geq 0 \\ \frac{\alpha-1}{2}x^2, & x<0\end{cases}$
ELU	$\begin{cases} x, & x\geq 0 \\ \alpha(e^x-1), & x<0\end{cases}$	$\begin{cases} x, & x\geq 0 \\ \log(1+\frac{x}{\alpha}), & -\alpha<x<0\end{cases}$	$\begin{cases} 0, & x\geq 0 \\ (\alpha+x)(\log(\frac{x}{\alpha}+1)-1)-\frac{x^2}{2}, & -\alpha<x<0\end{cases}$	$\begin{cases} 0, & x\geq 0 \\ \alpha(e^x-x)-\frac{x^2}{2}, & x<0\end{cases}$
softplus	$\log(1+e^x)$	$\log(e^x-1)$ $(x>0)$	No analytic expression	No analytic expression

Unlike the MAC and the ADMM based methods, this relaxation does not include nonlinear equality constraints. Moreover, problem (10) is block multi-convex, i.e., it is convex w.r.t. each block of variables when the remaining blocks are fixed. They proposed a new BCD method to solve it. However, in the test time, they had to solve problem (10) with fixed $\{W^i\}_{i=1}^{n-1}$ or retrain W^{n-1} in the output layer. Askari et al. [23] followed the idea proposed by Zhang and Brand [22]. They demonstrated how to convert more general activation functions as convex minimization problems. However, their trained weights can only be used to initialize feed-forward NNs for SGD. Gu et al. [24] presented a follow-up work of [23]. Their formulation is equivalent to that of LPOM. However, their solving method needs to solve constrained problems for updating layer-wise activations, which is not efficient. Also, their solving method needs to be tailored for every activation function, so it is not general. Notably, based on the fact that the composition of convolution and average pooling is also linear, they incorporated an empirical trick to extend their formulation to directly handle convolution and average pooling.

3 Lifted Proximal Operator Machine

In this section, we introduce the idea of LPOM and discuss its advantages over existing NN training methods.

3.1 Reformulation by Proximal Operator

LPOM aims to solve problem (2). The basic idea of LPOM is as follows. Consider a simplified version of problem (2) with only a single constraint:

$$\min_{x,y} s(x), \text{s.t. } x=\phi(y), \tag{11}$$

where x and y are univariate. We first construct a function $h(x,y)$, such that $x=\phi(y)=\operatorname{argmin}_x h(x,y)$. Then we relax problem (11) as the following problem:

$$\min_{x,y} s(x)+\mu h(x,y), \tag{12}$$

where $\mu>0$ is a penalty parameter.

We first describe the construction of $h(x,y)$ for problem (11). The proximal operator [26]

$$\operatorname{prox}_f(y)=\operatorname*{argmin}_x f(x)+\frac{1}{2}(x-y)^2, \tag{13}$$

is a basic operation to update variables in an optimization algorithm. So we consider using it to construct $h(x,y)$. Here we assume that the activation function ϕ is non-decreasing. Then $\phi^{-1}(x)=\{y|x=\phi(y)\}$ is a convex set. $\phi^{-1}(x)$ is a singleton $\{y\}$ iff ϕ is strictly increasing at $\phi(y)$. Define

$$f(x)=\int_0^x(\phi^{-1}(y)-y)dy.$$

Note that $f(x)$ is well defined, if allowed to take value of $+\infty$, even if $\phi^{-1}(y)$ is non-unique for some y between 0 and x. Anyway, we do *not* explicitly use ϕ^{-1}, f, and g (to be defined later) in the computation. It is easy to show that the optimality condition of (13) is $0\in(\phi^{-1}(x)-x)+(x-y)$. So the solution to (13) is exactly $x=\phi(y)$. Hence, we may choose $h(x,y)=f(x)+\frac{1}{2}(x-y)^2$, where $f(x)$ is a univariate function.

Below we consider the original problem (2). We first extend the above $h(x,y)$ to the matrix form $h(X,Y)$, where X and Y are matrices of the same size. Since we already have $h(x,y)$, an element-wise function $h(X,Y)$ is the most desired. Let the subscript kl refer to the (k,l)-th entry of a matrix. We define $h(X,Y)=\sum_k\sum_l h(X_{kl},Y_{kl})$. In order to get a more compact representation of $h(X,Y)$, for a matrix $X=(X_{kl})$, we define $f(X)=(f(X_{kl}))$. Namely, matrix X is an array of entries denoted by X_{kl}. $f(X)$ is an element-wise function on X, i.e., it is a matrix with the same size as X and its (k,l)-th entry is $f(X_{kl})$. Then we have

$$\begin{aligned} h(X,Y) &= \sum\nolimits_k\sum\nolimits_l h\left(X_{kl},Y_{kl}\right) \\ &= \sum\nolimits_k\sum\nolimits_l\left\{f(X_{kl})+\frac{1}{2}\left(X_{kl}-Y_{kl}\right)^2\right\} \\ &= \mathbf{1}^T f(X)\mathbf{1}+\frac{1}{2}\|X-Y\|_F^2. \end{aligned}$$

Accordingly, the matrix form of the proximal operator in (13) for problem (2) becomes:

$$\begin{aligned}&\underset{X^i}{\operatorname{argmin}}\ h(X^i, W^{i-1}X^{i-1})\\ &\qquad\equiv \mathbf{1}^T f(X^i)\mathbf{1} + \frac{1}{2}\|X^i - W^{i-1}X^{i-1}\|_F^2,\end{aligned} \tag{14}$$

where X^i and $W^{i-1}X^{i-1}$ play the roles of x and y in (13), respectively. Similarly, we can show that the optimality condition of problem (14) for X^i is

$$\mathbf{0} \in \phi^{-1}(X^i) - W^{i-1}X^{i-1}, \tag{15}$$

where $\phi^{-1}(X^i)$ is also defined element-wise. So the optimal solution to (14) is $X^i = \phi(W^{i-1}X^{i-1})$, which is exactly the constraint in problem (2). Thus we may relax problem (2) naively as:

$$\begin{aligned}&\min_{\{W^i\},\{X^i\}} \ell(X^n, D)\\ &+\sum_{i=2}^{n} \mu_i \left(\mathbf{1}^T f(X^i)\mathbf{1} + \frac{1}{2}\|X^i - W^{i-1}X^{i-1}\|_F^2\right).\end{aligned} \tag{16}$$

However, since the recursion constraints in (2) are not independent, there is a fundamental difference between problem (11) and problem (2). Namely, X^i appears in both $X^i = \phi(W^{i-1}X^{i-1})$ and $X^{i+1} = \phi(W^iX^i)$ for $i = 2, \cdots, n-1$. So it also appears in both $h(X^i, W^{i-1}X^{i-1})$ and $h(X^{i+1}, W^iX^i)$. In order to correctly approximate problem (2), $X^i = \phi(W^{i-1}X^{i-1})$ and $X^{i+1} = \phi(W^iX^i)$ should both satisfy the optimality condition of

$$\min_{X^i} \mu_i h(X^i, W^{i-1}X^{i-1}) + \mu_{i+1} h(X^{i+1}, W^iX^i)$$

for X^i, which is as follows:

$$\begin{aligned}\mathbf{0} \in &\mu_i(\phi^{-1}(X^i) - W^{i-1}X^{i-1})\\ &+\mu_{i+1}(W^i)^T(W^iX^i - X^{i+1}),\ i = 2, \cdots, n-1.\end{aligned} \tag{17}$$

It is also the optimality condition of problem (16) for X^i. Unfortunately, we can see that $X^i = \phi(W^{i-1}X^{i-1})$ and $X^{i+1} = \phi(W^iX^i)$ do *not* both satisfy the above!

In order for $X^i = \phi(W^{i-1}X^{i-1})$ and $X^{i+1} = \phi(W^iX^i)$ to both satisfy the optimality condition of the relaxed problem w.r.t. X^i, we need to modify (17) as

$$\begin{aligned}\mathbf{0} \in &\mu_i(\phi^{-1}(X^i) - W^{i-1}X^{i-1})\\ &+\mu_{i+1}(W^i)^T(\phi(W^iX^i) - X^{i+1}),\ i = 2, \cdots, n-1.\end{aligned} \tag{18}$$

This corresponds to the following problem:

$$\begin{aligned}&\min_{\{W^i\},\{X^i\}} \ell(X^n, D) + \sum_{i=2}^{n} \mu_i \Big(\mathbf{1}^T f(X^i)\mathbf{1}\\ &+\mathbf{1}^T g(W^{i-1}X^{i-1})\mathbf{1} + \frac{1}{2}\|X^i - W^{i-1}X^{i-1}\|_F^2\Big),\end{aligned} \tag{19}$$

where

$$g(x) = \int_0^x (\phi(y) - y)dy.$$

Similarly, $g(X)$ is an element-wise function for a matrix X. Problem (19) is the final formulation of LPOM. Note that the recursion constraints in (2) ensure that $\{X^i\}_{i=2}^n$ match the feed-forward pass of the network. The introduction of g leads to correct reformulation and also allows us to use simple feed-forward process to infer new samples. We want to highlight that this is non-trivial and non-obvious. We give the $f(x)$'s and $g(x)$'s of some representative activation functions in Table 3.

3.2 Advantages of LPOM

We denote $F(W, X)$ as the objective function of LPOM in (19). Then we have the following theorem:

Theorem 1. *Suppose that $\ell(X^n, D)$ is convex in X^n and ϕ is non-decreasing. Then $F(W, X)$ is block multi-convex, i.e., it is convex in each X^i and W^i while keeping the other blocks of variables fixed.*

Proof. $F(W, X)$ can be equivalently rewritten as

$$\begin{aligned}F(W, X) = \ell(X^n, D) + &\sum_{i=2}^{n} \mu_i \Big(\mathbf{1}^T \tilde{f}(X^i)\mathbf{1}\\ &+\mathbf{1}^T \tilde{g}(W^{i-1}X^{i-1})\mathbf{1} - \langle X^i, W^{i-1}X^{i-1}\rangle\Big),\end{aligned} \tag{20}$$

where $\tilde{f}(x) = \int_0^x \phi^{-1}(y)dy$ and $\tilde{g}(x) = \int_0^x \phi(y)dy$. Since both ϕ and ϕ^{-1} are non-decreasing, both $\tilde{f}(x)$ and $\tilde{g}(x)$ are convex. It is easy to show that $\mathbf{1}^T\tilde{g}(W^{i-1}X^{i-1})\mathbf{1}$ is convex in X^{i-1} when W^{i-1} is fixed and convex in W^{i-1} when X^{i-1} is fixed. The term $\langle X^i, W^{i-1}X^{i-1}\rangle$ in $F(W, X)$ is linear in one block when the other two blocks are fixed. The proof is completed. □

The above theorem allows us to use BCD methods to solve LPOM. Since each subproblem is convex, we can obtain the optimal solutions for updating X^i and W^i. In contrast, the subproblems in the penalty and the ADMM based methods are all nonconvex.

When compared with ADMM based methods [17], [21], LPOM does not contain Lagrange multipliers and needs no more auxiliary variables than $\{X^i\}_{i=2}^n$. Moreover, our solving method of LPOM does not need additional auxiliary variables (see Section 4). So LPOM has much less variables than ADMM based methods and hence needs less memory. Actually, since SGD needs to save $\{X^i\}_{i=2}^n$, the memory cost of LPOM is almost the same as that of SGD[2].

When compared with the penalty methods [18], [19], the optimality conditions of LPOM are simpler. For example, the optimality conditions for $\{X^i\}_{i=2}^{n-1}$ and $\{W^i\}_{i=1}^{n-1}$ in LPOM are (18) and

$$(\phi(W^iX^i) - X^{i+1})(X^i)^T = \mathbf{0}, \quad i = 1, \cdots, n-1, \tag{21}$$

while those for MAC are

$$\begin{aligned}&(X^i - \phi(W^{i-1}X^{i-1}))\\ &+ (W^i)^T[(\phi(W^iX^i) - X^{i+1}) \circ \phi'(W^iX^i)] = \mathbf{0},\\ &i = 2, \cdots, n-1.\end{aligned} \tag{22}$$

and

$$[(\phi(W^iX^i) - X^{i+1}) \circ \phi'(W^iX^i)](X^i)^T = \mathbf{0}, i = 1, \cdots, n-1. \tag{23}$$

We can see that the optimality conditions for MAC have an extra $\phi'(W^iX^i)$, which is nonlinear. The optimality

2. We implement SGD and LPOM using MATLAB. At the end of each epoch training in Table 5, we sum up all variables' memory usages. SGD and LPOM use the identical memory (628.431MB).

conditions for [19] can be found in Appendix A of the Supplementary Material. They also have an extra $\phi'(U^i)$. This may imply that the solution sets of MAC and [19] are more complex than those of LPOM. So it may be easier for LPOM to find good solutions.

When compared with the equivalent representations of the activation functions methods [22], [23], the formulation and the solving method of LPOM can handle much more general activation functions than [22] and [23], respectively. Note that a solving method is general means that it is identical for all feasible activation functions. Moreover, when the weights of a network are trained, the introduction of g in LPOM allows us to apply a feed-forward process to predict the label of a test sample. In comparison, Zhang and Brand [22] only considered the ReLU activation function. To get the potential label of a test sample, they had to solve an optimization problem or retrain the output layer. Although the formulation in [23] is applicable to general activation functions, its solving method is not general and the presented solving method is for ReLU only. Moreover, the formulation of Askari et al. [23] is incorrect as its optimality conditions for $\{X^i\}_{i=2}^{n-1}$ and $\{W^i\}_{i=1}^{n-1}$ are

$$\mathbf{0} \in \mu_i(\phi^{-1}(X^i) - W^{i-1}X^{i-1}) - \mu_{i+1}(W^i)^T X^{i+1}, \quad i=2,\cdots,n-1,$$

and

$$X^{i+1}(X^i)^T = \mathbf{0},\ i=1,\cdots,n-1,$$

respectively. We can see that the recursive equality constraints in (2) do not satisfy the above. Furthermore, somehow Askari et al. [23] added extra constraints $X^i \geq \mathbf{0}$ for any activation function. So their formulation cannot approximate the original problem (2) well. This may explain why the results of Askari et al. [23] are not good. Actually, they can only initialize the weights of feed-forward NNs with ReLU activations for SGD.

When compared with gradient-based methods, such as SGD, LPOM can handle any non-decreasing Lipschitz continuous activation function without computational difficulties, including being saturating (e.g., sigmoid and tanh) and non-differentiable (e.g., ReLU and leaky ReLU) and could update the layer-wise weights and activations in parallel (see Section 5). In contrast, gradient-based methods can only handle limited activation functions (e.g., ReLU, leaky ReLU, and softplus), in order to avoid the gradient vanishing or exploding issues, and they do not easily parallelize over layers when computing the gradients and activations. Moreover, gradient-based methods need much parameter tuning, which is not easy [15], while the tuning of penalty parameters μ_i's in LPOM is much simpler.

4 Solving LPOM

The block multi-convexity (Theorem 1) motivates a BCD method to solve LPOM. Namely, we update X^i or W^i by fixing all other blocks of variables. We summarize the whole solving process in Algorithm 1, where the optimization is performed using a mini-batch of training samples. We can prove the convergence of Algorithm 1[3]. Below we give the details of the algorithm, which is serial.

3. The proofs of theorems can be found in the Supplementary Material.

Algorithm 1 Solving LPOM

```
Input: training dataset, batch size $m_1$, iteration no.s $S$ and $K_1$.
  for $s = 1$ to $S$ do
    Randomly choose $m_1$ training samples $X^1$ and $D$.
    ⓐ Solve $\{X^i\}_{i=2}^{n-1}$ by iterating Eq. (26) for $K_1$ times.
    ⓑ Solve $X^n$ by iterating Eq. (29) for $K_1$ times.
    ⓒ Solve $\{W^i\}_{i=1}^{n-1}$ by applying Algorithm 2 to (31).
  end for
Output: $\{W^i\}_{i=1}^{n-1}$.
```

4.1 Updating $\{X^i\}_{i=2}^{n}$

We first describe the update of $\{X^i\}_{i=2}^n$. We update $\{X^i\}_{i=2}^n$ from $i = 2$ to n successively, just like the feed-forward process of NNs. For $i = 2,\cdots,n-1$, with $\{W^i\}_{i=1}^{n-1}$ and other $\{X^j\}_{j=2,j\neq i}^n$ fixed, problem (19) reduces to

$$\min_{X^i} \mu_i\left(\mathbf{1}^T f(X^i)\mathbf{1} + \frac{1}{2}\|X^i - W^{i-1}X^{i-1}\|_F^2\right) + \mu_{i+1}\left(\mathbf{1}^T g(W^iX^i)\mathbf{1} + \frac{1}{2}\|X^{i+1} - W^iX^i\|_F^2\right). \tag{24}$$

Its optimality condition is:

$$\mathbf{0} \in \mu_i(\phi^{-1}(X^i) - W^{i-1}X^{i-1}) + \mu_{i+1}((W^i)^T(\phi(W^iX^i) - X^{i+1})). \tag{25}$$

Based on fixed-point iteration [27] and in order to avoid using ϕ^{-1}, we may update X^i by iterating

$$X^{i,t+1} = \phi\left(W^{i-1}X^{i-1} - \frac{\mu_{i+1}}{\mu_i}(W^i)^T(\phi(W^iX^{i,t}) - X^{i+1})\right) \tag{26}$$

until convergence, where the superscript t is the iteration number. The convergence analysis is as follows:

Theorem 2. *Suppose that ϕ is differentiable and $|\phi'(x)| \leq \kappa$. If $\rho < 1$, then the iteration is convergent and the convergent rate is linear, where $\rho = \frac{\mu_{i+1}}{\mu_i}\kappa^2\sqrt{|||(W^i)^T||W^i|||_1\ |||(W^i)^T||W^i|||_\infty}$.*

Note that the choice of ρ in the above theorem is quite conservative. So in our experiments, we do not obey the choice as long as the iteration converges.

When considering X^n, problem (19) reduces to

$$\min_{X^n} \ell(X^n, D) + \mu_n\left(\mathbf{1}^T f(X^n)\mathbf{1} + \frac{1}{2}\|X^n - W^{n-1}X^{n-1}\|_F^2\right). \tag{27}$$

Assume that $\ell(X^n, D)$ is differentiable w.r.t. X^n. Then the optimality condition is:

$$\mathbf{0} \in \frac{\partial \ell(X^n, D)}{\partial X^n} + \mu_n(\phi^{-1}(X^n) - W^{n-1}X^{n-1}). \tag{28}$$

Also by fixed-point iteration, we may update X^n by iterating

$$X^{n,t+1} = \phi\left(W^{n-1}X^{n-1} - \frac{1}{\mu_n}\frac{\partial \ell(X^{n,t}, D)}{\partial X^n}\right) \tag{29}$$

until convergence. The convergence analysis is as follows:

Theorem 3. *Suppose that $\phi(x)$ is differentiable and $|\phi'(x)| \leq \kappa$ and $\left|\left|\left|\left(\frac{\partial^2 \ell(X,D)}{\partial X_{kl}\partial X_{pq}}\right)\right|\right|\right|_1 \leq \eta$. If $\tau < 1$, then the iteration is convergent and the convergent rate is linear, where $\tau = \frac{\kappa\eta}{\mu_n}$.*

If $\ell(X^n, D)$ is the MSE loss function, i.e., $\ell(X^n, D) = \frac{1}{2}\|X^n - D\|_F^2$, then $\left|\left|\left|\left(\frac{\partial^2 \ell(X,D)}{\partial X_{kl}\partial X_{pq}}\right)\right|\right|\right|_1 = 1$. So we have $\mu_n > \kappa$.

It is worth noting that f and g in (19) are integrals. Their expressions are very complex and may be not analytic (see Table 3). So it is best to use their derivatives rather than themselves in the computation. However, the derivative of f includes ϕ^{-1} (see (25) and (28)). We argue that it is better to avoid using ϕ^{-1} when updating $\{X^i\}_{i=2}^n$. The reasons are as follows. For a commonly used activation function ϕ, the domain of ϕ^{-1} (or equivalently the range of ϕ) is usually not $(-\infty, +\infty)$. So we need to constrain the input of ϕ^{-1} in the computation. This results in constrained problems and cannot be solved as efficiently as the unconstrained ones. The other big disadvantage of using ϕ^{-1} is that it may not be single-valued (See Table 3). This can impose great difficulty in computation and ensuring convergence. Moreover, different ϕ^{-1} may have a different domain. Since one has to tailor the update of $\{X^i\}_{i=2}^n$ for each ϕ^{-1}, the whole solving method cannot be applicable to general activation functions. This is the reason why the solving methods in [23] and [24] need to be tailored for every activation function. Since we only use ϕ and do not use ϕ^{-1}, our approach has no such limitation. So our solving method is more practical than those in [23] and [24].

4.2 Updating $\{W^i\}_{i=1}^{n-1}$

The update of $\{W^i\}_{i=1}^{n-1}$ is fully parallel. When $\{X^i\}_{i=2}^n$ are fixed, problem (19) reduces to

$$\min_{W^i} \mathbf{1}^T g(W^i X^i)\mathbf{1} + \frac{1}{2}\|W^i X^i - X^{i+1}\|_F^2,\ i = 1, \cdots, n-1. \tag{30}$$

The above problem for solving W^i is independent of the other problems for $\{W^j\}_{j=1, j\neq i}^{n-1}$. So it can be solved in parallel. We rewrite (30) as

$$\min_{W^i} \mathbf{1}^T \tilde{g}(W^i X^i)\mathbf{1} - \langle X^{i+1}, W^i X^i\rangle, \tag{31}$$

where $\tilde{g}(x) = \int_0^x \phi(y)dy$, as introduced before. Suppose that $\phi(x)$ is β-Lipschitz continuous, which is true for almost all used activation functions. Then $\tilde{g}(x)$ is β-smooth:

$$|\tilde{g}'(x) - \tilde{g}'(y)| = |\phi(x) - \phi(y)| \leq \beta|x - y|. \tag{32}$$

We solve problem (31) by APG [28] by locally linearizing $\hat{g}(W) = \tilde{g}(WX)$. However, since the Lipschitz constant of the gradient of $\hat{g}(W)$, which is $\beta\|X\|_2^2$, can be very large, the convergence can be slow. So we develop a variant of APG that is tailored for solving (31) much more efficiently.

From an optimization perspective, problem (31) can be expressed more generally as

$$\min_x G(x) \equiv \varphi(Ax) + \psi(x), \tag{33}$$

where both $\varphi(y)$ and $\psi(x)$ are convex and $\varphi(y)$ is L_φ-smooth: $\|\nabla\varphi(x) - \nabla\varphi(y)\| \leq L_\varphi\|x - y\|, \forall x, y$. Assume that the following minimization

$$x_{k+1} = \operatorname*{argmin}_x \langle\nabla\varphi(Ay_k), A(x - y_k)\rangle + \frac{L_\varphi}{2}\|A(x - y_k)\|^2 + \psi(x) \tag{34}$$

Algorithm 2 Solving problem (33).

```
Input: $x_0$, $x_1$, $\theta_0 = 0$, $k = 1$, iteration no. $K_2$.
  for $k = 1$ to $K_2$ do
      Compute $\theta_k$ via $1 - \theta_k = \sqrt{\theta_k}(1 - \theta_{k-1})$.
      Compute $y_k$ via $y_k = \theta_k x_k - \sqrt{\theta_k}(\theta_{k-1}x_{k-1} - x_k)$.
      Update $x_{k+1}$ via (34).
   end for
Output: $x_k$.
```

is easy to solve for any given y_k. We propose Algorithm 2 to solve (33), which is derived from the proof of its convergence theorem:

Theorem 4. *If we use Algorithm 2 to solve problem (33), then the convergence rate is at least $O(k^{-2})$:*

$$G(x_k) - G(x^*) + \frac{L_\varphi}{2}\|z_k\|^2 \leq \frac{4}{k^2}\left(G(x_1) - G(x^*) + \frac{L_\varphi}{2}\|z_1\|^2\right),$$

where $z_k = A[\theta_{k-1}x_{k-1} - x_k + (1 - \theta_{k-1})x^*]$ and x^* is any optimal solution to problem (33).

When we consider solving problem (31), the instantiation of subproblem (34) becomes:

$$\begin{aligned} W^{i,t+1} = &\operatorname*{argmin}_W \langle\phi(Y^{i,t}X^i), (W - Y^{i,t})X^i\rangle \\ &+ \frac{\beta}{2}\|(W - Y^{i,t})X^i\|_F^2 - \langle X^{i+1}, WX^i\rangle. \end{aligned} \tag{35}$$

It is a least-square problem and its solution is:

$$W^{i,t+1} = Y^{i,t} - \frac{1}{\beta}(\phi(Y^{i,t}X^i) - X^{i+1})(X^i)^\dagger, \tag{36}$$

where $Y^{i,t}$ plays the role of y_k in Algorithm 2.

5 Parallel Solution of LPOM

The solving method presented in the previous section is serial. In this section, we investigate the parallel solution of LPOM.

5.1 Asynchronous-Parallel LPOM

As presented in Section 4, the original method to solve LPOM is BCD. So it is natural to use asynchronous-parallel BCD (async-BCD) [29], [30] to solve LPOM in parallel. However, the existing async-BCD mainly uses gradient descent to update each block only once, while in LPOM the gradient w.r.t. X^i involves the inverse of activation function, which is the reason why LPOM uses fixed-point iteration instead (see (26) and (29)). So the existing async-BCD actually does not apply. In the following, we propose a new async-BCD method.

Suppose the optimization problem is

$$\min_{\{z_i\}} F(z_1, \cdots, z_n). \tag{37}$$

Our async-BCD goes as follows: If z_i is chosen to be updated, it is updated as:

$$\begin{aligned} z_i^{k+1} = &\operatorname*{argmin}_{z_i} F(z_1, \cdots, z_{i-1}, z_i, z_{i+1}, \cdots, z_n) \\ &+ \frac{\gamma}{2}\|z_i - z_i^k\|^2, \end{aligned} \tag{38}$$

where z_j's, $j \neq i$, take the latest value *that is available to* z_i. Note that such values of z_j's may be older than the really latest values of z_j's that is being computed, due to, say, communication delay. The proximal term $\frac{\gamma}{2}\|z_i - z_i^k\|^2$ is necessary to guarantee the convergence of our async-BCD.

To analyze the convergence of our async-BCD, we make the following assumptions.

Assumption 1. *$F(z) \equiv F(z_1, \cdots, z_n)$ has coordinate Lipschitz continuous gradients, i.e. for all $i \in \{1, \cdots, n\}$, we have*

$$\|\nabla_i F(z) - \nabla_i F(\bar{z})\| \leq L_i \|z - \bar{z}\|.$$

Assumption 2. *The staleness in information is bounded, i.e.,*

$$k - \tau_j^i(k) \leq \Delta, \quad k = 0, 1, \ldots,$$

where $\tau_j^i(k)$ is the time of z_j that is available to update z_i at time k.[4]

We note that both assumptions are standard for analyzing asynchronous algorithms [29], [30]. Now we claim the convergence guarantee for our async-BCD.

Theorem 5. *Suppose that $F(z)$ is block multi-convex. Under Assumptions 1 and 2, by setting the proximity parameter γ in* (38) *as: $\frac{\gamma}{2} + \frac{L^2}{2\gamma}(\Delta+1) - \frac{L^2}{2\gamma}(\Delta+1)^2 > 0$, where $L = \max\{L_i\}$, we have $\left\|\nabla_i F(z^k)\right\| \to 0, \quad \forall i \in \{1, \cdots, n\}$.*

Since $\gamma > 0$, we have $\gamma > L\sqrt{\Delta(\Delta+1)}$ based on the above theorem. So the choice of γ depends on the estimations of Δ and L. Our async-BCD can be applied to solve LPOM, resulting in async-LPOM. For async-LPOM, we may set Δ to be the number of CPU cores. Suppose that $\phi(x)$ is β-Lipschitz continuous (defined in (32)), we may set L as $L = \beta \max\{\|W^2\|_2^2, \cdots, \|W^{n-1}\|_2^2, \|X^1\|_2^2, \cdots, \|X^{n-1}\|_2^2\}$. The theoretical estimate of γ is very large. In practice, we choose γ by gradually increasing its value until async-LPOM converges. However, async-LPOM does not fully exploit the properties of problem (19) that each X^i is only related to X^{i-1} and X^i and $\{W^i\}_{i=1}^{n-1}$ can be updated with full parallelization. So async-LPOM is not as fast as we expected. This motivates us to consider developing synchronous-parallel LPOM (sync-LPOM), which better exploits the characteristics of LPOM.

5.2 Synchronous-Parallel LPOM

We first analyze the serial LPOM in Algorithm 1. When $\{X^i\}_{i=2}^n$ are fixed, the update of each W^i is independent of other W^j's, $j \neq i$ (see (30)). So step ⓒ in Algorithm 1 can be done with full parallelization. We generate $n-1$ threads, each thread solving for each W^i independently. With $\{W^i\}_{i=1}^{n-1}$ fixed, the update of X^i ($i \in \{2, \cdots, n-1\}$) is related to X^{i-1} and X^{i+1} (see (26)), and the update of X^n is related to X^{n-1} (see (29)). So we only need to synchronously parallelize steps ⓐ and ⓑ.

We parallelize the update of X^i as follows. We create $n-1$ threads. Each thread iterates Eq. (26) or Eq. (29) to update X^i. This step is executed by each thread independently and simultaneously. When all the threads finish the step, we repeat the procedure for more times until the maximum time is reached. However, we notice the coupling when updating adjacent X^i's (see (26) and (29)). Decoupling the blocks facilitates the convergence of a parallel algorithm. Maximizing the degree of parallelism is also pursued by a parallel algorithm. Then the odd-even alternating scheme to update X^i's is the only choice that can have both maximum parallelism and minimum coupling. Namely, we first synchronously update the odd subsequence $\{X^3, X^5, \cdots\}$. Then with the updated $\{X^3, X^5, \cdots\}$, we synchronously update the even subsequence $\{X^2, X^4, \cdots\}$. Besides, inspired by the momentum used in SGD, we take previous values of each X^i into consideration[5], resulting in a moving average of the updated sequence of X^i. The whole algorithm of sync-LPOM is summarized in Algorithm 3. Note that sync-LPOM is a special case of async-LPOM, where the order of blocks to be updated is allowed to be non-random. So the conclusion of Theorem 5 also applies to sync-LPOM.

Algorithm 3 Synchronously solving LPOM

```
Input: training dataset, batch size $m_1$, $0 < \nu < 1$, iteration no.s $S$, $K_1$, $K_2$, and $K_3$.
  for $s = 1$ to $S$ do
    Randomly choose $m_1$ training samples $X^1$ and $D$.
    // update $\{X^i\}_{i=2}^n$.
    for $k = 1$ to $K_1$ do
      // update the odd subsequence.
      Thread $i \in \{3, 5, \cdots\}$ executes the following steps:
        Solve $\tilde{X}^{i,k+1}$ by iterating Eq. (26) or (29) for $K_3$ times using $\{X^{i-1,k}, X^{i,k}, X^{i+1,k}\}$ or $\{X^{n-1,k}, X^{n,k}\}$.
        $X^{i,k+1} = \nu \tilde{X}^{i,k+1} + (1-\nu) X^{i,k}$.
      // update the even subsequence.
      Thread $i \in \{2, 4, \cdots\}$ executes the following steps:
        Solve $\tilde{X}^{i,k+1}$ by iterating Eq. (26) or (29) for $K_3$ times using $\{X^{i-1,k+1}, X^{i,k}, X^{i+1,k+1}\}$ or $\{X^{n-1,k+1}, X^{n,k}\}$.
        $X^{i,k+1} = \nu \tilde{X}^{i,k+1} + (1-\nu) X^{i,k}$.
    end for
    // update $\{W^i\}_{i=1}^{n-1}$.
    Thread $i \in \{1, \cdots, n-1\}$ executes the following step:
      Solve $W^i$ by iterating Eq. (36) for $K_2$ times.
  end for
Output: $\{W^i\}_{i=1}^{n-1}$.
```

6 Experiments

In this section, we conduct extensive experimental evaluations for the proposed LPOM as well as sync-LPOM. We first evaluate the performance of LPOM on image classification tasks. We also conduct ablation studies to analyze LPOM. We then investigate the computation efficiency and performance of sync-LPOM on autoencoder training tasks.

The reasons why we test sync-LPOM on the autoencoder training tasks are as follows. First, currently LPOM and sync-LPOM are only applicable to fully-connected feed-forward NNs. Second, our primary aim is to evaluate the speedup of sync-LPOM. However, sync-LPOM cannot be

4. Note that in this paper we follow the convention in [31]. k is a global counter: whenever a block is updated, k increases by 1.

5. In order to be consistent with sync-LPOM, serial LPOM also uses this update rule in the experiments.

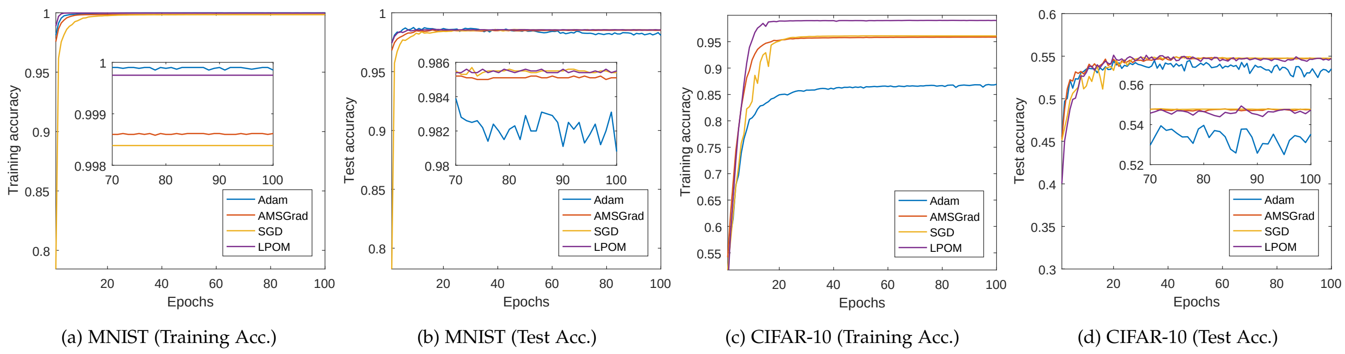


Fig. 1. Comparison of LPOM and three gradient-based methods on the MNIST and the CIFAR-10 datasets. The inset figures in (a), (b), and (d) show the accuracies for the final 30 epochs in detail. **Images in this paper are best viewed in color!**

accelerated if there exists layers with dominant computation. For example, with a 784-10-10-10 network, the update of X^2 is 16 times slower than those of X^3 and X^4. Both the serial LPOM and sync-LPOM will be dominated by updating X^2. So sync-LPOM will not have speedup over serial LPOM. As a result, sync-LPOM favours the equal-width networks, which are not common in classification networks. This motivates us choosing autoencoders. Third, deep autoencoders are challenging, because they have to reconstruct their inputs under the constraint that one layer is very low-dimensional. The difficulty mainly comes from the network architectures rather than the datasets used. Training autoencoders on benchmark datasets is considered as a standard problem for evaluating optimization methods on feed-forward NNs [18], [32]. For the above reasons, we test sync-LPOM with various autoencoder architectures and datasets.

6.1 Results of LPOM

We test LPOM by comparing with three representative gradient-based methods (SGD, Adam [10], and AMSGrad [11]) and three non-gradient based methods [17], [23], [24]. The other non-gradient based methods do not train fully-connected feed-forward NNs for classification tasks (e.g., using skip connections [22], training autoencoders [18], and learning for hashing [21]). So we do not compare with them. For simplicity, we use the MSE loss function. Since several methods [23], [24] only present the solving methods for ReLU, we use the ReLU activation function as well. Unlike [23] and [24], we do not use any regularization on the weights $\{W^i\}_{i=1}^{n-1}$ because it usually does not help to decrease the training loss. We run LPOM and three gradient-based methods with the same inputs and random initializations [33]. We implement LPOM, SGD, Adam, and AMSGrad with MATLAB. For LPOM, we set $\mu_i = 2$ in (19) for all the networks. For SGD, the learning rate is set to be the largest feasible value and multiply by 0.1 at the half and three quarters of the total epochs. For Adam and AMSGrad, we carefully tune the parameters to achieve the best performance. For [23] and [24], we use the source codes with default parameter settings provided by the authors. For [17], we read the results from Fig. 1 (b) of the paper.

6.1.1 *Comparison with Gradient-Based Methods*

We conduct experiments on the MNIST [34], CIFAR-10 [35], and ImageNet [36] datasets. The MNIST dataset has 60,000

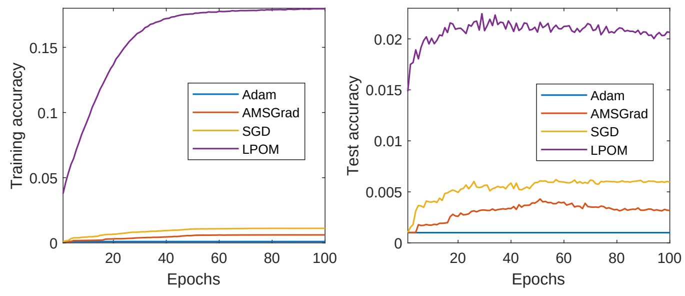


Fig. 2. Comparison of LPOM and three gradient-based methods on the ImageNet32x32 dataset.

training images and 10,000 test images associated with labels from 10 classes. We use $28{\times}28 = 784$ raw pixels as the inputs and do not use any pre-processing or data augmentation. For all the compared methods, in each epoch the entire training samples are passed through once. The network architecture may affect the performance. Since overparameterization greatly facilitates gradient-based methods when training [37], [38], we perform the comparisons with overparameterized networks. Following [19], we use a 784-2048-2048-2048-10 feed-forward network. We run all the methods for 100 epochs with a fixed batch size 100. The training and the test accuracies are shown in Fig. 1 (a) and (b). We can see that the final training accuracies of all the methods are approximately equal to 100%. However, the test accuracy of LPOM is comparable with or slightly better than those of Adam, AMSGrad, and SGD. The final test accuracies are: Adam 98.1%, AMSGrad 98.5%, SGD 98.5%, and LPOM 98.6%.

The CIFAR-10 dataset has 50,000 training and 10,000 test color images associated with labels from 10 classes. We use $32 \times 32 \times 3 = 3072$ raw pixels as the inputs. As in [19], we use a 3072-4000-1000-4000-10 feed-forward network. We normalize each color image by subtracting the training dataset's means of the red, green, and blue channels, respectively. We do not use any data augmentation. We run all the methods for 100 epochs with a fixed batch size 100. The training and the test accuracies are shown in Fig. 1 (c) and (d). We can see that only the training accuracy of LPOM is approximately equal to 100%. The gradient-based methods do not achieve zero training loss. This may be because the used network has a low-dimensional hidden layer, making it hard to optimize. When comparing the test accuracy, we can see that

TABLE 4
Comparison of test accuracies of LPOM, [23], and [24] on the MNIST dataset using different networks. For each network, the "Hidden Layers" denotes the network architecture of its hidden layers. For example, "300-100" means that the network has two hidden layers. The first and the second hidden layers have 300 and 100 neurons, respectively.

Hidden Layers	300	300-100	500-150	500-200-100	400-200-100-50
[23]	71.6%	72.2%	72.7%	73.8%	80.3%
[24]	97.2%	95.5%	95.9%	96.3%	96.1%
LPOM	**97.8%**	**97.6%**	**97.8%**	**97.9%**	**97.9%**

TABLE 5
Comparison of test accuracies of LPOM, SGD, [17], [23], and [24] on the SVHN dataset.

Methods	SGD	[17]	[23]	[24]	LPOM
Accuracy	95.0%	96.5%	97.5%	98.2%	**98.5%**

LPOM is able to match or beat the other methods. The final test accuracies are: Adam 53.5%, AMSGrad 54.7%, SGD 54.7%, and LPOM 54.7%. Notice that LPOM attains nearly perfect training accuracy but the test accuracy is comparable to other methods that have much lower training accuracy. We want to note that the situation is not an overfitting problem. First, overfitting is used to evaluate the performance of a trainable model rather than an optimization method. So the definition of overfitting does not apply. Second, if a model achieves a comparable or better training accuracy but the test accuracy is much worse than other models, we may consider that it overfits the training dataset. While the network trained by LPOM has a better training accuracy, its test accuracy is comparable to or slightly better than the networks trained by other methods. So the phenomenon of overfitting does not apply to this situation. Third, LPOM is an NN optimization method. So its primary goal is fitting the training dataset. The test accuracy depends strongly on the type and amount of weight regularization used during training, which is an important issue in practice. Since we focus on optimization, weight regularization is outside the scope of our discussion.

The ImageNet dataset has 1,281,167 training images and 50,000 validation images from 1,000 classes. The images in the ImageNet dataset are typically $224 \times 224 \times 3$ pixels. For fast experimentation, we use the ImageNet32x32 dataset [39] instead, which can be downloaded from the official website of the ImageNet project[6]. The ImageNet32x32 dataset preserves the complexity of the original ImageNet dataset and is widely adopted in automated machine learning (AutoML). It is a downsampled version of the ImageNet dataset with a size of $32 \times 32 \times 3$ pixels. It has the same number of images and classes as the ImageNet dataset. We take all validation images as the test set. We use the same preprocessing for each image as for the CIFAR-10 images. We use the raw pixels as the inputs and do not use any data augmentation. We run all the methods for 100 epochs with a fixed batch size 100. We use a 3072-1024-1024-1024-1000 feed-forward network. The training and the test accuracies are shown in Fig. 2. We can see that LPOM achieves the best training and test accuracies. The final training accuracies are: Adam 0.102%, AMSGrad 0.611%, SGD 1.112%, and LPOM 17.916%. The final test accuracies are: Adam 0.100%, AMSGrad 0.318%, SGD 0.596%, and LPOM 2.064%. These accuracies are very low compared to those reported in the literature. The reasons are as follows. First, we use a fully-connected NN rather than a CNN. Second, a lower resolution of the images makes the classification task much more difficult. Third, finding the optimal depth and hidden layer widths of a fully-connected NN that fits for ImageNet32x32 requires much effort, which is not the goal of our paper, and we did not find such references in the literature, so we only choose a relatively simple one. Fourth, we do not use data augmentation to enhance test performance. Fifth, we do not use extra learning tricks for all the compared methods, e.g., weight regularization, BN, and momentum.

6. http://image-net.org/download-images

TABLE 6
The training and the test accuracies of LPOM on the MNIST dataset with varying μ and activation function used. We set $\mu_i = \mu$ in (19).

	μ	sigmoid	tanh	ReLU	leaky ReLU	ELU	softplus
Training Acc.	2	0.9993	0.9987	0.9984	1.0000	0.9477	0.9769
	5	0.9989	0.9973	0.9969	0.9998	0.9748	0.9829
	10	0.9983	0.9976	0.9967	0.9996	0.9809	0.9812
	20	0.9965	0.9918	0.9971	0.9991	0.9804	0.9807
	50	0.9887	0.9915	0.9973	0.9991	0.9832	0.9358
	100	0.9735	0.9856	0.9971	0.9981	0.9811	0.9582
Test Acc.	2	0.9769	0.9758	0.9761	0.9790	0.9378	0.9618
	5	0.9784	0.9745	0.9761	0.9796	0.9590	0.9672
	10	0.9752	0.9748	0.9745	0.9780	0.9663	0.9632
	20	0.9738	0.9696	0.9745	0.9760	0.9674	0.9661
	50	0.9662	0.9716	0.9743	0.9767	0.9721	0.9282
	100	0.9598	0.9678	0.9746	0.9756	0.9707	0.9512

6.1.2 Comparison with Other Non-gradient Based Methods

We first compare with [23] and [24] using the same architectures on the MNIST dataset. As in [23], we run all the methods for 17 epochs with a fixed batch size 100. We do not use any pre-processing or data augmentation. Table 4 summarizes the test accuracies of the three methods, where the best values are in boldface. Since the formulation in [24] is equivalent to that of LPOM, we can see that [24] and LPOM with the ReLU activation function perform much better than [23]. This complies with our analysis in Subsection 3.2. Despite using equivalent formulations, LPOM still outperforms [24], which demonstrates the superiority of our solving method.

Following the settings of dataset and network architecture in [17], we test LPOM on the Street View House Numbers (SVHN) dataset [40]. Table 5 summarizes the test accuracies of SGD, [17], [23], [24], and LPOM, where the best values are in boldface. We can see that LPOM achieves the best performance. This further testifies to the advantage of LPOM.

6.1.3 Ablation Studies of LPOM

We perform ablation studies of LPOM to evaluate the effects of the penalty parameters μ_i's in (19) and the activation function used. We use the MSE loss function and the MNIST

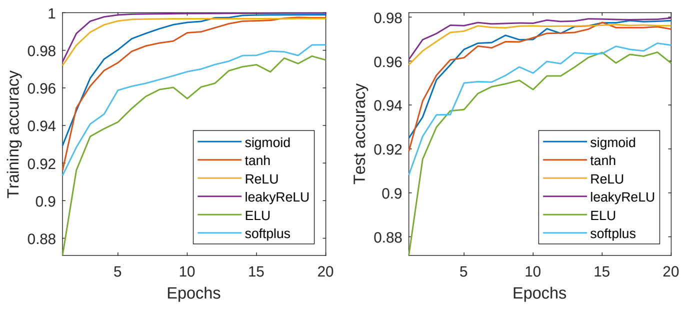


Fig. 3. Comparison of LPOM with various activation functions. The penalty parameters μ_i's in (19) are all fixed at 5.

dataset without using any pre-processing or data augmentation. We use a 784-400-200-100-50-10 feed-forward network, which is the deepest network in [23]. For all the cases, we use the identical source code and random initializations [33]. For each case, we run LPOM for 20 epochs with a fixed batch size 100.

We test LPOM with all the activation functions shown in Table 3. We set $\alpha = 0.01$ and $\alpha = 1$ for the leaky ReLU and ELU activation functions, respectively. For the penalty parameters μ_i's in (19), we set $\mu_i = \mu$ for every activation function, where $\mu \in \{2, 5, 10, 20, 50, 100\}$. The reason why we do not tune different μ_i's will be given later. Table 6 shows the final training and test accuracies of LPOM with various μ and activation functions. We can see that LPOM with every activation function remains stable over a significantly large range of μ. This further confirms that the parameter setting of LPOM is very easy. For the classification task, LPOM with the sigmoid, tanh, ReLU, or leaky ReLU activation function performs better than the rest two functions when $\mu \leq 50$. In Fig. 3, we compare the behavior of LPOM with $\mu_i = 5$ for different activation functions. It can be seen that LPOM with the ReLU or leaky ReLU activation function converges faster.

Our empirical parameter setting for LPOM (i.e., setting $\mu_i = \mu$ in (19)) actually has a theoretical ground. Gu et al. [24] developed a variable scaling strategy to prove that our parameter setting for LPOM is theoretically guaranteed for the ReLU activation function. Namely, we can theoretically reduce the penalty parameters μ_i's in (19) to only one parameter μ for ReLU. Actually, the variable scaling strategy works only for the positive homogeneous functions. Namely, $\phi(\lambda x) = \lambda\phi(x)$ for all $\lambda \geq 0$. Such functions can be written as:

$$\phi(x) = \begin{cases} \alpha x, & x \geq 0; \\ \beta x, & x < 0. \end{cases}, \tag{39}$$

where $\alpha > 0$ and $\beta \geq 0$. We provide the proof in Appendix G of the Supplementary Material. The form in (39) includes the linear, ReLU, and leaky ReLU activation functions. Our empirical setting $\mu_i = \mu$ in (19) has not been theoretically proven for all feasible activation functions, e.g., ELU, softplus, sigmoid, and tanh. However, we want to note that the positive homogeneous activation functions are the most popular in practice. The ELU and softplus activation functions are smooth approximations to the positive homogeneous counterparts. So using a single parameter μ also roughly applies to ELU and softplus. As for the sigmoid and tanh activation functions, we can see from Table 6 that they also works very well when setting $\mu_i = \mu$ in (19). So we argue that using a single penalty parameter in LPOM should suffice for most of the commonly used activation functions. Finally, we want to highlight that Theorem 3 gives a theoretical reference for choosing μ_n. For example, when using the MSE loss function and the tanh activation function, we should set $\mu_n > 1$ for LPOM. Overall, we may conclude that the tuning of penalty parameters in LPOM is very simple.

6.2 Results of Sync-LPOM

We implement serial LPOM, sync-LPOM, SGD, Adam, and AMSGrad in C++. No regularization is used on the weights $\{W^i\}_{i=1}^{n-1}$. We use the MSE loss function and the ReLU activation function. A smaller value of training or test loss identifies a better performance. For all the methods, we use the identical inputs, random initializations [33], and batch size (here is 100). For sync-LPOM, we use POSIX Threads as the parallel programming framework. For both LPOM and sync-LPOM, we set $\mu_i = 20$. For SGD, the learning rate is set to be the largest feasible value and multiply by 0.1 at the half and three quarters of the total epochs. For Adam and AMSGrad, we tune the parameters to achieve the best performance. All the experiments are carried out on a single Intel(R) Core(TM) i7-8700 CPU 3.20GHz computer with 12 cores. As in [32], we test on three gray-scale image datasets, which is summarized in Table 7. The Curves dataset consists of synthetic curves, the MNIST dataset contains hand-written digits, and the Faces dataset is the augmented Olivetti face dataset.

6.2.1 *Computation Efficiency on CPUs*

The encoding and decoding times for all the methods are equal. So we only compare the time cost in the training. We conduct three groups of experiments on the MNIST dataset.

For programming convenience, we assign each layer to one CPU core, which reduces the interferences between the updates of different layers and thus can better compare the speedup performance. Accordingly, for sync-LPOM, the ideal number of cores is equal to the number of network layers. We first test the speedup of sync-LPOM with ideal CPU cores on various equal-width autoencoders (here the dimension is 784). The layer speedup ratio is defined as:

$$\text{layer speedup ratio} = \frac{\text{serial time per epoch}}{\text{parallel time per epoch}}.$$

The training time and the layer speedup ratio are shown in Fig. 4 (a) and (b), respectively. We can see that the training time of sync-LPOM increases more slowly than serial LPOM with increasing hidden layers, and the layer speedup ratio is nearly linear.

Then with fixed depth and width (here is 784) of autoencoders, we test the speedup of sync-LPOM with varying numbers of CPU cores. The core speedup ratio here is defined as:

$$\text{core speedup ratio} = \frac{\text{serial time per epoch}}{\text{parallel time per epoch using } d \text{ CPU cores}}.$$

The results are shown in Fig. 4 (c) and (d). Let n be the number of layers. When $d \leq n/2$, we can see that all the core

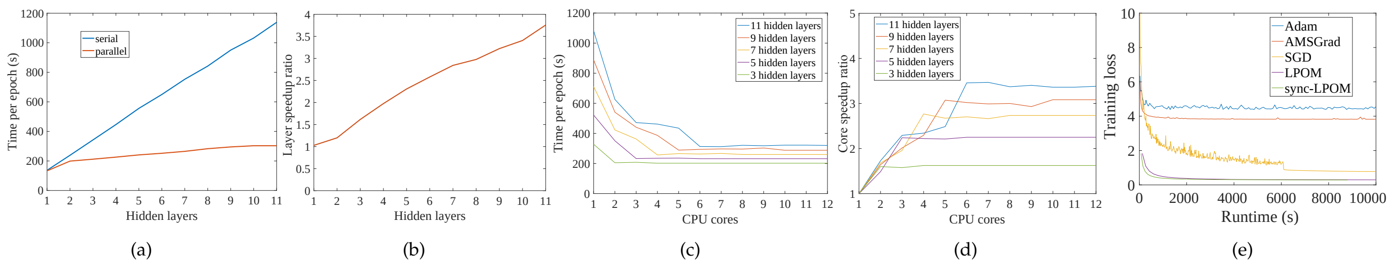
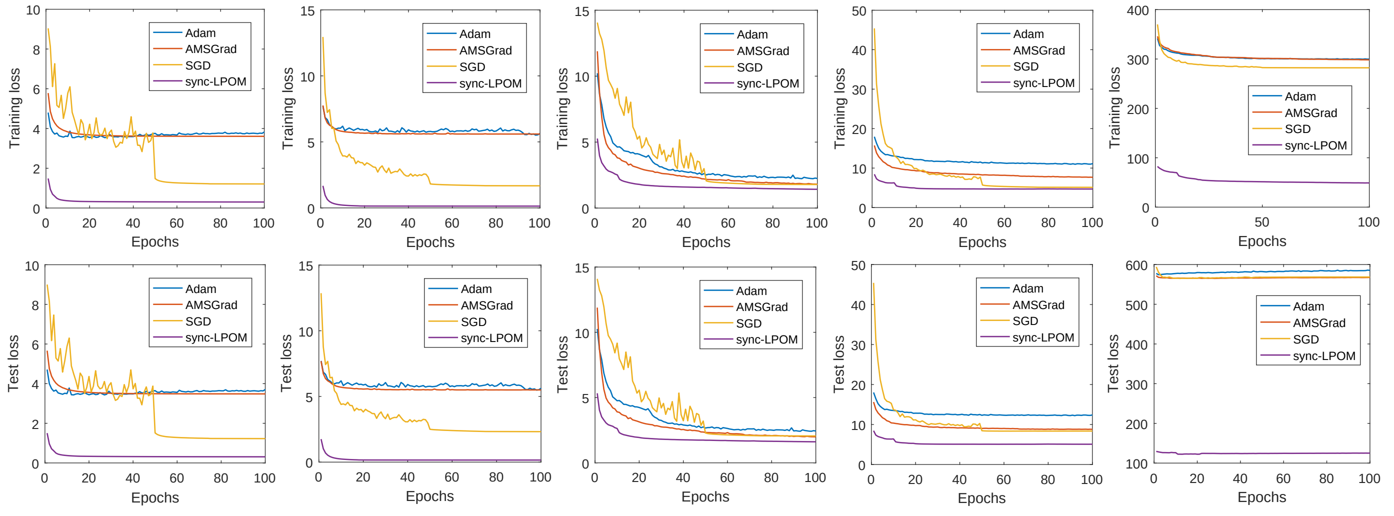


Fig. 4. Results of the speedup experiments. (a) and (b) are the training time and the speedup ratio with a variety of layers, respectively. (c) and (d) are the training time and the speedup ratio with a variety of CPU cores, respectively. (e) are curves of the training loss vs. the running time.



(a) MNIST (single-layer): >**20x** (b) MNIST (multi-layer): >**20x** (c) Curves (deep): **1.4x** (d) MNIST (deep): **1.5x** (e) Faces (deep): >**20x**

Fig. 5. Comparison of Adam, AMSGrad, SGD, and sync-LPOM with single-layer, multi-layer, and deep autoencoders. The first row is the training losses, while the second row is the test losses. The architectures are given in Table 7. The numbers in boldface denote the real speedup ratios of sync-LPOM over SGD with respective autoencoders.

TABLE 7
Experimental settings of autoencoder training.

Dataset	#Training	#Test	Encoder Dimensions
MNIST	60,000	10,000	784-500 784-1024-500
Curves	20,000	10,000	784-400-200-100-50-25-6
MNIST	60,000	10,000	784-1000-500-250-30
Faces	103,500	41,400	625-2000-1000-500-30

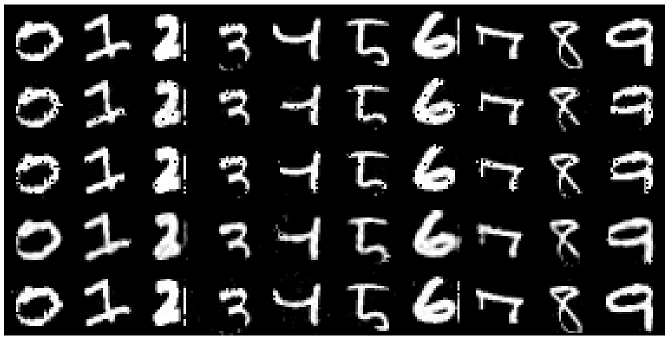


Fig. 6. Reconstructions using multi-layer autoencoders trained on the MNIST dataset. The top row is original images. The following rows are the results of autoencoders trained by Adam, AMSGrad, SGD, and sync-LPOM, respectively.

speedup ratios grow linearly with increasing CPU cores. When $d > n/2$, the core speedup ratios become constant (see Fig. 4 (d)). The reason is as follows. Due to the odd-even alternating scheme of sync-LPOM for updating X^i, we only need $n/2$ cores to synchronously update X^i. Synchronously updating W^i requires $n-1$ cores. However, updating X^i is more expensive than updating W^i. So the core speedup ratio with $n/2$ cores is competitive with n cores.

In Fig. 4 (e), we plot the training loss against the running time for Adam, AMSGrad, SGD, LPOM, and sync-LPOM with the autoencoder 784-1024-500-1024-784 [41]. For SGD, the learning rate multiplies 0.1 at the 300-th and 450-th epochs. We can see that sync-LPOM converges the fastest and achieves the lowest training loss.

6.2.2 Optimization Performance

We test sync-LPOM with two shallow and three deep autoencoders considered in [32], [41]. Table 7 gives the used datasets, the sizes of the training and the test sets, and the tested encoder network architectures. Note that we use symmetric autoencoders, where the decoder architecture is the mirror image of the encoder. We run all the methods for 100 epochs. For SGD, the learning rate multiplies 0.1 at the 50-th and 75-th epochs.

The training and the test losses with the single-layer and the multi-layer autoencoders are shown in Fig. 5 (a) and (b), respectively. We can see that sync-LPOM indeed has the best performance. Adam and AMSGrad achieve lower training and test losses than SGD in the first several epochs. However, they are inferior to SGD at the end of training. Some reconstructed samples with multi-layer autoencoders on the test set are shown in Fig. 6. We can see that sync-LPOM has the best reconstruction performance. Deep autoencoder problems are more difficult than the shadow ones. The training and the test losses are shown in Fig. 5 (c)-(e). It can be seen that on the Curves and the MNIST datasets, sync-LPOM is slightly better than SGD. However, when considering the Faces dataset, we can see that sync-LPOM is significant better than SGD, Adam, and AMSGrad.

6.2.3 Discussions

We also test the real speedups of sync-LPOM over SGD with the five autoencoders shown in Table 7. For each antoencoder, we first run sync-LPOM for 50 epochs and record its running times and training losses. We denote its time and loss sequences as $\{t_i^1\}_{i=1}^{50}$ and $\{e_i^1\}_{i=1}^{50}$, respectively. Let e_j^1 be the first element in $\{e_i^1\}_{i=1}^{50}$ that is less than or equal to $e_{50}^1+0.05$, which is called as the initial stable point. Then we run SGD until its training loss e_k^2 is less than or equal to e_j^1. We denote the corresponding running time as t_k^2. Then the real speedup ratio of sync-LPOM over SGD is defined as: t_k^2/t_j^1, where a value greater than 1 means that sync-LPOM is faster than SGD to achieve a satisfactory training loss. If SGD runs 20 times longer than t_j^1 but its loss is still higher than e_j^1, we terminate SGD and mark the real speedup ratio as: >20x. The real speedup ratio with each autoencoder is shown below the corresponding figure in Fig. 5, which is in boldface. We can see that sync-LPOM achieves superior speedups over SGD with various autoencoders.

We want to note that we do not compare with parallel SGD. The reasons are as follows. First, LPOM is parallelizable across data or layers. We only utilize its layer parallelization. Implementing data parallelization is more or less an engineering work. So we do not work on this. Since the parallel SGD in [42] is data-parallel, we do not include it for comparison. Second, async-SGD and sync-SGD may achieve speedup, but their performances are much worse than SGD [43]. Third, *we never claim that sync-LPOM is faster than SGD in one epoch. We only claim that it is faster than SGD to achieve a relatively low training loss.* So we use serial SGD as our competitor, which is more reasonable.

That sync-LPOM (or LPOM) achieves lower training and test losses than SGD is because it properly lifts the dimension of the problem and solves the lifted problem instead. So it can easily move to good points in the higher dimensional space. In contrast, SGD directly optimizes the problem. So the optimization path is restricted by the loss landscape, which is extremely complex. So we credit the success of sync-LPOM (or LPOM) to "the blessing of dimensionality."

7 Conclusion

In this work we have developed LPOM to train fully-connected feed-forward NNs. By rewriting the activation function as an equivalent proximal operator, LPOM formulates the training of an NN as a block multi-convex model. This leads to our novel BCD solving method with convergence guarantee. Due to the formulation and solving method, LPOM avoids the gradient vanishing or exploding problem and is applicable to general non-decreasing Lipschitz continuous activation functions. Moreover, it does not require more auxiliary variables than the layer-wise activations and its parameter tuning is relatively simple. It could also be solved in parallel. We first present a new async-BCD method with convergence guarantee. Then we use it to solve LPOM and obtain async-LPOM. For faster speed, we develop the sync-LPOM. Our experimental results show that LPOM works better than SGD, Adam, AMSGrad, [23], [17], and [24] on fully-connected feed-forward NNs. We also verify the efficiency and effectiveness of sync-LPOM on various autoencoder training problems. Future work includes extending LPOM to train convolutional and recurrent neural networks and applying LPOM to network quantization.

Acknowledgments

Z. Lin is supported by National Key R&D Program of China (2019AAA0105200), NSF of China (grant nos. 61625301 and 61731018), Beijing Academy of Artificial Intelligence, and Zhejiang Lab (grant nos. 2019KB0AC01 and 2019KB0AB02). J. Li is supported by NSF of China (grant nos. 61802269 and 61972132) and the Fundamental Research Funds for the Central Universities. C. Xu is supported by NSF of China (grant no. 61876007).

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.
[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.
[3] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.
[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.
[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press Cambridge, 2016, vol. 1.
[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, p. 533, 1986.
[7] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the International Conference on Machine Learning*, 2013, pp. 1139–1147.
[8] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.
[9] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, no. 2, pp. 26–31, 2012.
[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations*, 2015.
[11] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and beyond," in *Proceedings of the International Conference on Learning Representations*, 2018.

[12] R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping from saddle points-online stochastic gradient for tensor decomposition," in *Proceedings of the Conference on Learning Theory*, 2015, pp. 797–842.

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[15] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the International Conference on Machine Learning*, 2011, pp. 265–272.

[16] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 4107–4115.

[17] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein, "Training neural networks without gradients: A scalable ADMM approach," in *Proceedings of the International Conference on Machine Learning*, 2016, pp. 2722–2731.

[18] M. Carreira-Perpinan and W. Wang, "Distributed optimization of deeply nested systems," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2014, pp. 10–19.

[19] J. Zeng, S. Ouyang, T. T.-K. Lau, S. Lin, and Y. Yao, "Global convergence in deep learning with variable splitting via the Kurdyka-Łojasiewicz property," *arXiv preprint arXiv:1803.00225*, 2018.

[20] Z. Lin, R. Liu, and Z. Su, "Linearized alternating direction method with adaptive penalty for low-rank representation," in *Advances in Neural Information Processing Systems*, 2011, pp. 612–620.

[21] Z. Zhang, Y. Chen, and V. Saligrama, "Efficient training of very deep neural networks for supervised hashing," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1487–1495.

[22] Z. Zhang and M. Brand, "Convergent block coordinate descent for training Tikhonov regularized deep neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 1721–1730.

[23] A. Askari, G. Negiar, R. Sambharya, and L. E. Ghaoui, "Lifted neural networks," *arXiv preprint arXiv:1805.01532*, 2018.

[24] F. Gu, A. Askari, and L. E. Ghaoui, "Fenchel lifted networks: A Lagrange relaxation of neural network training," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, vol. 108, 2020, pp. 3362–3371.

[25] J. Li, C. Fang, and Z. Lin, "Lifted proximal operator machines," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4181–4188.

[26] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[27] E. Kreyszig, *Introductory Functional Analysis with Applications*. Wiley New York, 1978, vol. 1.

[28] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, pp. 183–202, 2009.

[29] X. Lian, H. Zhang, C.-J. Hsieh, Y. Huang, and J. Liu, "A comprehensive linear speedup analysis for asynchronous stochastic parallel optimization from zeroth-order to first-order," in *Advances in Neural Information Processing Systems*, 2016, pp. 3054–3062.

[30] T. Sun, R. Hannah, and W. Yin, "Asynchronous coordinate descent under more realistic assumptions," in *Advances in Neural Information Processing Systems*, 2017, pp. 6182–6190.

[31] D. P. Bertsekas and J. N. Tsitsiklis, Eds., *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989.

[32] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[33] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

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

[35] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[37] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai, "Gradient descent finds global minima of deep neural networks," *arXiv preprint arXiv:1811.03804*, 2018.

[38] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," *arXiv preprint arXiv:1811.03962*, 2018.

[39] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of ImageNet as an alternative to the CIFAR datasets," *arXiv preprint arXiv:1707.08819*, 2017.

[40] 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*, vol. 2011, no. 2, 2011, p. 5.

[41] J. Feng and Z.-H. Zhou, "Autoencoder by forest," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[42] 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.

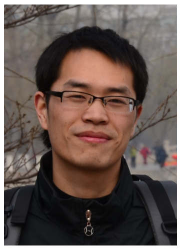
[43] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu, "Asynchronous stochastic gradient descent with delay compensation," in *Proceedings of the International Conference on Machine Learning*, vol. 108, 2017, pp. 4120–4129.

[44] D. P. Bertsekas, *Nonlinear Programming: 2nd Edition*. Athena Scientific, 1999.

[45] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM Journal on Imaging Sciences*, vol. 6, no. 3, pp. 1758–1789, 2013.

[46] G. H. Golub and C. F. Van Loan, *Matrix Computations*. The Johns Hopkins University Press, 2012, vol. 3.

[47] Y. Nesterov, Ed., *Introductory Lectures on Convex Optimization: A Basic Course*. Springer, 2004.



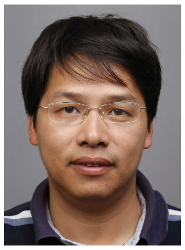
Jia Li received his Ph.D. degree in computer science from Beijing Jiaotong University in 2017. He was a Postdoctoral Researcher at Peking University from 2018 to 2020. He is currently a Lecturer in the School of Artificial Intelligence, Beijing Normal University. His research interests include machine learning, computer vision, and image processing.

Mingqing Xiao received the B.S. degree in computer science and technology and the double B.S. degree in psychology from Peking University in 2020. He is currently pursuing the Ph.D. degree in the School of Electronics Engineering and Computer Science, Peking University. His research interests include machine learning and computer vision.

Cong Fang received his Ph.D. degree from Peking University in 2019. He was a Postdoctoral Researcher at Princeton University from 2019 to 2020. He is currently a Postdoctoral Researcher at University of Pennsylvania. His research interests include optimization, machine learning, computer vision, and pattern recognition.

Yue Dai is currently an undergraduate student in the College of Software, Beihang University, where he is working for a double degree in mathematics. His research interests include machine learning and computer vision.

Chao Xu received the BE degree from Tsinghua University, in 1988, the MS degree from the University of Science and Technology of China, in 1991, and the PhD degree from the Institute of Electronics, Chinese Academy of Sciences, in 1997. Between 1991 and 1994 he was employed as an assistant professor with the University of Science and Technology of China. Since 1997, he has been with School of EECS at Peking University where he is currently a professor. His research interests include image and video coding, processing and understanding. He has authored or co-authored more than 80 publications and 5 patents in these fields.



Zhouchen Lin (M'00-SM'08-F'18) received the PhD degree from Peking University in 2000. He is currently a professor with the Key Laboratory of Machine Perception, School of EECS, Peking University. His research interests include computer vision, image processing, machine learning, pattern recognition, and numerical optimization. He is an area chair of CVPR 2014/16/19/20/21, ICCV 2015, NIPS/NeurIPS 2015/18/19/20, AAAI 2019/20, IJCAI 2020/21, ICLR 2021 and ICML 2020. He was an associate editor of the IEEE Transactions on Pattern Analysis and Machine Intelligence and currently is an associate editor of the International Journal of Computer Vision. He is a Fellow of IAPR and IEEE.