Lifted Proximal Operator Machines

Jia Li Cong Fang Zhouchen Lin[*]
Key Laboratory of Machine Perception (MOE), School of EECS, Peking University, P. R. China
jiali.gm@gmail.com fangcong@pku.edu.cn zlin@pku.edu.cn

Abstract

We propose a new optimization method for training feed-forward neural networks. By rewriting the activation function as an equivalent proximal operator, we approximate a feed-forward neural network by adding the proximal operators to the objective function as penalties, hence we call the lifted proximal operator machine (LPOM). LPOM is *block multi-convex* in all layer-wise weights and activations. This allows us to use block coordinate descent to update the layer-wise weights and activations. Most notably, we only use the mapping of the activation function *itself*, rather than its derivative, thus avoiding the gradient vanishing or blow-up issues in gradient based training methods. So our method is applicable to various non-decreasing Lipschitz continuous activation functions, which *can be saturating and non-differentiable*. LPOM does not require more auxiliary variables than the layer-wise activations, thus using roughly the same amount of memory as stochastic gradient descent (SGD) does. Its parameter tuning is also much simpler. We further prove the convergence of updating the layer-wise weights and activations and point out that the optimization could be made parallel by asynchronous update. Experiments on MNIST and CIFAR-10 datasets testify to the advantages of LPOM.

Introduction

Feed-forward deep neural networks (DNNs) are cascades of fully connected layers and there are no feedback connections. In recent years, with the advances in hardware and dataset sizes, feed-forward DNNs have become standard in many tasks, such as image recognition (Krizhevsky, Sutskever, and Hinton, 2012), speech recognition (Hinton et al., 2012), natural language understanding (Collobert et al., 2011), and building the Go game learning system (Silver et al., 2016).

For several decades, training a DNN is accomplished by optimizing a highly nonconvex and nested function of the network weights. The predominant method for training DNNs is stochastic gradient descent (SGD) (Rumelhart, Hinton, and Williams, 1986), whose effectiveness has been demonstrated by the successes of DNNs in various real-world applications. Recently, many variants of SGD have been proposed, which use adaptive learning rates and momentum terms, e.g., Nesterov momentum (Sutskever et al., 2013), AdaGrad (Duchi, Hazan, and Singer, 2011), RMSProp (Dauphin, de Vries, and Bengio, 2015), and Adam (Kingma and Ba, 2014). SGD and its variants use a few training samples to estimate the full gradient, making the computational complexity of each iteration small. Moreover, the estimated gradients have noise, which is helpful for escaping saddle points (Ge et al., 2015). However, they have some drawbacks as well. One major problem is the vanishing or blow-up gradient issue, where the magnitudes of gradients decrease or increase exponentially with the number of layers. This causes slow or unstable convergence, especially in very deep networks. This flaw can be remitted by using non-saturating activation functions, such as rectified linear unit (ReLU), and modified network architectures, such as ResNet (He et al., 2016). However, the fundamental problem remains. Also, their parameters are difficult to tune (e.g., learning rates and convergence criteria) (Le et al., 2011). Furthermore, they cannot deal with non-differentiable activation functions directly (e.g., binarized neural networks (Hubara et al., 2016)) and do not allow parallel weight updates across the layers (Le et al., 2011). For more on the limitations of SGD, please refer to (Taylor et al., 2016) and (Le et al., 2011).

The drawbacks of SGD motivate research on alternative training methods for DNNs. Recently, training a feed-forward neural network is formulated as a constrained optimization problem, where the network activations are introduced as auxiliary variables and the network configuration is guaranteed by layer-wise constraints (Carreira-Perpinan and Wang, 2014). It breaks the dependency among the nested functions into equality constraints, so many standard optimization methods can be utilized. Some methods of this type of approach were studied and they differed in how to handle the equality constraints. Carreira-Perpinan and Wang (2014) approximated the equality constraints via quadratic penalties and alternately optimized network weights and activations. Zeng et al. (2018) introduced one more block of auxiliary variables per layer and also approximated the equality constraints via quadratic penalties. Inspired by alternating direction method of multiplier (ADMM) (Lin, Liu, and Su, 2011), Taylor et al. (2016) and Zhang, Chen, and Saligrama (2016) used the augmented Lagrangian approach to obtain exact enforcement of the equality constraints. However, the two methods involved the Lagrange multipliers and nonlinear

[*]Corresponding author.

constraints, thus were more memory demanding and more difficult in optimization. Motivated by the fact that the ReLU activation function is equivalent to a simple constrained convex minimization problem, Zhang and Brand (2017) relaxed the nonlinear constraints as penalties, which encode the network architecture and the ReLU activation function. Thus, the nonlinear constraints no longer exist. However, their approach is limited to the ReLU function and does not apply to other activation functions. Askari et al. (2018) followed this idea by considering more complex convex optimization problems and discussed several types of non-decreasing activation functions. However, their methods to update the weights and activations are still limited to the ReLU function. Their approach cannot outperform SGD and can only serve for initializing SGD. Actually, we have found that their formulation was incorrect (see Subsection "Advantages of LPOM").

This paper makes the following contributions:

- We propose a new formulation to train feed-forward DNNs, which we call the lifted proximal operator machine (LPOM)[1]. LPOM is block multi-convex, i.e., the problem is convex w.r.t. weights or activations of each layer when the remaining weights and activations are fixed. In contrast, almost all existing DNN training methods do not have such a property. This greatly facilitates the training of DNNs.
- Accordingly, we apply block coordinate descent (BCD) to solve LPOM. Most notably, the update of the layer-wise weights or activations only utilizes the activation function itself, rather than its derivative, thus avoiding the gradient vanishing or blow-up issues in gradient based training methods. Moreover, LPOM does not need more auxiliary variables than the layer-wise activations, thus its memory cost is close to that of SGD. It is also easy to tune the penalties in LPOM. We further prove that the iterations to update layer-wise weights or activations are convergent.
- Since only the activation function itself is involved in computation, LPOM is able to handle general non-decreasing Lipschitz continuous activation functions, which can be saturating (such as sigmoid and tanh) and non-differentiable (such as ReLU and leaky ReLU). So LPOM successfully overcomes the computational difficulties when using most of existing activation functions.

We implement LPOM on fully connected DNNs and test it on benchmark datasets, MNIST and CIFAR-10, and obtain satisfactory results. For convolutional neural networks (CNNs), since we have not reformulated pooling and skip-connections, we leave the implementation of LPOM on CNNs to future work. Note that the existing non-gradient based approaches also focus on fully connected DNNs first (Carreira-Perpinan and Wang, 2014; Zeng et al., 2018; Taylor et al., 2016; Zhang, Chen, and Saligrama, 2016; Zhang and Brand, 2017; Askari et al., 2018). We also point out that LPOM could be solved in parallel by asynchronous update.

[1]Two patents were filed in Nov. 2017 and Oct. 2018, respectively.

Related Work

The optimization problem for training a standard feed-forward neural network is:

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

where $X^1 \in \mathbb{R}^{n_1\times m}$ is a batch of training samples, $L \in \mathbb{R}^{c\times m}$ denotes the corresponding labels, n_1 is the dimension of the training samples, m is the batch size, c is the number of classes, $\{W^i\}_{i=1}^{n-1}$ are the weights to be learned in which the biases have been omitted for simplicity, $\phi(\cdot)$ is an element-wise activation function (e.g., sigmoid, tanh, and ReLU), and $\ell(\cdot,\cdot)$ is the loss function (e.g., the least-square error or the cross-entropy error). Here the neural network is defined as a nested function, where the first layer function of the neural network is $\phi(W^1X^1)$, the i-th layer ($i=2,\cdots,n$) function has the form $\phi(W^iX)$, and X is the output of the $(i-1)$-th layer function. A common approach to optimize (1) is by SGD, i.e., calculating the gradient w.r.t. all weights of the network using backpropagation and then updating the weights by gradient descent.

By introducing the layer-wise activations as a block of auxiliary variables, the training of a neural network can be equivalently formulated as an equality constrained optimization problem (Carreira-Perpinan and Wang, 2014):

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

where X^i is the activation of the i-th layer and other notations are the same as those in (1). The constraints in (2) ensure that the auxiliary variables $\{X^i\}_{i=2}^n$ exactly match the forward pass of the network. Compared with problem (1), problem (2) is constrained. But since the objective function is not nested, hence much simpler, such an equivalent reformulation may lead to more flexible optimization methods. Note that when using SGD to solve problem (1), it actually works on problem (2) implicitly as *the activations $\{X^i\}_{i=2}^n$ need be recorded in order to compute the gradient.*

Inspired by the quadratic-penalty method, Carreira-Perpinan and Wang (2014) developed the method of auxiliary coordinates (MAC) to solve problem (2). MAC uses quadratic penalties to approximately enforce equality constraints and tries to solve the following problem:

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

where $\mu > 0$ is a constant that controls the weight of the constraints and $\|\cdot\|_F$ is the Frobenius norm. Zeng et al. (2018) decoupled the nonlinear activations in (2) with new auxiliary variables:

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

This is called as the 3-splitting formulation. Accordingly, problem (2) is the 2-splitting formulation. Following the

MAC method, rather than directly solving problem (4), they optimized the following problem instead:

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

They adapted a BCD method to solve the above problem.

Taylor et al. (2016) also considered solving problem (4). Inspired by ADMM (Lin, Liu, and Su, 2011), they added a Lagrange multiplier to the output layer, which yields

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

where M is the Lagrange multiplier and $\beta > 0$ and $\mu_i > 0$ are constants. Note that the activation function on the output layer is absent. So (6) is only a heuristic adaptation of ADMM. Zhang, Chen, and Saligrama (2016) adopted a similar technique but used a different variable splitting scheme:

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

Despite the nonlinear equality constraints, which ADMM is not designed to handle, they added a Lagrange multiplier for each constraint in (7). Then the augmented Lagrangian problem is as follows:

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

where A^i and B^i are the Lagrange multipliers.

Different from naively applying the penalty method and ADMM, Zhang and Brand (2017) interpreted the ReLU activation function as a simple smooth convex optimization problem. Namely, the equality constraints in problem (2) using the ReLU activation function can be rewritten as a convex minimization 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 $\mathbf{0}$ is a zero matrix with an appropriate size. Based on this observation, they approximated problem (2) with the activation function being ReLU in the following way:

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

where the penalty terms encode both the network structure and activation function. Unlike MAC and ADMM based methods, it does not include nonlinear activations. Moreover, the major advantage is that problem (10) is block multi-convex, i.e., the problem is convex w.r.t. each block of variables when the remaining blocks are fixed. They developed a new BCD method to solve it. They also empirically demonstrated the superiority of the proposed approach over SGD based solvers in Caffe (Jia et al., 2014) and the ADMM based method (Zhang, Chen, and Saligrama, 2016). Askari et al. (2018) inherited the same idea. By introducing a more complex convex minimization problem, they could handle more general activation functions.

Lifted Proximal Operator Machine

In this section, we describe our basic idea of LPOM and its advantages over existing DNN training methods.

Reformulation by Proximal Operator

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 $\phi(y)$. We want to construct an objective function $h(x, y)$, parameterized by y, such that its minimizer is exactly $x = \phi(y)$. Accordingly, we may replace the constraint $x = \phi(y)$ by minimizing $h(x, y)$, which can be added to the loss of DNNs as a penalty.

Since the proximal operator (Parikh and Boyd, 2014)

$$\text{prox}_f(y) = \operatorname*{argmin}_x f(x) + \frac{1}{2}(x - y)^2, \tag{11}$$

is commonly used in optimization, we consider using the proximal operator to construct the optimization problem. 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, ϕ^{-1}, f, and g (to be defined later) will *not* be explicitly used in our computation. It is easy to show that the optimality condition of (11) is $0 \in (\phi^{-1}(x) - x) + (x - y)$. So the solution to (11) is exactly $x = \phi(y)$.

Note that $f(x)$ is a univariate function. For a matrix $X = (X_{kl})$, we define $f(X) = (f(X_{kl}))$. Then the optimality condition of the following minimization problem:

$$\operatorname*{argmin}_{X^i} \mathbf{1}^T f(X^i)\mathbf{1} + \frac{1}{2}\|X^i - W^{i-1}X^{i-1}\|_F^2, \tag{12}$$

where $\mathbf{1}$ is an all-one column vector, is

$$\mathbf{0} \in \phi^{-1}(X^i) - W^{i-1}X^{i-1}, \tag{13}$$

where $\phi^{-1}(X^i)$ is also defined element-wise. So the optimal solution to (12) is

$$X^i = \phi(W^{i-1}X^{i-1}), \tag{14}$$

which is exactly the constraint in problem (2). So we may approximate problem (2) naively as:

$$\begin{aligned}\min_{\{W^i\},\{X^i\}} & \ell(X^n, L) \\ &+ \sum_{i=2}^{n}\mu_i\left(\mathbf{1}^T f(X^i)\mathbf{1} + \frac{1}{2}\|X^i - W^{i-1}X^{i-1}\|_F^2\right).\end{aligned} \tag{15}$$

Table 1: The $f(x)$ and $g(x)$ of several representative activation functions. Note that $0<\alpha<1$ for the leaky ReLU function and $\alpha>0$ for the exponential linear unit (ELU) function. We only use $\phi(x)$ 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	$\begin{cases} x, & x\geq 0 \\ \alpha(e^x-1), & x<0\end{cases}$	$\begin{cases} x, & x\geq 0 \\ \log(1+\frac{x}{\alpha}), & x<0\end{cases}$	$\begin{cases} 0, & x\geq 0 \\ (\alpha+x)(\log(\frac{x}{\alpha}+1)-1)-\frac{x^2}{2}, & x<0\end{cases}$	$\begin{cases} 0, & x\geq 0 \\ \alpha(e^x-x)-\frac{x^2}{2}, & x<0\end{cases}$
softplus	$\log(1+e^x)$	$\log(e^x-1)$	No analytic expression	No analytic expression

However, its optimality conditions for $\{X^i\}_{i=2}^{n-1}$ are:

$$\begin{aligned}\mathbf{0}\in&\mu_i(\phi^{-1}(X^i)-W^{i-1}X^{i-1})\\ &+\mu_{i+1}(W^i)^T(W^iX^i-X^{i+1}),\ i=2,\cdots,n-1.\end{aligned}\tag{16}$$

We can clearly see that the equality constraints (14) in problem (2) do *not* satisfy the above!

In order that the equality constraints (14) fulfill the optimality conditions of the approximating problem, which is necessary if we want to use the simple feed-forward process to infer new samples, we need to modify (16) as

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

This corresponds to the following problem:

$$\begin{aligned}\min_{\{W^i\},\{X^i\}}\ &\ell(X^n,L)+\sum_{i=2}^{n}\mu_i\Bigg(\mathbf{1}^Tf(X^i)\mathbf{1}\\ &+\mathbf{1}^Tg(W^{i-1}X^{i-1})\mathbf{1}+\frac{1}{2}\|X^i-W^{i-1}X^{i-1}\|_F^2\Bigg),\end{aligned}\tag{18}$$

where

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

$g(X)$ is also defined element-wise for a matrix X. The $f(x)$'s and $g(x)$'s of some representative activation functions are shown in Table 1. (18) is the formulation of our proposed LPOM, where we highlight that the introduction of g is nontrivial and non-obvious.

Advantages of LPOM

Denote the objective function of LPOM in (18) as $F(W,X)$. Then we have the following theorem:

Theorem 1 *Suppose $\ell(X^n,L)$ is convex in X^n and ϕ is non-decreasing. Then $F(W,X)$ is block multi-convex, i.e., convex in each X^i and W^i if all other blocks of variables are fixed.*

Proof. $F(W,X)$ can be simplified to

$$\begin{aligned}F(W,X)=\ &\ell(X^n,L)+\sum_{i=2}^{n}\mu_i\Big(\mathbf{1}^T\tilde{f}(X^i)\mathbf{1}\\ &+\mathbf{1}^T\tilde{g}(W^{i-1}X^{i-1})\mathbf{1}-\langle X^i,W^{i-1}X^{i-1}\rangle\Big),\end{aligned}\tag{19}$$

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 verify 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 remaining 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. □

Theorem 1 allows for efficient BCD algorithms to solve LPOM and guarantees that the optimal solutions for updating X^i and W^i can be obtained, due to the convexity of subproblems. In contrast, the subproblems in the penalty and the ADMM based methods are all nonconvex.

When compared with ADMM based methods (Taylor et al., 2016; Zhang, Chen, and Saligrama, 2016), LPOM does not require Lagrange multipliers and more auxiliary variables than $\{X^i\}_{i=2}^n$. Moreover, we have designed delicate algorithms so that no auxiliary variables are needed either when solving LPOM (see Section "Solving LPOM"). So LPOM has much less variables than ADMM based methods and hence saves memory greatly. Actually, its memory cost is close to that of SGD as SGD needs to save $\{X^i\}_{i=2}^n$.

When compared with the penalty methods (Carreira-Perpinan and Wang, 2014; Zeng et al., 2018), 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 (17) and

$$(\phi(W^iX^i)-X^{i+1})(X^i)^T=\mathbf{0},\quad i=1,\cdots,n-1,\tag{20}$$

while those for MAC are

$$\begin{aligned}&(X^i-\phi(W^{i-1}X^{i-1}))\\ &+(W^i)^T[(\phi(W^iX^i)-X^{i+1})\circ\phi'(W^iX^i)]=\mathbf{0},\\ &i=2,\cdots,n-1.\end{aligned}\tag{21}$$

and

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

where $\circ$ denotes the element-wise multiplication. We can see that the optimality conditions for MAC have extra $\phi'(W^iX^i)$, which is nonlinear. The optimality conditions for Zeng et al. (2018) can be found in Supplementary Materials. They also have an extra $\phi'(U^i)$. This may imply that the solution sets of MAC and (Zeng et al., 2018) are more complex and also "larger" than that of LPOM. So it may be easier to find good solutions of LPOM.

When compared with the convex optimization reformulation methods (Zhang and Brand, 2017; Askari et al., 2018), LPOM can handle much more general activation functions. Note that Zhang and Brand (2017) only considered ReLU. Although Askari et al. (2018) claimed that their formulation can handle general activation functions, its solution method was still restricted to ReLU. Moreover, Askari et al. (2018) do not have a correct reformulation as its optimality conditions for $\{X^i\}_{i=2}^{n-1}$ and $\{W^i\}_{i=1}^{n-1}$ are

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

and

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

respectively. It is clear that the equality constraints (14) do not satisfy the above. Moreover, somehow Askari et al. (2018) further added extra constraints $X^i\geq\mathbf{0}$, no matter what the activation function is. So their reformulation cannot approximate the original DNN (2) well. This may explain why Askari et al. (2018) could not obtain good results. Actually, they can only provide good initialization for SGD.

When compared with gradient based methods, such as SGD, LPOM can work with any non-decreasing Lipschitz continuous activation function without numerical 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 in an asynchronous way (see next section)[2]. In contrast, gradient based methods can only work with limited activation functions, such as ReLU, leaky ReLU, and softplus, in order to avoid the gradient vanishing or blow-up issues, and they cannot be parallelized when computing the gradient and the activations. Moreover, gradient based methods require much parameter tuning, which is difficult (Le et al., 2011), while the tuning of penalties μ_i's in LPOM is much simpler.

Solving LPOM

Thanks to the block multi-convexity (Theorem 1), LPOM can be solved by BCD. Namely, we update X^i or W^i by fixing all other blocks of variables. The optimization can be performed using a mini-batch of training samples, as summarized in Algorithm 1. Below we give more details.

[2]But our current implementation is still serial.

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 $L$.
    Solve $\{X^i\}_{i=2}^{n-1}$ by iterating Eq. (25) for $K_1$ times (or until convergence).
    Solve $X^n$ by iterating Eq. (28) for $K_1$ times.
    Solve $\{W^i\}_{i=1}^{n-1}$ by applying Algorithm 2 to (30).
  end for
Output: $\{W^i\}_{i=1}^{n-1}$.
```

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

We first introduce the serial method for updating $\{X^i\}_{i=2}^n$. We update $\{X^i\}_{i=2}^n$ from $i=2$ to n successively, just like the feed-forward process of DNNs. 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 (18) reduces to

$$\begin{aligned}\min_{X^i}\ &\mu_i\left(\mathbf{1}^Tf(X^i)\mathbf{1}+\frac{1}{2}\|X^i-W^{i-1}X^{i-1}\|_F^2\right)\\ &+\mu_{i+1}\left(\mathbf{1}^Tg(W^iX^i)\mathbf{1}+\frac{1}{2}\|X^{i+1}-W^iX^i\|_F^2\right).\end{aligned} \tag{23}$$

The optimality condition is:

$$\begin{aligned}\mathbf{0}\in&\mu_i(\phi^{-1}(X^i)-W^{i-1}X^{i-1})\\ &+\mu_{i+1}((W^i)^T(\phi(W^iX^i)-X^{i+1})).\end{aligned} \tag{24}$$

Based on fixed-point iteration (Kreyszig, 1978) and in order to avoid using ϕ^{-1}, we may update X^i by iterating

$$X^{i,t+1}=\phi\left(W^{i-1}X^{i-1}-\frac{\mu_{i+1}}{\mu_i}(W^i)^T(\phi(W^iX^{i,t})-X^{i+1})\right) \tag{25}$$

until convergence, where the superscript t is the iteration number. The convergence analysis is as follows[3]:

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

In the above, $|A|$ is a matrix whose entries are the absolute values of A, $\|\cdot\|_1$ and $\|\cdot\|_\infty$ are the matrix 1-norm (largest absolute column sum) and the matrix ∞-norm (largest absolute row sum), respectively. 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 (18) reduces to

$$\min_{X^n}\ell(X^n,L)+\mu_n\left(\mathbf{1}^Tf(X^i)\mathbf{1}+\frac{1}{2}\|X^n-W^{n-1}X^{n-1}\|_F^2\right). \tag{26}$$

[3]The proofs of theorems can be found in Supplementary Materials.

Assume that the loss function is differentiable w.r.t. X^n. The optimality condition is:

$$\mathbf{0} \in \frac{\partial \ell(X^n, L)}{\partial X^n} + \mu_n(\phi^{-1}(X^n) - W^{n-1}X^{n-1}). \quad (27)$$

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

$$X^{n,t+1} = \phi\left(W^{n-1}X^{n-1} - \frac{1}{\mu_n}\frac{\partial \ell(X^{n,t}, L)}{\partial X^n}\right) \quad (28)$$

until convergence. The convergence analysis is as follows:

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

If $\ell(X^n, L)$ is the least-square error, i.e., $\ell(X^n, L) = \frac{1}{2}\|X^n - L\|_F^2$, then $\left|\left|\left|\left(\frac{\partial^2 \ell(X,L)}{\partial X_{kl}\partial X_{pq}}\right)\right|\right|\right|_1 = 1$. So we obtain $\mu_n > \gamma$.

The above serial update procedure can be easily changed to asynchronously parallel update: each X^i is updated using the latest information of other X^j's, $j \neq i$.[4]

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

$\{W^i\}_{i=1}^{n-1}$ can be updated with full parallelization. When $\{X^i\}_{i=2}^{n}$ are fixed, problem (18) reduces to

$$\min_{W^i} \mathbf{1}^T g(W^iX^i)\mathbf{1} + \frac{1}{2}\|W^iX^i - X^{i+1}\|_F^2, \; i = 1, \cdots, n-1, \quad (29)$$

which can be solved in parallel. (29) can be rewritten as

$$\min_{W^i} \mathbf{1}^T \tilde{g}(W^iX^i)\mathbf{1} - \langle X^{i+1}, W^iX^i\rangle, \quad (30)$$

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 activation functions in use. Then $\tilde{g}(x)$ is β-smooth:

$$|\tilde{g}'(x) - \tilde{g}'(y)| = |\phi(x) - \phi(y)| \le \beta|x - y|. \quad (31)$$

Problem (30) could be solved by APG (Beck and Teboulle, 2009) by locally linearizing $\hat{g}(W) = \tilde{g}(WX)$. However, the Lipschitz constant of the gradient of $\hat{g}(W)$, which is $\beta\|X\|_2^2$, can be very large, hence the convergence can be slow. Below we propose a variant of APG that is tailored for solving (30) much more efficiently.

Consider the following problem:

$$\min_x F(x) \equiv \varphi(Ax) + h(x), \quad (32)$$

where both $\varphi(y)$ and $h(x)$ are convex. Moreover, $\varphi(y)$ is L_φ-smooth: $\|\nabla\varphi(x) - \nabla\varphi(y)\| \le L_\varphi\|x - y\|, \forall x, y$. We assume that the following problem

$$x_{k+1} = \operatorname*{argmin}_x \langle \nabla\varphi(Ay_k), A(x - y_k)\rangle + \frac{L_\varphi}{2}\|A(x - y_k)\|^2 + h(x) \quad (33)$$

[4]The convergence can be proven when the objective function is augmented with a proximal term $\frac{\sigma}{2}\|X^i - X^{i,0}\|_F^2$, where $X^{i,0}$ is chosen as the last update of X^i. As the implementation and analysis of asynchronously parallel update deserve an independent paper, we choose not to squeeze them in this paper.

Algorithm 2: Solving (32).

```
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 (33).
  end for
Output: $x_k$.
```

is easy to solve for any given y_k. We propose Algorithm 2 to solve (32), which naturally comes from the proof of its convergence theorem:

Theorem 4 *If we use Algorithm 2 to solve problem (32), then the convergence rate is at least $O(k^{-2})$:*

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

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 (32).

Problem (30) is an instantiation of (32). Accordingly, the instantiation of subproblem (33) is as follows:

$$\begin{aligned} W^{i,t+1} = \operatorname*{argmin}_W & \langle \phi(Y^{i,t}X^i), (W - Y^{i,t})X^i\rangle \\ & + \frac{\beta}{2}\|(W - Y^{i,t})X^i\|_F^2 - \langle X^{i+1}, WX^i\rangle. \end{aligned} \quad (34)$$

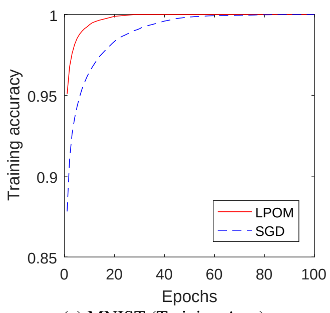
It is a least-square problem and the solution is:

$$W^{i,t+1} = Y^{i,t} - \frac{1}{\beta}(\phi(Y^{i,t}X^i) - X^{i+1})(X^i)^\dagger, \quad (35)$$

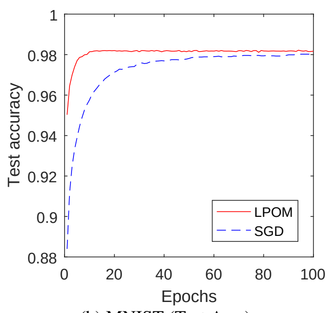
where $(X^i)^\dagger$ is the pseudo-inverse of X^i and $Y^{i,t}$ plays the role of y_k in Algorithm 2.

Experiments

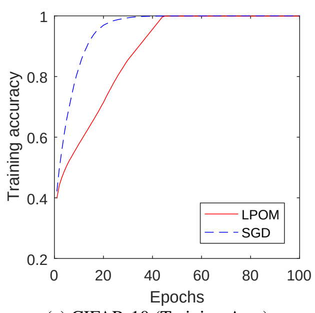
We evaluate LPOM by comparing with SGD and two non-gradient based methods (Askari et al., 2018; Taylor et al., 2016). The other non-gradient based methods do not train fully connected feed-forward neural networks for classification tasks (e.g., using skip connections (Zhang and Brand, 2017), training autoencoders (Carreira-Perpinan and Wang, 2014), and learning for hashing (Zhang, Chen, and Saligrama, 2016)). So we cannot include them for comparison. For simplicity, we utilize the least-square loss function and the ReLU activation function unless specified otherwise. Unlike (Askari et al., 2018), we do not use any regularization on the weights $\{W^i\}_{i=1}^{n-1}$. We run LPOM and SGD with the same inputs and random initializations (Glorot and Bengio, 2010). We implement LPOM with MATLAB without optimizing the code. We use the SGD based solver in Caffe (Jia et al., 2014). For the Caffe solver, we modify the demo code and carefully tune the parameters to achieve the best performances. For (Askari et al., 2018), we quote their results. For (Taylor et al., 2016), we read the results from Fig.1 (b) of the paper.



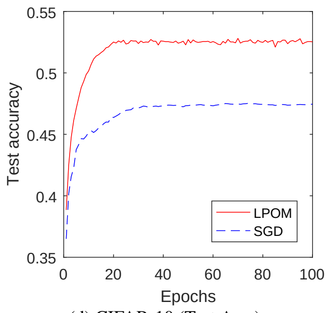
(a) MNIST (Training Acc.)



(b) MNIST (Test Acc.)



(c) CIFAR-10 (Training Acc.)



(d) CIFAR-10 (Test Acc.)

Figure 1: Comparison of LPOM and SGD on the MNIST and the CIFAR-10 datasets.

Table 2: Comparison of accuracies of LPOM and (Askari et al., 2018) on the MNIST dataset using different networks.

Hidden layers	300	300-100	500-150	500-200-100	400-200-100-50
(Askari et al., 2018)	89.8%	87.5%	86.5%	85.3%	77.0%
LPOM	97.7%	96.9%	97.1%	96.2%	96.1%

Table 3: Comparison with SGD and (Taylor et al., 2016) on the SVHD dataset.

SGD	95.0%
(Taylor et al., 2016)	96.5%
LPOM	98.3%

Comparison with SGD

We conduct experiments on two datasets, i.e., MNIST [5] and CIFAR-10 (Krizhevsky and Hinton, 2009). For the MNIST dataset, we use $28 \times 28 = 784$ raw pixels as the inputs. It includes 60,000 training images and 10,000 test images. We do not use pre-processing or data augmentation. For LPOM and SGD, in each epoch the entire training samples are passed through once. The performance depends the choice of network architecture. Following (Zeng et al., 2018), we implement a 784-2048-2048-2048-10 feed-forward neural network. For LPOM, we simply set $\mu_i = 20$ in (18). We run LPOM and SGD for 100 epochs with a fixed batch size 100. The training and test accuracies are shown in Fig. 1 (a) and (b). We can see that the training accuracies of the two methods are both approximately equal to 100%. However, the test accuracy of LPOM is slightly better than that of SGD (98.2% vs. 98.0%).

For the CIFAR-10 dataset, as in (Zeng et al., 2018) we implement a 3072-4000-1000-4000-10 feed-forward neural network. We normalize color images by subtracting the training dataset's means of the red, green, and blue channels, respectively. We do not use pre-processing or data augmentation. For LPOM, we set $\mu_i = 100$ in (18). We run LPOM and SGD for 100 epochs with a fixed batch size 100. The training and test accuracies are shown in Fig. 1 (c) and (d). We can see that the training accuracies of SGD and LPOM are approximately equal to 100%. However, the test accuracy of LPOM is better than that of SGD (52.5% vs. 47.5%).

Comparison with Other Non-gradient Based Methods

We compare against (Askari et al., 2018) with identical architectures on the MNIST dataset. Askari et al. (2018) only use the ReLU activation function in real computation. As in (Askari et al., 2018), we run LPOM for 17 epochs with a fixed batch size 100. For LPOM, we set $\mu_i = 20$ for all the networks. We do not use pre-processing or data augmentation. The test accuracies of the two methods are shown in Table 2. We can see that LPOM with the ReLU activation function performs better than (Askari et al., 2018) with significant gaps. This complies with our analysis in Subsection "Advantages of LPOM".

Following the settings of dataset and network architecture in (Taylor et al., 2016), we test LPOM on the Street View House Numbers (SVHN) dataset (Netzer et al., 2011). For LPOM, we set $\mu_i = 20$ in (18). The test accuracies of SGD, (Taylor et al., 2016), and LPOM are shown in Table 3. We can see that LPOM outperforms SGD and (Taylor et al., 2016). This further verifies the advantage of LPOM.

Conclusions

In this work we have proposed LPOM to train fully connected feed-forward neural networks. Using the proximal operator, LPOM transforms the neural network into a new block multi-convex model. The transformation works for general non-decreasing Lipschitz continuous activation functions. We apply the block coordinate descent algorithm to solve LPOM, where each subproblem has convergence guarantee. LPOM does not require more auxiliary variables than the layer-wise activations and its penalties are relatively easy to tune. It could also be solved in parallel in an asynchronous way. Our experimental results show that LPOM works better than SGD, (Askari et al., 2018), and (Taylor et al., 2016) on fully connected neural networks. Future work includes extending LPOM to train convolutional and recurrent neural networks and applying LPOM to network quantization.

Acknowledgements

Z. Lin is supported by 973 Program of China (grant no. 2015CB352502), NSF of China (grant nos. 61625301 and 61731018), and Microsoft Research Asia.

[5] `http://yann.lecun.com/exdb/mnist/`

References

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

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

Carreira-Perpinan, M., and Wang, W. 2014. Distributed optimization of deeply nested systems. In *International Conference on Artificial Intelligence and Statistics*, 10–19.

Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; and Kuksa, P. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12:2493–2537.

Dauphin, Y.; de Vries, H.; and Bengio, Y. 2015. Equilibrated adaptive learning rates for non-convex optimization. In *Advances in Neural Information Processing Systems*, 1504–1512.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12:2121–2159.

Ge, R.; Huang, F.; Jin, C.; and Yuan, Y. 2015. Escaping from saddle points-online stochastic gradient for tensor decomposition. In *Conference on Learning Theory*, 797–842.

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256.

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

Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.-R.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T. N.; et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29(6):82–97.

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

Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; and Darrell, T. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*, 675–678. ACM.

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

Kreyszig, E. 1978. *Introductory Functional Analysis with Applications*, volume 1. Wiley New York.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 1097–1105.

Le, Q. V.; Ngiam, J.; Coates, A.; Lahiri, A.; Prochnow, B.; and Ng, A. Y. 2011. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, 265–272. Omnipress.

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

Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; and Ng, A. Y. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on Deep Learning and Unsupervised Feature Learning*, volume 2011, 5.

Parikh, N., and Boyd, S. 2014. Proximal algorithms. *Foundations and Trends® in Optimization* 1(3):127–239.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning representations by back-propagating errors. *Nature* 323(6088):533.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587):484.

Sutskever, I.; Martens, J.; Dahl, G.; and Hinton, G. 2013. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, 1139–1147.

Taylor, G.; Burmeister, R.; Xu, Z.; Singh, B.; Patel, A.; and Goldstein, T. 2016. Training neural networks without gradients: A scalable ADMM approach. In *International Conference on Machine Learning*, 2722–2731.

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

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

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