PROX-SGD: TRAINING STRUCTURED NEURAL NETWORKS UNDER REGULARIZATION AND CONSTRAINTS

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we consider the problem of training structured neural networks (NN) with nonsmooth regularization (e.g. ℓ_1-norm) and constraints (e.g. interval constraints). We formulate training as a constrained nonsmooth nonconvex optimization problem, and propose a convergent proximal-type stochastic gradient descent (Prox-SGD) algorithm. We show that under properly selected learning rates, with probability 1, every limit point of the sequence generated by the proposed Prox-SGD is a stationary point. Finally we demonstrate its flexibility, showing how Prox-SGD can be used to train either sparse or binary neural networks through an adequate selection of the regularization function and constraint set, supporting the theoretical analysis by extensive numerical tests.

1 INTRODUCTION

In this paper, we consider the problem of training neural networks (NN) under constraints and regularization. It is formulated as an optimization problem

$$\underset{\boldsymbol{x}\in\mathbb{X}\subseteq\mathbb{R}^n}{\text{minimize}} \underbrace{\sum_{i=1}^{m} f_i(\boldsymbol{x},\boldsymbol{y}_i)}_{\triangleq f(\boldsymbol{x})} + r(\boldsymbol{x}), \tag{1}$$

where $\boldsymbol{x}$ is the parameter vector to optimize, $\boldsymbol{y}_i$ is the i-th training example which consists of a pair of training input and desired output, and m is the number of training examples. The training loss f is assumed to be smooth but nonconvex with respect to $\boldsymbol{x}$, the regularization r is assumed to be convex, proper and lower semicontinuous, and $\mathbb{X}$ is a convex, closed and bounded constraint set.

When $r(\boldsymbol{x}) = 0$ and $\mathbb{X} = \mathbb{R}^n$, stochastic gradient descent (SGD) has been used to solve the optimization problem (1). At each iteration, a minibatch of the m training examples are drawn randomly, and the obtained gradient is an unbiased estimate of the true gradient. Therefore SGD generally moves along the descent direction, see Bertsekas & Tsitsiklis (2000). SGD can be accelerated by replacing the instantaneous gradient estimates by a momentum aggregating all gradient in past iterations. Despite the success and popularity of SGD with momentum, its convergence had been an open problem. Assuming f is convex, analyzing the convergence was first attempted in Kingma & Ba (2015) and later concluded in Reddi et al. (2018). The proof for a nonconvex f was later given in Chen et al. (2019); Lei et al. (2019).

In machine learning, the regularization function r is typically used to promote a certain structure in the optimal solution, for example sparsity as in, e.g., feature selection and compressed sensing, or a zero-mean-Gaussian prior on the parameters (Bach et al., 2011; Boyd et al., 2010). It can be interpreted as a penalty function since at the optimal point $\boldsymbol{x}^\star$ of problem (1), the value $r(\boldsymbol{x}^\star)$ will be small, provided that the regularization parameter μ is properly selected. One nominant example is the Tikhonov regularization $r(\boldsymbol{x}) = \mu\|\boldsymbol{x}\|_2^2$ for some predefined constant μ, and it can be used to alleviate the ill-conditioning and ensure that the magnitude of the weights will not become exceedingly large. Another commonly used regularization, the ℓ_1-norm where $r(\boldsymbol{x}) = \mu\|\boldsymbol{x}\|_1 = \mu\sum_{j=1}^n |x_j|$ (the convex surrogate of the ℓ_0-norm), would encourage a sparse solution. In the context of NN, it is used to (i) promote a sparse neural network (SNN) to alleviate overfitting and to allow a better generalization, (ii) accelerate the training process, and (iii) prune the network to reduce its complexity, see Louizos et al. (2018) and Gale et al. (2019) for more details.

Technically, it is difficult to analyze the regularizations as some commonly used convex regularizers are nonsmooth, for example, ℓ_1-norm. In current implementations of Tensorflow, the gradient of $|x|$ is simply set to 0 when $x = 0$. This amounts to the stochastic subgradient descent method and usually exhibits slow convergence. Other techniques to promote a SNN includes magnitude pruning and variational dropout, see Gale et al. (2019).

Although regularization can be interpreted as a constraint from the duality theory, sometimes it may still be more desirable to use explicit constraints, for example, $\sum x_j^2 \leq \alpha$, where the summation is over the weights on the same layer. This is useful when we already know how to choose α. Another example is the lower and upper bound on the weights, that is, $\boldsymbol{l} \leq \boldsymbol{w} \leq \boldsymbol{u}$ for some predefined $\boldsymbol{l}$ and $\boldsymbol{u}$. Compared with regularization, constraints do not encourage the weights to stay in a small neighborhood of the initial weight, see Chapter 7.2 of Goodfellow et al. (2016) for more details.

The set $\mathbb{X}$ models such explicit constraints, but it poses an additional challenge for stochastic gradient algorithms as the new weight obtained from the SGD method (with or without momentum) must be projected back to the set $\mathbb{X}$ to maintain its feasibility. However, projection is a nonlinear operator, so the unbiasedness of the random gradient would be lost.

In this paper, we propose a convergent proximal-type stochastic gradient algorithm (with momentum) to train neural networks under regularization and constraints. The convergence analysis is based on the framework developed in Ruszczyński (1980), and we present here the key technical ideas. It turns out momentum, known as gradient averaging in automatic control community, plays a central role in the convergence analysis. We establish that with probability (w.p.) 1, every limit point of the sequence generated by Prox-SGD is a stationary point of the nonsmooth nonconvex problem (1). This is in sharp contrast to unconstrained optimization, where the convergence of the vanilla SGD method has long been well understood while the convergence of the SGD method with momentum was only settled recently. Nevertheless, the convergence rate of the proposed algorithm is not derived in the current work and is worth further investigating.

To test the proposed algorithm, we consider two applications. The first application is to train a SNN, and we leverage ℓ_1-regularization, that is,

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \sum_{i=1}^{m} f_i(\boldsymbol{x}, \boldsymbol{y}_i) + \mu||\boldsymbol{x}||_1. \tag{2}$$

The second application is to train a binary neural network (BNN) where the weights (and activations) are either 1 or -1 (see Courbariaux et al. (2015; 2016); Hou et al. (2017); Yin et al. (2018); Bai et al. (2019) for more details). To achieve this, we augment the loss function with a term that penalizes weights not being +1 or -1:

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{a}}{\text{minimize}} \quad & \sum_{i=1}^{m} f_i(\boldsymbol{x}, \boldsymbol{y}_i) + \frac{\mu}{4}\sum_{j=1}^{n}(a_j(x_j+1)^2 + (1-a_j)(x_j-1)^2) \\ \text{subject to} \quad & a_l = 0 \text{ or } 1, j = 1, \ldots, n, \end{aligned}$$

where μ is a given penalty parameter. The binary variable a_l can be interpreted as a switch: when $a_j = 0$, $a_j(x_j + 1)^2$ is deactivated, and there is a strong incentive for x_j to be 1 (the analysis for $a_j = 1$ is similar). We relax a_j to be a continuous variable between 0 and 1. To summarize, a BNN can be obtained by solving the following regularized optimization problem under constraints with respect to $\boldsymbol{x}$ and $\boldsymbol{a}$

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{a}}{\text{minimize}} \quad & \sum_{i=1}^{m} f_i(\boldsymbol{x}, \boldsymbol{y}_i) + \frac{\mu}{4}\sum_{j=1}^{n}(a_j(x_j+1)^2 + (1-a_j)(x_j-1)^2) \\ \text{subject to} \quad & 0 \leq a_j \leq 1, j = 1, \ldots, n. \end{aligned} \tag{3}$$

If μ is properly selected (or sufficiently large), the optimal a_j will be exactly or close to 0 or 1. Consequently, regularization and constraints offer interpretability and flexibility, which allows us to use more accurate models to promote structures in the neural networks, and the proposed convergent Prox-SGD algorithm ensures efficient training of such models.

2 THE PROPOSED PROX-SGD ALGORITHM

In this section, we describe the Prox-SGD algorithm to solve (1).

Background and setup. We make the following blanket assumptions on problem (1).

- $f_i(\boldsymbol{x}, \boldsymbol{y}^{(i)})$ is smooth (continuously differentiable) but not necessarily convex.
- $\nabla_{\boldsymbol{x}} f_i(\boldsymbol{x}, \boldsymbol{y}^{(i)})$ is Lipschitz continuous with constant L_i.
- $r(\boldsymbol{x})$ is convex and lower semicontinuous (not necessarily smooth).
- $\mathbb{X}$ is convex and compact (closed and bounded).

We are interested in algorithms that can find a stationary point of (1). A stationary point $\boldsymbol{x}^\star$ satisfies the optimality condition: at $\boldsymbol{x} = \boldsymbol{x}^\star$, $r(\boldsymbol{x})$ has a subgradient $\boldsymbol{\xi}(\boldsymbol{x}^\star)$ such that

$$(\boldsymbol{x} - \boldsymbol{x}^\star)^T \nabla f(\boldsymbol{x}^\star) + (\boldsymbol{x} - \boldsymbol{x}^\star)^T \boldsymbol{\xi}(\boldsymbol{x}^\star) \geq 0, \forall \boldsymbol{x} \in \mathbb{X}. \tag{4}$$

When $r(\boldsymbol{x}) = 0$ and $\mathbb{X} = \mathbb{R}^n$, the deterministic optimization problem 1 can be solved by the (batch) gradient descent method. When m, the number of training examples, is large, it is computationally expensive to calculate the gradient. Instead, we estimate the gradient by a minibatch of $m(t)$ training examples. We denote the minibatch by $\mathbb{M}(t)$: its elements are drawn uniformly from $\{1, 2, \ldots, m\}$ and there are $m(t)$ elements. Then the estimated gradient is

$$\boldsymbol{g}(t) \triangleq \frac{1}{m(t)} \sum_{i \in \mathbb{M}(t)} \nabla f_i(\boldsymbol{x}(t), \boldsymbol{y}^{(i)}) \tag{5}$$

and it is an unbiased estimate of the true gradient.

The proposed algorithm. The instantaneous gradient $\boldsymbol{g}(t)$ is used to form an aggregate gradient (momentum) $\boldsymbol{v}(t)$, which is updated recursively as follows

$$\boldsymbol{v}(t) = (1 - \rho(t))\boldsymbol{v}(t-1) + \rho(t)\boldsymbol{g}(t), \tag{6}$$

where $\rho(t)$ is stepsize (the learning rate) for the momentum and $\rho(t) \in (0, 1]$.

At iteration t, we solve an approximation subproblem and denote its solution as $\widehat{\boldsymbol{x}}(\boldsymbol{x}(t), \boldsymbol{v}(t), \boldsymbol{\tau}(t))$, or simply $\widehat{\boldsymbol{x}}(t)$

$$\widehat{\boldsymbol{x}}(t) \triangleq \underset{\boldsymbol{x} \in \mathbb{X}}{\operatorname{argmin}} \left\{ (\boldsymbol{x} - \boldsymbol{x}(t))^T \boldsymbol{v}(t) + \frac{1}{2} (\boldsymbol{x} - \boldsymbol{x}^t)^T \operatorname{diag}(\boldsymbol{\tau}(t)) (\boldsymbol{x} - \boldsymbol{x}(t)) + r(\boldsymbol{x}) \right\}. \tag{7}$$

A quadratic regularization term is incorporated so that the subproblem (7) is strongly convex and its modulus is the minimum element of the vector $\boldsymbol{\tau}(t)$, denoted as $\underline{\tau}(t)$ and $\underline{\tau}(t) = \min_{j=1,\ldots,n} \tau_j(t)$. Note that $\underline{\tau}(t)$ should be lower bounded by a positive constant that is strictly larger than 0, so that the quadratic regularization in (7) will not vanish.

The difference between two vectors $\widehat{\boldsymbol{x}}(t)$ and $\boldsymbol{x}(t)$ specifies a direction starting at $\boldsymbol{x}(t)$ and ending at $\widehat{\boldsymbol{x}}(t)$. This update direction is used to refine the weight vector

$$\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + \epsilon(t)(\widehat{\boldsymbol{x}}(t) - \boldsymbol{x}(t)), \tag{8}$$

where $\epsilon(t)$ is a stepsize (learning rate) for the weight and $\epsilon(t) \in (0, 1]$. Note that $\boldsymbol{x}(t+1)$ is feasible as long as $\boldsymbol{x}(t)$ is feasible, as it is the convex combination of two feasible points $\boldsymbol{x}(t)$ and $\widehat{\boldsymbol{x}}(t)$ while the set $\mathbb{X}$ is convex.

The above steps (5)-(8) are summarized in Algorithm 1, which is termed proximal-type Stochastic Gradient Descent (Prox-SGD), for the reason that the explicit constraint $x \in \mathbb{X}$ in (7) can also be formulated implicitly as a regularization function, more specifically, the indicator function $\sigma_{\mathbb{X}}(\boldsymbol{x})$. If all elements of $\boldsymbol{\tau}(t)$ are equal, then $\widehat{\boldsymbol{x}}(t)$ is exactly the proximal operator

$$\begin{aligned}
\widehat{\boldsymbol{x}}(t) &= \underset{\boldsymbol{x}}{\operatorname{argmin}} \left\{ \left\| \boldsymbol{x} - \left(\boldsymbol{x}(t) - \frac{1}{\tau(t)} \boldsymbol{v}(t) \right) \right\|_2^2 + \frac{1}{\tau(t)} r(\boldsymbol{x}) + \delta_{\mathbb{X}}(\boldsymbol{x}) \right\} \\
&\triangleq \operatorname{Prox}_{\frac{1}{\tau(t)} r(\boldsymbol{x}) + \delta_{\mathbb{X}}(\boldsymbol{x})} \left(\boldsymbol{x}(t) - \frac{1}{\tau(t)} \boldsymbol{v}(t) \right).
\end{aligned}$$

Prox-SGD in Algorithm 1 bears a similar structure as several SGD algorithms, without and with momentum, see Table 1, and it allows to interpret some existing algorithms as special cases of the

Algorithm 1 Proximal-type Stochastic Gradient Descent (Prox-SGD) Method

Input: $\boldsymbol{x}(0) \in \mathbb{X}$, $\boldsymbol{v}(-1) = \mathbf{0}$, $t = 0, T$, $\{\rho(t)\}_{t=0}^{T}$, $\{\epsilon(t)\}_{t=0}^{T}$.
for $t = 0 : 1 : T$ **do**

1. Compute the instantaneous gradient $\boldsymbol{g}(t)$ based on the minibatch $\mathbb{M}(t)$:

$$\boldsymbol{g}(t) = \frac{1}{m(t)} \sum_{i \in \mathbb{M}(t)} \nabla_{\boldsymbol{x}} f_i(\boldsymbol{x}(t), \boldsymbol{y}^{(i)}).$$

2. Update the momentum: $\mathbf{v}(t) = (1 - \rho(t))\mathbf{v}(t-1) + \rho(t)\boldsymbol{g}(t)$.

3. Compute $\widehat{\boldsymbol{x}}(t)$ by solving the approximation subproblem:

$$\widehat{\boldsymbol{x}}(t) = \arg\min_{\boldsymbol{x} \in \mathbb{X}} \left\{ (\boldsymbol{x} - \boldsymbol{x}(t))^T \mathbf{v}(t) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}(t))^T \text{diag}(\boldsymbol{\tau}(t))(\boldsymbol{x} - \boldsymbol{x}(t)) + \mu \cdot r(\boldsymbol{x}) \right\}.$$

4. Update the weight: $\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + \epsilon(t)(\widehat{\boldsymbol{x}}(t) - \boldsymbol{x}(t))$.

end for

Algorithm	momentum	weight	quadratic gain in subproblem	regularization	constraint set
Prox-SGD	$\rho(t)$	$\epsilon(t)$	$\boldsymbol{\tau}(t)$	convex	convex compact
SGD	1	$\epsilon(t)$	$\mathbf{1}$	0	$\mathbb{R}^n$
AdaGrad	1	ϵ	$\sqrt{\mathbf{r}(t)} + \delta\mathbf{1}^{\dagger}$	0	$\mathbb{R}^n$
RMSProp	1	ϵ	$\sqrt{\mathbf{r}(t) + \delta\mathbf{1}}^{\ddagger}$	0	$\mathbb{R}^n$
ADAM	ρ	$\frac{\epsilon}{1-\rho^t}$	$\sqrt{\frac{\mathbf{r}(t)}{1-\rho_2^t}} + \delta\mathbf{1}^{\ddagger}$	0	$\mathbb{R}^n$
AMSGrad	ρ	ϵ	$\sqrt{\widehat{\boldsymbol{r}}(t)}$ $\widehat{\boldsymbol{r}}(t) = \max(\widehat{\boldsymbol{r}}(t-1), \boldsymbol{r}(t))^{\ddagger}$	0	$\mathbb{R}^n$
ADABound	ρ	1	$\frac{\mathbf{1}}{\text{clip}(\epsilon(t)/\sqrt{\boldsymbol{r}(t)}, \boldsymbol{\eta}_l(t), \boldsymbol{\eta}_u(t))}^{\ddagger}$	0	$\mathbb{R}^n$

Table 1: Connection between the proposed framework and existing methods, where ρ, β, ϵ and δ are some predefined constants. $^{\dagger}\mathbf{r}(t) = \mathbf{r}(t-1) + \mathbf{g}(t) \odot \mathbf{g}(t)$, $^{\ddagger}\mathbf{r}(t) = \beta\mathbf{r}(t-1) + (1-\beta)\mathbf{g}(t) \odot \mathbf{g}(t)$.

proposed framework. For example, no momentum is used in SGD, and this amounts to setting $\rho(t) = 1$ in Algorithm 1. In ADAM, the learning rate for momentum is a constant ρ and the learning rate for the weight vector is given by $\epsilon/(1-\rho^t)$ for some ϵ, and this simply amounts to setting $\rho(t) = \rho$ and $\epsilon(t) = \epsilon/(1-\rho^t)$ in Algorithm 1. This interpretation also implies that the convergence conditions to be proposed shortly later are also suffcient for existing algorithms.

Solving the approximation subproblem (7). Since (7) is strongly convex, $\widehat{\boldsymbol{x}}(t)$ is unique. Generally $\widehat{\boldsymbol{x}}(t)$ in (7) does not admit a closed-form expression and should be solved by a generic solver. However, some important special cases that are frequently used in practice can be solved efficiently.

• The trivial case is $\mathbb{X} = \mathbb{R}^n$ and $r = 0$, where

$$\widehat{\boldsymbol{x}}(t) = \boldsymbol{x}(t) - \frac{\boldsymbol{v}(t)}{\boldsymbol{\tau}(t)}, \tag{9}$$

where the vector division is understood to be element-wise. When $\mathbb{X} = \mathbb{R}^n$ and $r(\boldsymbol{x}) = \mu\|\boldsymbol{x}\|_1$, $\widehat{\boldsymbol{x}}(t)$ has a closed-form expression that is known as the soft-thresholding operator

$$\widehat{\boldsymbol{x}}(t) = S_{\frac{\mu\mathbf{1}}{\boldsymbol{\tau}(t)}}\left(\boldsymbol{x}(t) - \frac{\boldsymbol{v}(t)}{\boldsymbol{\tau}(t)}\right), \tag{10}$$

where $S_{\boldsymbol{a}}(\boldsymbol{b}) \triangleq \max(\boldsymbol{b} - \boldsymbol{a}, \mathbf{0}) - \max(-\boldsymbol{b} - \boldsymbol{a}, \mathbf{0})$ (Bach et al., 2011).

• If $\mathbb{X} = \mathbb{R}^n$ and $r(\boldsymbol{x}) = \mu\|\boldsymbol{x}\|_2$ and $\boldsymbol{\tau}(t) = \tau\boldsymbol{I}$ for some τ, then (Parikh & Boyd, 2014)

$$\widehat{\boldsymbol{x}}(t) = \begin{cases} (1 - \mu/\|\tau\boldsymbol{x}(t) - \boldsymbol{v}(t)\|_2)\,(\boldsymbol{x}(t) - \boldsymbol{v}(t)/\tau), & \text{if } \|\tau\boldsymbol{x}(t) - \boldsymbol{v}(t)\|_2 \geq \mu \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

If $\boldsymbol{x}$ is divided into blocks $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$, the ℓ_2-regularization is commonly used to promote block sparsity (rather than element sparsity by ℓ_1-regularization).

• When there is a bound constraint $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$, then $\widehat{\boldsymbol{x}}(t)$ can simply be obtained by first solving the approximation subproblem (7) without the bound constraint and then projecting the optimal point onto the interval $[\boldsymbol{l}, \boldsymbol{u}]$. For example, when $\mathbb{X} = \mathbb{R}^n$ and $r = 0$,

$$\widehat{\boldsymbol{x}}(t) = \left[\boldsymbol{x}(t) - \frac{\boldsymbol{v}(t)}{\boldsymbol{\tau}(t)}\right]_{\boldsymbol{l}}^{\boldsymbol{u}}, \tag{12}$$

with $[\boldsymbol{x}]_{\boldsymbol{l}}^{\boldsymbol{u}} = \text{clip}(\boldsymbol{x}, \boldsymbol{l}, \boldsymbol{u}) \triangleq \min(\max(\boldsymbol{x}, \boldsymbol{l}), \boldsymbol{u})$.

• If the constraint function is quadratic: $\mathbb{X} = \{\boldsymbol{x} : \|\boldsymbol{x}\|_2^2 \leq 1\}$, $\widehat{\boldsymbol{x}}(t)$ has a semi-analytical expression (up to a scalar Lagrange multiplier which can be found efficiently by the bisection method).

Approximation subproblem. We explain why we update the weights by solving an approximation subproblem (7). First, we denote $\widetilde{f}$ as the smooth part of the objective function in (7). Clearly it depends on $\boldsymbol{x}(t)$ and $\boldsymbol{v}(t)$ (and thus $\mathbb{M}(t)$), while $\boldsymbol{x}(t)$ and $\boldsymbol{v}(t)$ depend on the old weights $\boldsymbol{x}(0), \ldots, \boldsymbol{x}(t-1)$ and momentum and $\boldsymbol{v}(0), \ldots, \boldsymbol{v}(t-1)$. Define $\mathbb{F}(t) \triangleq \{\boldsymbol{x}(0), \ldots, \boldsymbol{x}(t), \mathbb{M}(0), \ldots, \mathbb{M}(t)\}$ as a shorthand notation to denote the past history of the proposed algorithm. We formally write $\widetilde{f}$ as

$$\widetilde{f}(\boldsymbol{x}; \mathbb{F}(t)) \triangleq (\boldsymbol{x} - \boldsymbol{x}(t))^T \boldsymbol{v}(t) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}(t))^T \text{diag}(\boldsymbol{\tau}(t))(\boldsymbol{x} - \boldsymbol{x}(t)). \tag{13}$$

It follows from the optimality of $\widehat{\boldsymbol{x}}(t)$ that

$$\widetilde{f}(\boldsymbol{x}(t); \mathbb{F}(t)) + r(\boldsymbol{x}(t)) \geq \widetilde{f}(\widehat{\boldsymbol{x}}(t); \mathbb{F}(t)) + r(\widehat{\boldsymbol{x}}(t)).$$

After inserting (13) and reorganizing the terms, the above inequality becomes

$$(\widehat{\boldsymbol{x}}(t) - \boldsymbol{x}(t))^T \boldsymbol{v}(t) + r(\widehat{\boldsymbol{x}}(t)) - r(\boldsymbol{x}(t)) \leq -\tau(t)\|\widehat{\boldsymbol{x}}(t) - \boldsymbol{x}(t)\|_2^2. \tag{14}$$

Since $\nabla f(\boldsymbol{x})$ is Lipschitz continuous with constant L, it readily follows that

$$\begin{aligned}
&f(\boldsymbol{x}(t+1)) + r(\boldsymbol{x}(t+1)) - (f(\boldsymbol{x}(t)) + r(\boldsymbol{x}(t))) \\
&\leq (\boldsymbol{x}(t+1) - \boldsymbol{x}(t))^T \nabla f(\boldsymbol{x}(t)) + \frac{L}{2}\|\boldsymbol{x}(t+1) - vx(t)\|_2^2 + r(\boldsymbol{x}(t+1)) - r(\boldsymbol{x}(t)) \qquad (15) \\
&\leq \epsilon(t)\left((\widehat{\boldsymbol{x}}(t) - \boldsymbol{x}(t))^T \nabla f(\boldsymbol{x}(t)) + r(\widehat{\boldsymbol{x}}(t) - r(\boldsymbol{x}(t))) + \frac{L}{2}\epsilon(t)\|\widehat{\boldsymbol{x}}(t) - \boldsymbol{x}(t)\|_2^2\right), \qquad (16)
\end{aligned}$$

where the first inequality follows from the descent lemma and the second inequality follows from the Jensen's inequality of the convex function r and the update rule (8).

If $\boldsymbol{v}(t) = \nabla f(\boldsymbol{x}(t))$ (which is true asymptotically as we show shortly later), by replacing $\nabla f(\boldsymbol{x}(t))$ in (16) by $\boldsymbol{v}(t)$ and inserting (14) into (16), we obtain

$$f(\boldsymbol{x}(t+1)) + r(\boldsymbol{x}(t+1)) - (f(\boldsymbol{x}(t)) + r(\boldsymbol{x}(t))) \leq \epsilon(t)\left(\frac{L}{2}\epsilon(t) - \tau(t)\right)\|\widehat{\boldsymbol{x}}(t) - \boldsymbol{x}(t)\|_2^2. \tag{17}$$

The right hand side (RHS) will be negative when $\epsilon(t) < \frac{2\tau(t)}{L}$: this will eventually be satisfied as we shall use a decaying $\epsilon(t)$. This implies that the proposed algorithm will decrease the objective value of (1) after each iteration.

Gradient averaging and algorithm convergence. It turns out that the gradient averaging (momentum) step in (6) is essential for the convergence of Prox-SGD. Under some technical assumptions, the aggregate gradient $\boldsymbol{v}(t)$ will converge to the true gradient. This remark is made rigorous in the following theorem.

Theorem 1. *Assume that the unbiased gradient $\boldsymbol{g}(t)$ has a bounded second moment*

$$\mathbb{E}\left[\|\boldsymbol{g}(t)\|_2^2 \,|\mathbb{F}(t)\right] \leq C, \tag{18}$$

for some finite and positive constant C, and the sequence of stepsizes $\{\rho(t)\}$ and $\{\epsilon(t)\}$ satisfy

$$\sum_{t=0}^{\infty} \rho(t) = \infty, \sum_{t=0}^{\infty} \rho(t)^2 < \infty, \sum_{t=0}^{\infty} \epsilon(t) = \infty, \sum_{t=0}^{\infty} \epsilon(t)^2 < \infty, \epsilon(t)/\rho(t) \to 0. \tag{19}$$

Then every limit point of the sequence $\{\boldsymbol{x}(t)\}$ is a stationary point of (1) w.p.1.

Proof. Under the assumptions (18) and (19), it follows from Lemma 1 of Ruszczyński (1980) that $\boldsymbol{v}(t) \to \nabla f(\boldsymbol{x}(t))$. Since the descent direction $\widehat{\boldsymbol{x}}(t) - \boldsymbol{x}(t)$ is a descent direction in view of (14) (and in the spirit of (A8) in Ruszczyński (1980)), the claim in the theorem can be obtained by following the same line of analysis in Theorem 2 of Ruszczyński (1980). We omit the details to avoid duplicating results from known works. □

The bounded second moment assumption on the gradient $\boldsymbol{g}$ in (18) and decreasing stepsizes in (19) are standard assumptions in stochastic optimization and SGD. What is noteworthy is that $\epsilon(t)$ should decrease faster than $\rho(t)$ to ensure that $\boldsymbol{v}(t) \to \nabla f(\boldsymbol{x}(t))$. But this is more of an interest from the theoretical perspective, and in practice, we observe that $\epsilon(t)/\rho(t) = a$ for some constant a that is smaller than 1 usually yields satisfactory performance, as we show numerically in the next section. Furthermore, the quadratic gain $\underline{\tau}(t)$ in the approximation subproblem (7) should be lower bounded by some positive constant. In practice, there are different rationales to define it, see Table 1.

3 Simulation Results

In this section, we first train a SNN to compare the proposed Prox-SGD algorithm with ADAM (Kingma & Ba, 2015) and two other adaptive optimization algorithms, namely AMSGrad (Reddi et al., 2018) and ADABound (Luo et al., 2019), on the MNIST and CIFAR-10 datasets. Then we train a BNN to illustrate the merit of regularization and constraints.

3.1 Experiment: Deep Neural Network on MNIST

To compare the convergence of the proposed Prox-SGD method with state-of-the-art algorithms, we train a 6-layer fully-connected deep neural network (DNN) for the MNIST multiclass classification problem. The chosen setup is particularly suited to evaluate the merit of our method, since deep fully-connected networks are prone to overfitting without regularization, even on a simple dataset such as MNIST. The exact settings of our DNN are shown in Table 2.

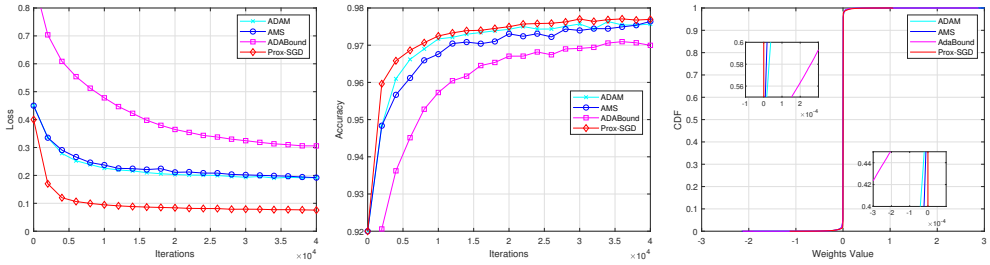
Table 2: DNN Settings

Parameter	**Value**
Dataset	MNIST
Size of train set	60000
Size of test set	10000
Number of input nodes	28×28
Number of hidden layers	6
Number of nodes per hidden layer	200
Number of output nodes	10
Activation function in hidden layers	ReLU
Activation function in output layer	Softmax
Regularization function	ℓ_1-Norm
Loss function	Cross entropy

Following the parameter configurations of ADAM in Kingma & Ba (2015), AMSGrad in Reddi et al. (2018), and ADABound in Luo et al. (2019), we set $\rho = 0.9$, $\beta = 0.999$ and $\epsilon = 0.001$, which are uniform for all the algorithms and commonly used for ADAM in practice. Note that we have also incorporated ℓ_1-regularization in these algorithms by the built-in function in Tensorflow/Pytorch, which amounts to adding the subgradient of the ℓ_1-norm to the gradient of the loss function. For the proposed Prox-SGD, $\rho(t)$ and $\epsilon(t)$ decrease over the iterations as follows,

$$\rho(t) = \frac{0.9}{(t+4)^{0.5}}, \;\; \epsilon(t) = \frac{0.06}{(t+4)^{0.5}}. \tag{20}$$

Recall that the ℓ_1-norm in the approximation subproblem naturally leads to the soft-thresholding proximal mapping, see (10). The regularization parameter μ in the soft-thresholding then permits controlling the sparsity of the parameter variable $\boldsymbol{x}$.

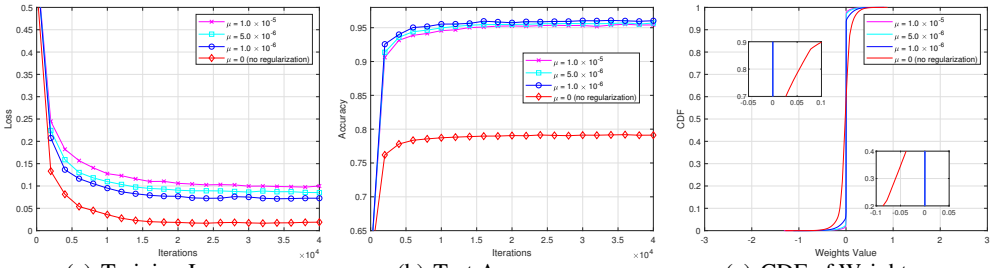


(a) Loss (b) Accuracy (c) Sparsity

Figure 1: Performance comparison for DNN on MNIST with ReLU.

In Fig. 1, adopting ReLU in each hidden layer, we compare the four algorithms in terms of three metrics. From Fig. 1 (a) and (b), Prox-SGD achieves the lowest loss value and best predicted accuracy with fast convergence compared to the other three algorithms. The sparsity of the trained model is measured by the cumulative distribution function (CDF) of the weights, which specifies the percentage of weights before any given value. From Fig. 1 (c), we can observe around 0 in the x-axis the abrupt change of the CDF, which implies that more than 90% of the weights are exactly zero. We also observe that Prox-SGD with the soft-thresholding proximal mapping achieves a similar sparsity as the other algorithms based on stochastic subgradient, so the proposed Prox-SGD does not have a clear advantage in this numerical example.

In Fig. 2, we test that the effect of the regularization parameter μ and the effect of ℓ_1-regularization in overcoming overfitting. To this end, we create an artificial DNN that overfits the training data, and we test Prox-SGD with and without the ℓ_1-regularization. The smallest training loss is achieved when there is no ℓ_1-regularization, but its test accuracy is also low: this is a clear sign of overfitting. Although ℓ_1-regularization increases the training loss, the test accuracy is significantly improved. From Fig. 2(c) we see that when there is no regularization, all weights are nonzero and many of them are very small, while more than 90% of all weights are exactly zero by the Prox-SGD algorithm. Note that the regularization parameter μ constitutes another hyperparameter and requires tuning. In practice we find that it is sufficient to find a range as all values in the range yield comparable performance.



(a) Training Loss (b) Test Accuracy (c) CDF of Weights

Figure 2: Performance comparison with different regularization parameter μ for DNN on MNIST with ReLU.

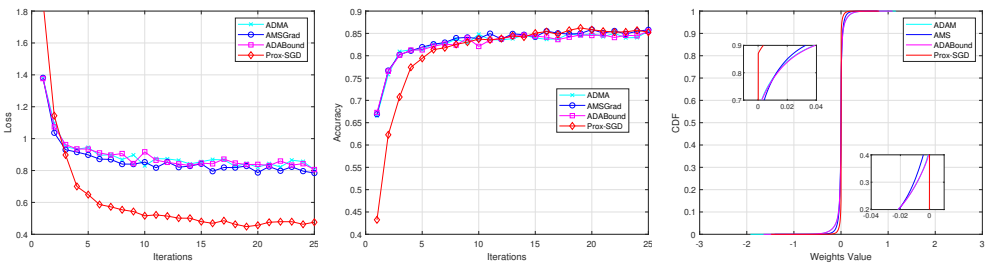
3.2 Experiment: Convolution Neural Network on CIFAR-10

To evaluate the performance of Prox-SGD for a larger and more complex dataset and network, we consider the multiclass classification problem on the standard CIFAR-10 dataset with convolution neural network (CNN). The network has 6 convolutional layers and each of them is followed by a max-pooling layer. While the CNN parameter settings are shown in Table 3 while the algorithm

Table 3: CNN Settings

Parameter	**Value**
Data set	CIFAR-10
Size of train set	50000
Size of test set	10000
Number of input nodes	32×32×3 (RGB)
Number of convolution layers	6
Size of convolution kernels	3×3
Number of convolution channels	32 (convolution 1-2) 64 (convolution 3-4) 128 (convolution 5-6)
Type of pooling layer	Max pooling
Size of pooling	2×2
Activation function	Elu (for convolution) Softmax (for output)
Regularization function	ℓ_1-norm
Loss function	Cross entropy

parameter settings are identical to the MNIST, we emphasize that ℓ_1-regularization is used in the benchmark algorithms and its subgradient is added to the gradient of the loss function.



(a) Training Loss (b) Test Accuracy (c) CDF of Weights

Figure 3: Performance comparison for CNN on CIFAR-10.

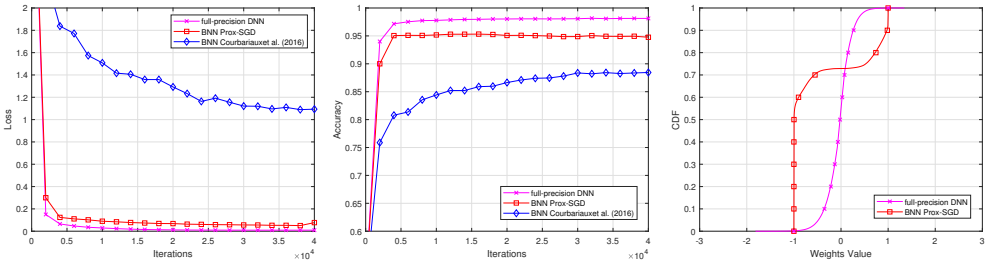
Fig. 3 shows the performance of Prox-SGD and the other three algorithms on CIFAR-10 dataset. On the one hand, Fig. 3(a) shows that Prox-SGD outperforms ADAM, AMSGrad and ADABound in the achieved loss value. On the other hand, the accuracy achieved by different algorithms is comparable, see Fig. 3(b). From the CDF of the weights in Fig. 3, we see again that the proposed prox-SGD achieves a much sparser network: around 90% of the weights are exactly zero. By comparison, only 40%-50% are exactly zero by other algorithms. Therefore, in this numerical example, the proposed Prox-SGD with soft-thresholding proximal mapping has a clear and substantial advantage than other stochastic subgradient-based algorithms.

3.3 Experiment: Training Binary Neural Networks

In this subsection, we evaluate the proposed algorithm Prox-SGD in training the BNN by solving problem (3). The simulation setup is as same as in Sec. 3.1 with $\mu = 2 \cdot 10^{-4}$, except that we use a tanh activation function to promote a binary activation output.

After customizing the general description in Algorithm 1 to problem (3), the approximation sub-problem is

$$(\widehat{\boldsymbol{x}}(t), \widehat{\boldsymbol{a}}(t)) = \underset{\mathbf{0} \le \boldsymbol{a} \le \mathbf{1}}{\arg\min} \left\{ \begin{array}{l} (\boldsymbol{x} - \boldsymbol{x}(t))^T \boldsymbol{v}_x(t) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}(t))^T \mathrm{diag}(\boldsymbol{\tau}_x(t))(\boldsymbol{x} - \boldsymbol{x}(t)) \\ +(\boldsymbol{a} - \boldsymbol{a}(t))^T \boldsymbol{v}_a(t) + \frac{1}{2}(\boldsymbol{a} - \boldsymbol{a}(t))^T \mathrm{diag}(\boldsymbol{\tau}_a(t))(\boldsymbol{a} - \boldsymbol{a}(t)) \end{array} \right\}.$$



(a) Training Loss (b) Test Accuracy (c) CDF of Weights

Figure 4: Performance comparison for BNN on MNIST

Both $\widehat{\boldsymbol{x}}(t)$ and $\widehat{\boldsymbol{a}}(t)$ have a closed-form expression (cf. (9) and (12))

$$\widehat{\boldsymbol{x}}(t) = \boldsymbol{x}(t) - \frac{\boldsymbol{v}_x(t)}{\boldsymbol{\tau}_x(t)}, \text{ and } \widehat{\boldsymbol{a}}(t) = \left[\boldsymbol{a}(t) - \frac{\boldsymbol{v}_a(t)}{\boldsymbol{\tau}_a(t)}\right]_{\mathbf{0}}^{\mathbf{1}}, \tag{21}$$

where $\boldsymbol{v}_x(t)$ and $\boldsymbol{v}_a(t)$ are the momentum updated in the spirit of (6), with the gradients given by

$$\boldsymbol{g}_x(t) = \frac{1}{m(t)} \sum_{i \in \mathbb{M}(t)} \nabla f_i(\boldsymbol{x}(t), \boldsymbol{y}^{(i)}) + \frac{\mu}{2}(\boldsymbol{x}(t) + 2\boldsymbol{a}(t) - \mathbf{1}), \text{ and } \boldsymbol{g}_a(t) = \mu \boldsymbol{x}(t).$$

The training loss is shown in Fig. 4 (a). We remark that during the training process of the proposed Prox-SGD, the weights are not binarized, for the reason that the penalty should regularize the problem in a way such that the optimal weights (to which Prox-SGD converges) are exactly or close to 1 or -1. After training is completed, the CDF of the learned weights is summarized in Fig. 4(c), and then the learned weights are binarized to generate a full BNN whose test accuracy is in Fig. 4 (b). On the one hand, we see from Fig. 4 (a)-(b) that the achieved training loss and test accuracy by BNN is worse than the standard full-precision DNN (possibly with soft-thresholding). This is expected as BNN imposes regularization and constraints on the optimization problem and reduces the search space. However, the difference in test accuracy is quite small. On the other hand, we see from Fig. 4(c) that the regularization in the proposed formulation (3) is very effective in promoting binary weights: 15% of weights are in the range (-1,-0.5) and 15% of weights are in the range (0.5,1), and all the other weights are either -1 or 1. As all weights are exactly or close to 1 or -1, we could just binarize the weights to exactly 1 or -1 only once, after the training is completed, and thus the incurred performance loss is small (98% versus 95% for test accuracy). In contrast, the weights generated by the full-precision DNN (that is, without regularization) are smoothly distributed in $[-2, 2]$.

Even though the proposed formulation (3) doubles the number of parameters to optimize (from $\boldsymbol{x}$ in full-precision DNN to $(\boldsymbol{x}, \boldsymbol{a})$ in BNN Prox-SGD), the convergence speed is equally fast. In comparison with the algorithm in Courbariaux et al. (2016), the proposed Prox-SGD converges much faster and achieves a much better training loss and test accuracy (95% versus 89%). The notable performance improvement is due to the regularization and constraints. Naturally we should make an effort of searching for a proper regularization parameter μ, but this effort is very well paid off. Furthermore, we observe in the simulations that the performance is not sensitive to the exact value of μ, as long as it is in an appropriate range.

4 Concluding Remarks

In this paper, we proposed Prox-SGD, a proximal-type stochastic gradient descent algorithm with momentum, for constrained optimization problems where the smooth loss function is augmented by a nonsmooth and convex regularization. We considered two applications, namely the stochastic training of SNN and BNN, to show that regularization and constraints can effectively promote structures in the learned network. More generally, incorporating regularization and constraints allows us to use a more accurate and interpretable model for the problem at hand and the proposed convergent Prox-SGD algorithms ensures efficient training. Numerical tests showed that Prox-SGD outperforms state-of-the-art algorithms, in terms of convergence speed, achieved training loss and/or the desired structure in the learned neural networks.

REFERENCES

Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. Optimization with Sparsity-Inducing Penalties. *Foundations and Trends in Machine Learning*, 4(1):1–106, 2011. doi: 10.1561/2200000015. URL http://arxiv.org/abs/1108.0775.

Yu Bai, Yu-Xiang Wang, and Edo Liberty. Proxquant: Quantized neural networks via proximal operators. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzMyhCcK7.

Dimitri P. Bertsekas and John N. Tsitsiklis. Gradient convergence in gradient methods with errors. *SIAM Journal on Optimization*, 10(3):627–642, 2000. ISSN 10526234. doi: 10.1137/S1052623497331063. URL http://link.aip.org/link/SJOPE8/v10/i3/p627/s1&Agg=doi.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1), 2010. ISSN 1935-8237. doi: 10.1561/2200000016. URL http://www.nowpublishers.com/product.aspx?product=MAL{&}doi=2200000016.

Xiangyi Chen, Sijia Liu, Ruoyu Sun, and Mingyi Hong. On the Convergence of A Class of Adam-Type Algorithms for Non-Convex Optimization. In *International Conference on Learning Representations*, 2019. URL http://arxiv.org/abs/1808.02941.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 28*, pp. 3123–3131. Curran Associates, Inc., 2015.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. 2016. URL http://arxiv.org/abs/1602.02830.

Trevor Gale, Erich Elsen, and Sara Hooker. The State of Sparsity in Deep Neural Networks. 2019. URL http://arxiv.org/abs/1902.09574.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

Lu Hou, Quanming Yao, and James T. Kwok. Loss-aware binarization of deep networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=S1oWlN9ll.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, pp. 1–15, 2015. ISBN 9781450300728. doi: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503. URL http://arxiv.org/abs/1412.6980.

Yunwen Lei, Ting Hu, and Ke Tang. Stochastic Gradient Descent for Nonconvex Learning without Bounded Gradient Assumptions. (1):1–6, 2019. URL http://arxiv.org/abs/1902.00908.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning Sparse Neural Networks through L_0 Regularization. In *International Conference on Learning Representations*, pp. 1–13, 2018. URL http://arxiv.org/abs/1712.01312.

Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive Gradient Method with Dynamic Bound of Learning Rate. pp. 1–19, 2019. URL http://arxiv.org/abs/1902.09843v1.

Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):127–239, 2014. ISSN 2167-3888. doi: 10.1561/2400000003. URL http://www.nowpublishers.com/articles/foundations-and-trends-in-optimization/OPT-003.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of ADAM and beyond. In *International Conference on Learning Representations*, pp. 1–23, 2018. URL http://arxiv.org/abs/1904.09237.

Andrzej Ruszczyński. Feasible direction methods for stochastic programming problems. *Mathematical Programming*, 19(1):220–229, dec 1980. ISSN 0025-5610. doi: 10.1007/BF01581643. URL http://www.springerlink.com/index/10.1007/BF01581643.

Penghang Yin, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin. BinaryRelax: A Relaxation Approach for Training Deep Neural Networks with Quantized Weights. *SIAM Journal on Imaging Sciences*, 11(4):2205–2223, January 2018. URL https://epubs.siam.org/doi/10.1137/18M1166134.