A PROXIMAL BLOCK COORDINATE DESCENT ALGORITHM FOR DEEP NEURAL NETWORK TRAINING

Tim Tsz-Kit Lau[*], **Jinshan Zeng**[†][*], **Baoyuan Wu**[‡], **Yuan Yao**[*]
[*]Department of Mathematics, The Hong Kong University of Science and Technology
[†]School of Computer and Information Engineering, Jiangxi Normal University
[‡]Tencent AI Lab

ABSTRACT

Training deep neural networks (DNNs) efficiently is a challenge due to the associated highly nonconvex optimization. The backpropagation (backprop) algorithm has long been the most widely used algorithm for gradient computation of parameters of DNNs and is used along with gradient descent-type algorithms for this optimization task. Recent work have shown the efficiency of block coordinate descent (BCD) type methods empirically for training DNNs. In view of this, we propose a novel algorithm based on the BCD method for training DNNs and provide its global convergence results built upon the powerful framework of the Kurdyka-Łojasiewicz (KL) property. Numerical experiments on standard datasets demonstrate its competitive efficiency against standard optimizers with backprop.

1 INTRODUCTION

Backprop (Rumelhart et al., 1986) is the most prevalent approach of computing gradients in DNN training. It is mostly used together with gradient descent-type algorithms, notably the classical stochastic gradient descent (SGD) (Robbins & Monro, 1951) and its variants such as Adam (Kingma & Ba, 2015). Regardless of the optimizers chosen for network training, backprop suffers from vanishing gradients (Bengio et al., 1994; Pascanu et al., 2013; Goodfellow et al., 2016). Various methods were proposed to alleviate this issue, e.g., rectified linear units (ReLUs) (Nair & Hinton, 2010) and Long Short-Term Memory (Hochreiter & Schmidhuber, 1997), but these methods are unable to completely tackle this inherent problem to backprop. One viable alternative to backprop with gradient-based optimizers to avoid vanishing gradients is to adopt gradient-free methods, including (but not limited to) the alternating direction method of multipliers (ADMM) (Taylor et al., 2016; Zhang et al., 2016) and the block coordinate descent (BCD) method (Carreira-Perpiñán & Wang, 2014; Zhang & Brand, 2017). The main idea of ADMM and BCD is to decompose the highly coupled and composite DNN training objective into several loosely coupled and almost separable simple subproblems. The efficiency of both ADMM and BCD has been illustrated empirically in Taylor et al. (2016), Zhang et al. (2016) and Zhang & Brand (2017). Meanwhile, BCD has been tremendously studied for nonconvex problems in machine learning (see e.g., Jain & Kar, 2017).

In this paper, we propose a novel algorithm based on BCD of Gauss-Seidel type. We define the loss function using the quadratic penalty method (Nocedal & Wright, 1999) by unrolling the nested structure of DNNs into separate "single-layer" training tasks. This algorithm involves simplifications of commonly used activation functions as projections onto nonempty closed convex sets (commonly used in convex analysis (Rockafellar & Wets, 1998)) so that the overall loss function is block multiconvex (Xu & Yin, 2013). This property allows us to obtain global convergence guarantees under the framework of KL property (Attouch et al., 2013; Bolte et al., 2014).

2 RELATED WORK

Carreira-Perpiñán & Wang (2014) and Zhang & Brand (2017) also suggest the use of BCD for training DNNs and observe empirically the per epoch efficiency where the training loss drops much faster than SGD. Multiple related work consider a similar scheme to ours. A very recent piece of work (Frerix et al., 2018) implements proximal steps for model parameter updates only but keep gradient steps for updating the activation parameters and the output layer parameters, whilst we also apply proximal steps for updating these parameters. In the problem formulation of Zhang & Brand (2017), bias vectors are not used in the layers. They concatenate all weight matrices in all hidden layers and all activation vectors (defined below) respectively into two separate blocks and update together with the weight matrix of the output layer so that these three blocks are updated alternately instead of an overall backward order as in backprop. Carreira-Perpiñán & Wang (2014) consider a specific DNN using squared loss and sigmoid activation function, and propose the so-called method of auxiliary coordinate (MAC). Our problem formulation is similar, but is further simplified described in the next section.

3 The proximal block coordinate descent algorithm

3.1 Preliminaries and notations

We consider a feedforward neural network with L hidden layers. Let d_ℓ be the number of nodes of the ℓ-th layer, N be the number of training samples and K be the number of classes. Note that $d_{L+1} = K$. We adopt the following notations: $\boldsymbol{x}_j \in \mathbb{R}^{d_0}$ the j-th training data, $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) \in \mathbb{R}^{d_0 \times N}$, $\boldsymbol{y}_j \in \mathbb{R}^{d_{L+1}}$ the one-hot vector of its corresponding label, y_{ji} the i-th entry of the column vector $\boldsymbol{y}_j$, $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N) \in \mathbb{R}^{d_{L+1} \times N}$, $\boldsymbol{W}_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ the weight matrix between the ℓ-th and $(\ell-1)$-th hidden layers, $\boldsymbol{b}_\ell \in \mathbb{R}^{d_\ell}$ the bias vector of the ℓ-th hidden layer, $\boldsymbol{W}_{L+1} \in \mathbb{R}^{d_{L+1} \times d_L}$ the weight matrix between the last hidden layer and the output layer, $\boldsymbol{b}_{L+1} \in \mathbb{R}^{L+1}$ the bias vector of the output layer. We denote a general activation function by h, $\mathcal{W} := \{\boldsymbol{W}_\ell\}_{\ell=1}^{L+1}$ and $\boldsymbol{b} := \{\boldsymbol{b}_\ell\}_{\ell=1}^{L+1}$.

3.2 Problem formulation

In the training of regularized DNNs, we are solving the following optimization problem:

$$\min_{\boldsymbol{a},\boldsymbol{z},\mathcal{W},\boldsymbol{b}} F(\boldsymbol{a}, \mathcal{W}, \boldsymbol{b}) \equiv \gamma_{L+1} \sum_{j=1}^{N} \mathcal{L}(\boldsymbol{W}_{L+1}\boldsymbol{a}_{L,j} + \boldsymbol{b}_{L+1}; \boldsymbol{y}_j) + \sum_{\ell=1}^{L+1} r_\ell(\boldsymbol{W}_\ell)$$

$$\text{subject to } \boldsymbol{z}_{\ell,j} = \boldsymbol{W}_\ell \boldsymbol{a}_{\ell-1,j} + \boldsymbol{b}_\ell,\ \boldsymbol{a}_{\ell,j} = h(\boldsymbol{z}_{\ell,j}) \text{ for all } \ell \in \{1, \ldots, L\}, j \in \{1, \ldots, N\}. \quad (1)$$

where $\mathcal{L}$ is a generic convex loss function and r_ℓ, $\ell = 1, \ldots, L+1$, are convex but possibly nonsmooth regularizers, $\boldsymbol{a} := \{\boldsymbol{a}_{\ell,j}\}_{\ell=1}^{L}$ the set of all activation vectors, $\boldsymbol{z} := \{\boldsymbol{z}_{\ell,j}\}_{\ell=1}^{L}$, $\boldsymbol{a}_{0,j} := \boldsymbol{x}_j$ for all $j \in \{1, \ldots, N\}$, and $\gamma_{L+1} > 0$. The problem (1) can be reformulated as follows:

$$\min_{\boldsymbol{a},\boldsymbol{z},\mathcal{W},\boldsymbol{b}} F(\boldsymbol{a}, \mathcal{W}, \boldsymbol{b}) + \sum_{j=1}^{N} \sum_{\ell=1}^{L} \frac{\rho_\ell}{2} \|h(\boldsymbol{z}_{\ell,j}) - \boldsymbol{a}_{\ell,j}\|^2 + \sum_{j=1}^{N} \sum_{\ell=1}^{L} \frac{\gamma_\ell}{2} \|\boldsymbol{W}_\ell \boldsymbol{a}_{\ell-1,j} + \boldsymbol{b}_\ell - \boldsymbol{z}_\ell\|^2, \quad (2)$$

where $\|\cdot\|$ is the standard Euclidean norm, $\rho_\ell > 0$ and $\gamma_\ell > 0$ for all $\ell \in \{1, \ldots, L\}$. The above scheme allows for any general activation functions such as hyperbolic tangent, sigmoid and ReLU. However, the formulation (2) is generally hard to solve explicitly (say, for tanh and sigmoid) but can be simplified if we consider the constraint $\boldsymbol{a}_{\ell,j} = h(\boldsymbol{z}_{\ell,j})$ as a projection onto a convex set. For instance, ReLU can be thought of as a projection onto the closed upper half-space. This is equivalent to imposing the constraint $\boldsymbol{a}_{\ell,j} \succeq \boldsymbol{0} \Leftrightarrow \boldsymbol{a}_{\ell,j} \in \mathbb{R}_+^{d_\ell}$ for all $\ell \in \{1, \ldots, L\}$, and for all $j \in \{1, \ldots, N\}$. For hyperbolic tangent and sigmoid functions, nonsmooth approximations are needed and are discussed in Appendix A.1. Inspired by the formulation in Zhang et al. (2016), we further introduce a set of auxiliary variables to the objective to get:

$$\min_{\boldsymbol{a},\mathcal{W},\boldsymbol{b},\boldsymbol{u}} \widetilde{F}(\boldsymbol{a}, \mathcal{W}, \boldsymbol{b}, \boldsymbol{u}) \equiv F(\boldsymbol{a}, \mathcal{W}, \boldsymbol{b}) + \sum_{j=1}^{N} \sum_{\ell=1}^{L} \left[\frac{\gamma_\ell}{2} \|\boldsymbol{W}_\ell \boldsymbol{a}_{\ell-1,j} + \boldsymbol{b}_\ell - \boldsymbol{a}_{\ell,j} + \boldsymbol{u}_{\ell,j}\|^2 + \iota_{\mathcal{S}_\ell}(\boldsymbol{a}_{\ell,j}) \right], \quad (3)$$

where $\boldsymbol{u} := \{\boldsymbol{u}_{\ell,j}\}_{\ell=1}^{L}$ is the set of auxiliary variables, $\mathcal{S}_\ell$ is a nonempty closed convex set and $\iota_{\mathcal{S}_\ell}$ is the indicator function of a nonempty closed convex set $\mathcal{S}_\ell$ so that $\iota_{\mathcal{S}_\ell}(\boldsymbol{u}) = 0$ for $\boldsymbol{u} \in \mathcal{S}_\ell$ and $+\infty$ otherwise. For the case of the ReLU activation function, $\mathcal{S}_\ell = \mathbb{R}_+^{d_\ell}$. This formulation (3) is more desirable since we eliminate the variable $\boldsymbol{z}$ to be optimized which probably speeds up the training. Also note that the objective function (3) is block multiconvex which allows for established convergence guarantees in existing literature using the proposed algorithm.

3.3 THE PROPOSED ALGORITHM

In our minimization algorithm, we perform a proximal step (as in the proximal point algorithm) for each parameter except the auxiliary variables (which are updated by direct minimization instead of dual gradient ascent in Zhang et al. (2016)) in a Gauss-Seidel fashion, and in a backward order based on the network structure. Adaptive momentum (Lau & Yao, 2017), though not included in the convergence analysis, is also used after each proximal point update due to its empirical usefulness for convergence. The overall algorithm is depicted in Algorithm 1 (see Appendix A.2).

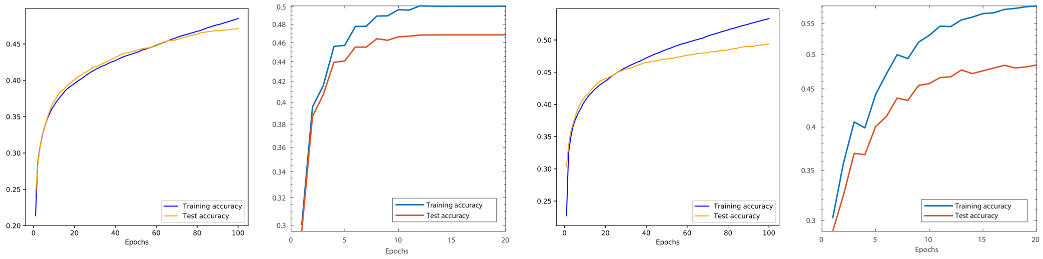
3.4 CONVERGENCE RESULTS

The problem formulation (3) involves regularized block multiconvex optimization and the proposed algorithm fits the general framework of Xu & Yin (2013). We analyze the convergence of Algorithm 1 under the assumptions in Assumption 1 (see Appendix A.4). The proof of the following theorem and its related convergence rate is given in Appendix A.4.

Theorem 1 (Global convergence) *Under Assumption 1, Proposition 1 and the fact that the sequence* $\{\boldsymbol{x}^{(k)}\}_{k\geq 1} := \{\boldsymbol{a}^{(k)}, \mathcal{W}^{(k)}, \boldsymbol{b}^{(k)}, \boldsymbol{u}^{(k)}\}_{k\geq 1}$ *generated by Algorithm 1 has a finite limit point* $\bar{\boldsymbol{x}} := \{\bar{\boldsymbol{a}}, \overline{\mathcal{W}}, \bar{\boldsymbol{b}}, \bar{\boldsymbol{u}}\}$ *where* $\widetilde{F}$ *satisfies the the Kurdyka-Łojasiewicz (KL) property (Definition 2, see Appendix A.4), the sequence* $\{\boldsymbol{x}^{(k)}\}_{k\geq 1}$ *converges to* $\bar{\boldsymbol{x}}$*, which is a critical point of* (3)*.*

4 EXPERIMENTAL RESULTS

We conduct experiments for two different structures on CIFAR-10 (Krizhevsky et al., 2009) with 50K training and 10K test samples, namely a 3072-4K-4K-4K-10 MLP and a 3072-4K-3072-4K-10 DNN with a residual connection in the second hidden layer (ResNet (He et al., 2016)). Experimental results on MNIST are in Appendix B. The BCD algorithm (20 epochs) is implemented using MATLAB while backprop (SGD; 100 epochs) is implemented using Keras with TensorFlow backend. Squared losses, ReLUs are used without regularizations. All weight matrices are initialized from a Gaussian distribution with a standard deviation of 0.01 and the bias vectors are initialized as vectors of all 0.1, while $\boldsymbol{a}$ and $\boldsymbol{u}$ are initialized by a single forward pass. Hyperparameters in BCD ($\beta_i = 0.95, \gamma_i = 0.1, t = 0.1, s = 1$ (MLP), $s = 0.1$ (ResNet)) and the learning rate (0.05) in SGD are tuned manually. We report the training and test accuracies (the median of 5 runs) as follows:



(a) MLP; SGD

(b) MLP; BCD: $\alpha_{\text{odd}} = 5, \alpha_{\text{even}} = 15$

(c) ResNet; SGD

(d) ResNet; BCD: $\alpha_{\text{odd}} = 10, \alpha_{\text{even}} = 50$

Figure 1: Training and test accuracies. Final test acc.: 1a: 0.4765; 1b: 0.4682; 1c: 0.494; 1d: 0.4843.

5 CONCLUSION AND FUTURE WORK

In this paper, we proposed an efficient BCD algorithm and established its convergence guarantees according to our block multiconvex formulation. Three major advantages of BCD include: (a) high per epoch efficiency at early stages (observed in Figure 1), i.e., the training and test accuracies of BCD grow much faster than SGD in terms of epoch at the early stage; (b) good scalability, i.e., BCD can be implemented in a distributed and parallel manner via data parallelism on multi-core CPUs; (c) gradient free, i.e., gradient computations are unnecessary used for the updates. One flaw of the BCD methods is that they generally require more memory than SGD method. Thus, a future direction is to study the feasibility of the stochastic and parallel block coordinate descent methods for DNN training.

ACKNOWLEDGMENTS

The authors would like to thank Ruohan Zhan for sharing experimental results on deep neural network training using BCD and ADMM algorithms.

REFERENCES

Hedy Attouch, Jérôme Bolte, and Benar Fux Svaiter. Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward-backward splitting, and regularized Gauss-Seidel methods. *Math. Prog.*, 137(1):91–129, Feb. 2013.

Heinz H. Bauschke and Patrick L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer International Publishing, 2nd edition, 2017.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Jacek Bochnak, Michel Coste, and Marie-Francoise Roy. *Real algebraic geometry*, volume 3. Ergeb. Math. Grenzgeb. Springer-Verlag, Berlin, 1998.

Jérôme Bolte, Aris Daniilidis, and Adrian Lewis. The Łojasiewicz inequality for nonsmooth subanalytic functions with applications to subgradient dynamical systems. *SIAM Journal on Optimization*, 17:1205–1223, 2007a.

Jérôme Bolte, Aris Daniilidis, Adrian Lewis, and Masahiro Shiota. Clark subgradients of stratifiable functions. *SIAM Journal on Optimization*, 18:556–572, 2007b.

Jérôme Bolte, Shoham Sabach, and Marc Teboulle. Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *Mathematical Programming*, 146(1):459–494, 2014.

Miguel Carreira-Perpiñán and Weiran Wang. Distributed optimization of deeply nested systems. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pp. 10–19, 2014.

Patrick L. Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. In Heinz H. Bauschke, Regina S. Burachik, Patrick L. Combettes, Veit Elser, D. Russell Luke, and Henry Wolkowicz (eds.), *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, pp. 185–212. Springer New York, 2011.

Thomas Frerix, Thomas Möllenhoff, Michael Moeller, and Daniel Cremers. Proximal backpropagation. *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

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

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9: 1735–1780, 1997.

Prateek Jain and Purushottam Kar. Non-convex optimization for machine learning. *Foundations and Trends® in Machine Learning*, 10(3–4):142–336, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-10 (Canadian Institute for Advanced Research). 2009. URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

Krzysztof Kurdyka. On gradients of functions definable in o-minimal structures. *Annales de l'institut Fourier*, 48:769–783, 1998.

Tsz Kit Lau and Yuan Yao. Accelerated block coordinate proximal gradients with applications in high dimensional statistics. *The 10th NIPS Workshop on Optimization for Machine Learning*, 2017.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`.

Stanisław Łojasiewicz. Une propriété topologique des sous-ensembles analytiques réels. In: Les Équations aux dérivées partielles. *Éditions du centre National de la Recherche Scientifique, Paris*, pp. 87–89, 1963.

Stanisław Łojasiewicz. Sur la geometrie semi-et sous-analytique. *Annales de l'institut Fourier*, 43: 1575–1595, 1993.

Jean-Jacques Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. *Comptes Rendus de l'Académie des Sciences (Paris), Série A*, 255:2897–2899, 1962.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. *In Proceedings of the 27th International Conference on Machine Learning (ICML)*, pp. 807–814, 2010.

Jorge Nocedal and Stephen J. Wright. Numerical optimization. *Springer*, 1999.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pp. 1310–1318, 2013.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

R. Tyrrell Rockafellar and Roger J.-B. Wets. *Variational Analysis*. Springer Verlag, 1998.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training neural networks without gradients: A scalable ADMM approach. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 2722–2731, 2016.

Yangyang Xu and Wotao Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on Imaging Sciences*, 6(3):1758–1789, 2013.

Ziming Zhang and Matthew Brand. Convergent block coordinate descent for training tikhonov regularized deep neural networks. In *Advances in Neural Information Processing Systems 30*, pp. 1719–1728. 2017.

Ziming Zhang, Yuting Chen, and Venkatesh Saligrama. Efficient training of very deep neural networks for supervised hashing. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1487–1495, 2016.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67:301–320, 2005.

APPENDICES

A ALGORITHMIC DETAILS

A.1 SIMPLIFICATIONS OF ACTIVATION FUNCTIONS

We first give the definition of the proximity operator which is required in the following analysis.

Definition 1 (Proximity operator (Moreau, 1962; Combettes & Pesquet, 2011))
Let λ be a positive parameter, $\mathcal{H}$ be a real Hilbert space (e.g., Euclidean space) and the function $g : \mathcal{H} \to (-\infty, +\infty]$. The *proximity operator* $\operatorname{prox}_{\lambda g} : \mathcal{H} \to \mathcal{H}$ of the function λg is defined through

$$\operatorname{prox}_{\lambda g}(x) := \operatorname*{argmin}_{y \in \mathcal{H}} g(y) + \frac{1}{2\lambda}\|y - x\|^2.$$

If g is convex, proper and lower semicontinuous, prox_g admits a unique solution. If g is nonconvex, then it is generally set-valued.

For the ReLU activation function, we can consider it as the projection onto the nonempty closed convex set $\mathbb{R}_+^{d_\ell}$, where d_ℓ is the dimension of the input variable. This is based on Bauschke & Combettes (2017, Proposition 24.47), which states that for any proper lower semicontinuous convex function ϕ on $\mathbb{R}$ and any closed interval $\mathit{\Omega}$ on $\mathbb{R}$ such that $\mathit{\Omega} \cap \operatorname{dom} \phi \neq \emptyset$,

$$\operatorname{prox}_{\phi + \iota_{\mathit{\Omega}}} = \mathcal{P}_{\mathit{\Omega}} \circ \operatorname{prox}_{\phi},$$

where $\mathcal{P}_{\mathit{\Omega}} \equiv \operatorname{prox}_{\iota_{\mathit{\Omega}}}$ is the projection operator onto the nonempty closed convex set $\mathit{\Omega}$. Since all the activation functions are elementwise operations, according to Bauschke & Combettes (2017, Proposition 24.11), we can extend the above results to the Euclidean space $\mathbb{R}^{d_\ell}$, i.e.,

$$(\forall \boldsymbol{x} \in \mathbb{R}^{d_\ell}) \quad \operatorname{prox}_{\boldsymbol{\phi} + \iota_{\boldsymbol{\Omega}}}(\boldsymbol{x}) = \Big(\operatorname{prox}_{\phi_i + \iota_{\mathit{\Omega}_i}}(x_i)\Big)_{1 \le i \le d_\ell} = \big(\mathcal{P}_{\mathit{\Omega}_i} \circ \operatorname{prox}_{\phi_i}(x_i)\big)_{1 \le i \le d_\ell},$$

where $\boldsymbol{x} = (x_i)_{1 \le i \le d_\ell}$, $\boldsymbol{\phi} = \bigoplus_{i=1}^{d_\ell} \phi_i$ and $\boldsymbol{\Omega} = \prod_{i=1}^{d_\ell} \mathit{\Omega}_i$.

Likewise, the sigmoid and tanh activation functions can be used with some tricks in this scheme. It should be noted that these two functions are not simple projections onto nonempty closed convex sets. Instead, if we consider the nonsmooth surrogates of them, they can be imposed as projections onto nonempty closed convex sets which are much easier to obtain.

For the tanh function, we use the following nonsmooth function (a.k.a. *hard tanh*) as an approximation:

$$f(x) = \begin{cases} -1 & \text{if } x < -1, \\ x & \text{if } x \in [-1, 1], \\ 1 & \text{if } x > 1. \end{cases}$$

Then we have

$$\mathcal{P}_{[-1,1]}(x) = \operatorname{prox}_{\iota_{[-1,1]}}(x) = \max\{-1, \min\{x, 1\}\}.$$

For the sigmoid function σ, recall that we have the following relationship:

$$\sigma(x) \equiv 0.5(1 + \tanh(0.5x)).$$

Using function transformation, the nonsmooth approximation of the sigmoid function (a.k.a. *hard sigmoid*[1]) is

$$g(x) = \begin{cases} 0 & \text{if } x < -2, \\ 0.25x + 0.5 & \text{if } x \in [-2, 2], \\ 1 & \text{if } x > 2. \end{cases}$$

We define the closed convex set $\Sigma := \{x \in \mathbb{R} : u \in [-2, 2], x = 0.25u + 0.5\}$. Then we have

$$\mathcal{P}_\Sigma(x) = \operatorname{prox}_{\iota_\Sigma}(x) = \max\{0, \min\{0.25x + 0.5, 1\}\} = 0.25 \max\{-2, \min\{x, 2\}\} + 0.5.$$

[1] Hard sigmoid can also be defined as: $g(x) = 0$ if $x < 2.5$, $g(x) = 0.2x + 0.5$ if $x \in [-2.5, 2.5]$, and $g(x) = 1$ if $x > 2.5$, as defined in TensorFlow.

A.2 THE PROXIMAL BCD ALGORITHM

Note that the extrapolations and adaptive momentums in the following algorithm are not considered in the convergence results but implemented in numerical experiments.

Algorithm 1: Proximal Block Coordinate Descent (BCD) Algorithm for Training DNNs

```
Input: training data $\{(\boldsymbol{x}_j, \boldsymbol{y}_j)\}_{j=1}^N$ and regularization parameters $\{\gamma_\ell\}_{\ell=1}^{L+1}$ and $\boldsymbol{a}_{L+1,j}^{(k)} := \boldsymbol{y}_j$ for all $k \in \mathbb{N}$ and $j \in \{1, \dots, N\}$.
Initialize $\mathcal{W}^{(0)}$, $\boldsymbol{b}^{(0)}$, $\alpha_i^{(0)} \in (0, \infty)$, $\beta_i^{(0)} \in (0, 1)$ for all $i$, and $s, t \in (0, 1]$; $\boldsymbol{a}^{(0)}$ initialized by forward propagation.
for $k = 1, 2, \dots$ do
    $\boldsymbol{W}_{L+1}^* = \operatorname{argmin}_{\boldsymbol{W}_{L+1}} \gamma_{L+1} \mathcal{L}(\boldsymbol{W}_{L+1} \boldsymbol{a}_{L,j}^{(k-1)} + \boldsymbol{b}_{L+1}^{(k-1)}; \boldsymbol{y}_j) + \frac{\alpha_1^{(k-1)}}{2} \|\boldsymbol{W}_{L+1} - \boldsymbol{W}_{L+1}^{(k-1)}\|_F^2 + r_{L+1}(\boldsymbol{W}_{L+1})$
    $\boldsymbol{W}_{L+1}^{(k)} = \boldsymbol{W}_{L+1}^{(k-1)} + \beta_1^{(k-1)} (\boldsymbol{W}_{L+1}^* - \boldsymbol{W}_{L+1}^{(k-1)})$
    $\boldsymbol{b}_{L+1}^* = \operatorname{argmin}_{\boldsymbol{b}_{L+1}} \gamma_{L+1} \mathcal{L}(\boldsymbol{W}_{L+1}^{(k-1)} \boldsymbol{a}_{L,j}^{(k-1)} + \boldsymbol{b}_{L+1}; \boldsymbol{y}_j) + \frac{\alpha_1^{(k-1)}}{2} \|\boldsymbol{b}_{L+1} - \boldsymbol{b}_{L+1}^{(k-1)}\|^2$
    $\boldsymbol{b}_{L+1}^{(k)} = \boldsymbol{b}_{L+1}^{(k-1)} + \beta_1^{(k-1)} (\boldsymbol{b}_{L+1}^* - \boldsymbol{b}_{L+1}^{(k-1)})$
    if $F(\cdots, \boldsymbol{W}_{L+1}^*, \boldsymbol{b}_{L+1}^*, \cdots) \leq F(\cdots, \boldsymbol{W}_{L+1}^{(k)}, \boldsymbol{b}_{L+1}^{(k)}, \cdots)$ then
        $\beta_1^{(k)} = t\beta_1^{(k-1)}$
    else
        $\beta_1^{(k)} = \min\{\beta_1^{(k-1)}/s, 1\}$
    for $\ell = L, \dots, 1$ do
        $\boldsymbol{a}_{\ell,j}^* = \operatorname{argmin}_{\boldsymbol{a}_{\ell,j}} \frac{\gamma_{\ell+1}}{2} \|\boldsymbol{W}_{\ell+1}^{(k)} \boldsymbol{a}_{\ell,j} + \boldsymbol{b}_{\ell+1}^{(k)} - \boldsymbol{a}_{\ell+1,j}^{(k)} + \boldsymbol{u}_{\ell+1,j}^{(k)}\|^2 + \frac{\gamma_\ell}{2} \|\boldsymbol{W}_\ell^{(k-1)} \boldsymbol{a}_{\ell-1,j}^{(k-1)} + \boldsymbol{b}_\ell^{(k-1)} - \boldsymbol{a}_{\ell,j} + \boldsymbol{u}_{\ell,j}^{(k)}\|^2 + \frac{\alpha_{2(L-\ell+1)}^{(k-1)}}{2} \|\boldsymbol{a}_{\ell,j} - \boldsymbol{a}_{\ell,j}^{(k-1)}\|^2 + \iota_{\mathcal{S}_\ell}(\boldsymbol{a}_{\ell,j})$ for all $j \in \{1, \dots, N\}$
        $\boldsymbol{a}_{\ell,j}^{(k)} = \boldsymbol{a}_{\ell,j}^{(k-1)} + \beta_{2(L-\ell+1)}^{(k-1)} (\boldsymbol{a}_{\ell,j}^* - \boldsymbol{a}_{\ell,j}^{(k-1)})$ for all $j \in \{1, \dots, N\}$
        if $F(\cdots, \boldsymbol{a}_{\ell,1}^*, \dots, \boldsymbol{a}_{\ell,N}^*, \cdots) \leq F(\cdots, \boldsymbol{a}_{\ell,1}^{(k)}, \dots, \boldsymbol{a}_{\ell,N}^{(k)}, \cdots)$ then
            $\beta_{2(L-\ell+1)}^{(k)} = t\beta_{2(L-\ell+1)}^{(k-1)}$
        else
            $\beta_{2(L-\ell+1)}^{(k)} = \min\{\beta_{2(L-\ell+1)}^{(k-1)}/s, 1\}$
        $\boldsymbol{u}_{\ell,j}^{(k)} = \operatorname{argmin}_{\boldsymbol{u}_{\ell,j}} \frac{\gamma_\ell}{2} \|\boldsymbol{W}_\ell^{(k-1)} \boldsymbol{a}_{\ell-1,j}^{(k-1)} + \boldsymbol{b}_\ell^{(k-1)} - \boldsymbol{a}_{\ell,j}^{(k)} + \boldsymbol{u}_{\ell,j}\|^2$
        $\boldsymbol{W}_\ell^* = \operatorname{argmin}_{\boldsymbol{W}_\ell} \sum_{j=1}^N \frac{\gamma_\ell}{2} \|\boldsymbol{W}_\ell \boldsymbol{a}_{\ell-1,j}^{(k-1)} + \boldsymbol{b}_\ell^{(k-1)} - \boldsymbol{a}_{\ell,j}^{(k)} + \boldsymbol{u}_{\ell,j}^{(k)}\|^2 + \frac{\alpha_{2(L-\ell+1)+1}^{(k-1)}}{2} \|\boldsymbol{W}_\ell - \boldsymbol{W}_\ell^{(k-1)}\|_F^2 + r_\ell(\boldsymbol{W}_\ell)$
        $\boldsymbol{W}_\ell^{(k)} = \boldsymbol{W}_\ell^{(k-1)} + \beta_{2(L-\ell+1)+1}^{(k)} (\boldsymbol{W}_\ell^* - \boldsymbol{W}_\ell^{(k-1)})$
        $\boldsymbol{b}_\ell^* = \operatorname{argmin}_{\boldsymbol{b}_\ell} \sum_{j=1}^N \frac{\gamma_\ell}{2} \|\boldsymbol{W}_\ell^{(k-1)} \boldsymbol{a}_{\ell-1,j}^{(k-1)} + \boldsymbol{b}_\ell - \boldsymbol{a}_{\ell,j}^{(k)} + \boldsymbol{u}_{\ell,j}^{(k)}\|^2 + \frac{\alpha_{2(L-\ell+1)+1}^{(k-1)}}{2} \|\boldsymbol{b}_\ell - \boldsymbol{b}_\ell^{(k-1)}\|^2$
        $\boldsymbol{b}_\ell^{(k)} = \boldsymbol{b}_\ell^{(k-1)} + \beta_{2(L-\ell+1)+1}^{(k-1)} (\boldsymbol{b}_\ell^* - \boldsymbol{b}_\ell^{(k-1)})$
        if $F(\cdots, \boldsymbol{W}_\ell^*, \boldsymbol{b}_\ell^*, \cdots) \leq F(\cdots, \boldsymbol{W}_\ell^{(k)}, \boldsymbol{b}_\ell^{(k)}, \cdots)$ then
            $\beta_{2(L-\ell+1)+1}^{(k)} = t\beta_{2(L-\ell+1)+1}^{(k-1)}$
        else
            $\beta_{2(L-\ell+1)+1}^{(k)} = \min\{\beta_{2(L-\ell+1)+1}^{(k-1)}/s, 1\}$
Output: $\mathcal{W}$, $\boldsymbol{b}$
```

A.3 Implementation details

For instance, if we take $r(\boldsymbol{W}_\ell) \equiv \lambda \|\boldsymbol{W}_\ell\|_F^2$, then we have

$$\begin{aligned}\boldsymbol{W}_{L+1}^* = &\left(\alpha_1^{(k-1)} \boldsymbol{W}_{L+1}^{(k-1)} + \gamma_{L+1} \sum_{j=1}^{N} (\boldsymbol{y}_j - \boldsymbol{b}_{L+1}^{(k-1)})(\boldsymbol{a}_{L,j}^{(k-1)})^\top \right) \\ &\qquad \left((\alpha_1^{(k-1)} + \lambda) \boldsymbol{I}_{d_L} + \gamma_{L+1} \sum_{j=1}^{N} \boldsymbol{a}_{L,j}^{(k-1)} (\boldsymbol{a}_{L,j}^{(k-1)})^\top \right)^{-1}\end{aligned}$$

$$\boldsymbol{b}_{L+1}^* = \frac{1}{1 + \alpha_1^{(k-1)}} \left(\alpha_1^{(k-1)} \boldsymbol{b}_{L+1}^{(k-1)} + \gamma_{L+1} \sum_{j=1}^{N} (\boldsymbol{y}_j - \boldsymbol{W}_{L+1}^{(k-1)} \boldsymbol{a}_{L,j}^{(k-1)}) \right)$$

For all $j \in \{1, \ldots, N\}$, and for $\ell = L, \ldots, 1$,

$$\begin{aligned}\boldsymbol{a}_{\ell,j}^* = &\left(\gamma_{\ell+1} \boldsymbol{W}_{\ell+1}^{(k)} (\boldsymbol{W}_{\ell+1}^{(k)})^\top + (\alpha_{2(L-\ell+1)}^{(k-1)} + \gamma_\ell) \boldsymbol{I}_{d_\ell} \right)^{-1} \Big(\gamma_{\ell+1} (\boldsymbol{W}_{\ell+1}^{(k)})^\top (\boldsymbol{a}_{\ell+1,j}^{(k)} - \boldsymbol{b}_{\ell+1}^{(k)}) \\ &\qquad + \gamma_\ell (\boldsymbol{W}_\ell^{(k-1)} \boldsymbol{a}_{\ell-1,j}^{(k-1)} + \boldsymbol{b}_\ell^{(k-1)} + \boldsymbol{u}_{\ell,j}^{(k-1)}) + \alpha_{2(L-\ell+1)}^{(k-1)} \boldsymbol{a}_{\ell,j}^{(k-1)} \Big)\end{aligned}$$

$$\begin{aligned}\boldsymbol{a}_{\ell,j}^* &= \mathcal{P}_{\mathcal{S}_\ell}(\boldsymbol{a}_{\ell,j}^*) \\ \boldsymbol{u}_{\ell,j}^* &= \boldsymbol{a}_{\ell,j}^{(k)} - \boldsymbol{W}_\ell^{(k-1)} \boldsymbol{a}_{\ell-1,j}^{(k-1)} - \boldsymbol{b}_\ell^{(k-1)}\end{aligned}$$

$$\begin{aligned}\boldsymbol{W}_\ell^* = &\left(\alpha_{2(L-\ell+1)+1}^{(k-1)} \boldsymbol{W}_\ell^{(k-1)} + \gamma_\ell \sum_{j=1}^{N} (\boldsymbol{a}_{\ell,j}^{(k)} - \boldsymbol{b}_\ell^{(k-1)} - \boldsymbol{u}_{\ell,j}^{(k)})(\boldsymbol{a}_{\ell-1,j}^{(k-1)})^\top \right) \\ &\qquad \left(\alpha_{2(L-\ell+1)+1}^{(k-1)} \boldsymbol{I}_{d_{\ell-1}} + \gamma_\ell \sum_{j=1}^{N} \boldsymbol{a}_{\ell-1,j}^{(k-1)} (\boldsymbol{a}_{\ell-1,j}^{(k-1)})^\top \right)^{-1}\end{aligned}$$

$$\boldsymbol{b}_\ell^* = \frac{1}{\gamma_\ell N + \alpha_{2(L-\ell+1)+1}^{(k-1)}} \left(\alpha_{2(L-\ell+1)+1}^{(k-1)} \boldsymbol{b}_\ell^{(k-1)} + \gamma_\ell \sum_{j=1}^{N} (\boldsymbol{a}_{\ell,j}^{(k)} - \boldsymbol{W}_\ell^{(k-1)} \boldsymbol{a}_{\ell-1,j}^{(k-1)}) \right)$$

A.4 Assumptions, definitions, propositions and related proofs

Assumption 1 *We have several assumptions on the functions $\widetilde{F}$: (i) The loss function $\mathcal{L}$ is chosen such that $\widetilde{F}$ is continuous*[2] *and bounded below on* $\operatorname{dom} \widetilde{F}$. *Problem* (3) *has a Nash point (Xu & Yin, 2013); (ii) For each block i, there exist constant* $0 < a_i \le A_i < \infty$ *such that the proximal parameters* $\alpha_i^{(k-1)}$ *obeys* $a_i \le \alpha_i^{(k-1)} \le A_i$.

Definition 2 (Kurdyka-Łojasiewicz property and KL function (Bolte et al., 2014))

1. The function $F : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is said to have the *Kurdyka-Łojasiewicz property* at $\bar{\boldsymbol{x}} \in \operatorname{dom} \partial F$ if there exist $\eta \in (0, +\infty]$, a neighborhood $\mathcal{B}_\rho(\bar{\boldsymbol{x}}) := \{\boldsymbol{x} : \|\boldsymbol{x} - \bar{\boldsymbol{x}}\| < \rho\}$ of $\bar{\boldsymbol{x}}$ and a continuous concave function $\varphi(t) := ct^{1-\theta}$ for some $c > 0$ and $\theta \in [0, 1)$ such that for all $\boldsymbol{x} \in \mathcal{B}_\rho(\bar{\boldsymbol{x}}) \cap [F(\bar{\boldsymbol{x}}) < F < F(\bar{\boldsymbol{x}}) + \eta]$, the Kurdyka-Łojasiewicz inequality holds

$$\varphi'(F(\boldsymbol{x}) - F(\bar{\boldsymbol{x}})) \operatorname{dist}(\mathbf{0}, \partial F(\boldsymbol{x})) \ge 1.$$

[2] Note that the indicator function $\iota_{\mathcal{S}_\ell}$ is continuous on $\operatorname{dom}(\iota_{\mathcal{S}_\ell})$ since according to its definition, $\iota_{\mathcal{S}_\ell}(\boldsymbol{a}_{\ell,j}) = 0$ if $\boldsymbol{a}_{\ell,j} \in \operatorname{dom}(\iota_{\mathcal{S}_\ell}) \equiv \mathbb{R}_+^{d_\ell}$ and $\iota_{\mathcal{S}_\ell}(\boldsymbol{a}_{\ell,j}) = \infty$ otherwise, for all $j \in \{1, \ldots, N\}$ and $\ell \in \{1, \ldots, L\}$. In general, the indicator function ι is called an *extended-value* convex function since it equals $+\infty$ if the variable is not in the domain of ι.

2. Proper lower semincontinuous functions which satisfy the Kurdyka-Łojasiewicz inequality at each point of $\operatorname{dom} \partial F$ are called *KL functions*.

Definition 3 (Semialgebraic function)

1. A set $\mathcal{D} \subset \mathbb{R}^n$ is called semialgebraic (Bochnak et al., 1998) if it can be represented as

$$\mathcal{D} = \bigcup_{i=1}^{s} \bigcap_{j=1}^{t} \left\{ \boldsymbol{x} \in \mathbb{R}^n : p_{ij}(\boldsymbol{x}) = 0, q_{ij}(\boldsymbol{x}) > 0 \right\},$$

where p_{ij}, q_{ij} are real polynomial functions for $1 \leq i \leq s, 1 \leq j \leq t$.

2. A function h is called *semialgebraic* if its graph $\operatorname{Gr}(h) := \{(x, h(x)) : x \in \operatorname{dom}(h)\}$ is a semialgebraic set.

Remark 1 KL functions include real analytic functions, functions on the o-minimal structure (Kurdyka, 1998), subanalytic functions (Bolte et al., 2007a;b), semialgebraic functions (see Definition 3) and locally strongly convex functions. Some other important facts regarding KL functions available in Łojasiewicz (1963; 1993); Kurdyka (1998); Bolte et al. (2007a;b); Attouch et al. (2013); Xu & Yin (2013) and references therein are summarized below.

1. The sum of a real analytic function and a semialgebraic function is a subanalytic function, and thus a KL function.

2. If a set $\mathcal{D}$ is semialgebraic, so is its closure $\operatorname{cl}(\mathcal{D})$.

3. If $\mathcal{D}_1$ and $\mathcal{D}_2$ are both semialgebraic, so are $\mathcal{D}_1 \cup \mathcal{D}_2$, $\mathcal{D}_1 \cap \mathcal{D}_2$, and $\mathbb{R}^n \setminus \mathcal{D}_1$.

4. Indicator functions of semialgebraic sets are semialgebraic, e.g., the indicator functions of nonnegative closed half space and a nonempty closed interval.

5. Finite sums and products of semialgebraic (real analytic) functions are semialgebraic (real analytic).

6. The composition of semialgebraic functions is semialgebraic.

Proposition 1 *The objective function* $\widetilde{F}$ *in* (3) *is a KL function, if the loss function* $\mathcal{L}$ *is chosen as one of the commonly used loss functions such as the squared loss, logistic loss, hinge loss or softmax cross-entropy loss, and the regularizers* r_ℓ*'s are chosen as* ℓ_1 *norms, squared* ℓ_2 *norms (a.k.a. weight decay),* ℓ_q *quasi-norms with* $q \in [0, 1)$*, or their sums such as the elastic net (Zou & Hastie, 2005).*

PROOF Recall that

$$\widetilde{F}(\boldsymbol{a}, \mathcal{W}, \boldsymbol{b}, \boldsymbol{u}) \equiv \underbrace{\gamma_{L+1} \sum_{j=1}^{N} \mathcal{L}(\boldsymbol{W}_{L+1} \boldsymbol{a}_{L,j} + \boldsymbol{b}_{L+1}; \boldsymbol{y}_j)}_{\widetilde{F}_1(\mathcal{W}, \boldsymbol{b})} + \underbrace{\sum_{\ell=1}^{L+1} r_\ell(\boldsymbol{W}_\ell)}_{\widetilde{F}_2(\mathcal{W})}$$

$$+ \underbrace{\sum_{j=1}^{N} \sum_{\ell=1}^{L} \frac{\gamma_\ell}{2} \| \boldsymbol{W}_\ell \boldsymbol{a}_{\ell-1,j} + \boldsymbol{b}_\ell - \boldsymbol{a}_{\ell,j} + \boldsymbol{u}_{\ell,j} \|^2}_{\widetilde{F}_3(\boldsymbol{a}, \mathcal{W}, \boldsymbol{b}, \boldsymbol{u})} + \underbrace{\sum_{j=1}^{N} \sum_{\ell=1}^{L} \iota_{\mathcal{S}_\ell}(\boldsymbol{a}_{\ell,j})}_{\widetilde{F}_4(\boldsymbol{a})}.$$

Any polynomial function is real analytic, so $\widetilde{F}_3$ is real analytic by Remark 1 item 5. In the same vein, the square loss function is also real analytic. The logistic loss and softmax cross-entropy loss are also real analytic (Xu & Yin, 2013). If $\mathcal{L}$ is the hinge loss, i.e., given $\boldsymbol{y} \in \mathbb{R}^{d_{L+1}}$, $\mathcal{L}(\boldsymbol{u}, \boldsymbol{y}) := \max\{0, 1 - \boldsymbol{u}^\top \boldsymbol{y}\}$ for any $\boldsymbol{u} \in \mathbb{R}^{d_{L+1}}$, it is semialgebraic, because its graph is $\operatorname{cl}(\mathcal{D})$, a closure of the set $\mathcal{D}$, where

$$\mathcal{D} = \{(\boldsymbol{u}, z) : 1 - \boldsymbol{u}^\top \boldsymbol{y} - z = 0, \boldsymbol{1} - \boldsymbol{u} \succ 0\} \cup \{(\boldsymbol{u}, z) : z = 0, \boldsymbol{u}^\top \boldsymbol{y} - 1 > 0\}.$$

Then again by Remark 1 item 5, $\widetilde{F}_1$ is either a real analytic function (squared, logistic and softmax cross-entropy losses) or a semialgebraic function (hinge loss). $\widetilde{F}_4$ is semialgebraic by Remark 1 items 4 and 5 since $\mathcal{S}_\ell$ is a nonempty closed interval for all $\ell \in \{1, \ldots, L\}$ depicted in Appendix A.1.

Concerning $\widetilde{F}_2$, which is the sum of the regularizers r_ℓ's, note that the ℓ_1 norm, the squared ℓ_2 norm, the ℓ_q quasi-norms with $q \in [0, 1)$ are all semialgebraic, and thus, the elastic net is also semialgebraic. By Remark 1 item 5, $\widetilde{F}_2$ is also semialgebraic.

Finally, using Remark 1 item 1, we conclude that $\widetilde{F}$ is subanalytic and hence a KL function. ■

We now present the proof of Theorem 1.

PROOF (THEOREM 1) Note that $\widetilde{F}$ is monotonically nonincreasing and converges to $\widetilde{F}(\bar{\boldsymbol{x}})$. If $\widetilde{F}(\boldsymbol{x}^{(k_0)}) = \widetilde{F}(\bar{\boldsymbol{x}})$ at some k_0, then $\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k_0)} = \bar{\boldsymbol{x}}$ for all $k \geq k_0$. It remains to consider $\widetilde{F}(\boldsymbol{x}^{(k)}) > \widetilde{F}(\bar{\boldsymbol{x}})$ for all $k \geq 0$. Since $\bar{\boldsymbol{x}}$ is a limit point and $\widetilde{F}(\boldsymbol{x}^{(k)}) \to \widetilde{F}(\bar{\boldsymbol{x}})$, there must exist an integer k_0 such that $\boldsymbol{x}^{(k_0)}$ is sufficiently close to $\bar{\boldsymbol{x}}$. Hence, $\{\boldsymbol{x}^{(k)}\}_{k \geq 1}$ converges according to Xu & Yin (2013, Lemma 2.6). ■

Theorem 2 (Convergence rate) *Suppose that $\{\boldsymbol{x}^{(k)}\}_{k\geq 1} := \{\boldsymbol{a}^{(k)}, \mathcal{W}^{(k)}, \boldsymbol{b}^{(k)}, \boldsymbol{u}^{(k)}\}_{k\geq 1}$ converges to a critical point $\bar{\boldsymbol{x}} := \{\bar{\boldsymbol{a}}, \overline{\mathcal{W}}, \bar{\boldsymbol{b}}, \bar{\boldsymbol{u}}\}$, at which $\widetilde{F}$ satisfies the KL property with $\varphi(t) := ct^{1-\theta}$ for some $c > 0$ and $\theta \in [0, 1)$. Then the following hold:*

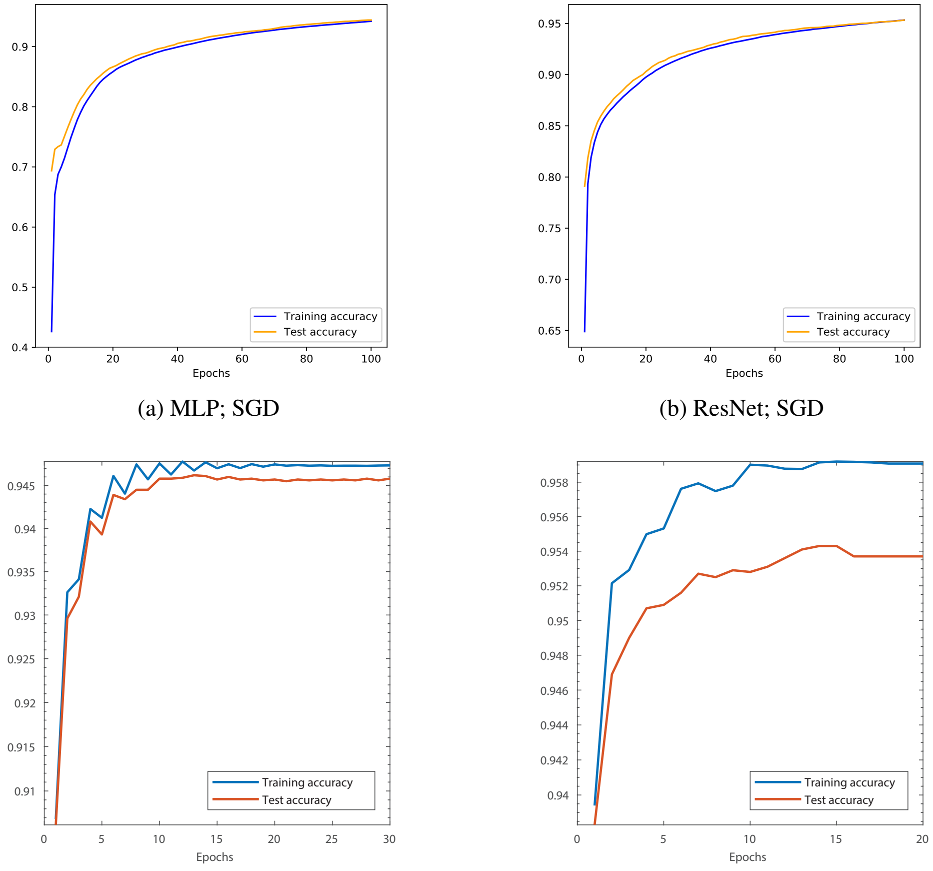
1. *If $\theta = 0$, $\boldsymbol{x}^{(k)}$ converges to $\bar{\boldsymbol{x}}$ in finitely many iterations.*
2. *If $\theta \in (0, 1/2]$, $\|\boldsymbol{x}^{(k)} - \bar{\boldsymbol{x}}\| \leq C\tau^k$ for all $k \geq k_0$, for certain $k_0 > 0$, $C > 0$, $\tau \in [0, 1)$.*
3. *If $\theta \in (1/2, 1)$, $\|\boldsymbol{x}^{(k)} - \bar{\boldsymbol{x}}\| \leq Ck^{-(1-\theta)/(2\theta-1)}$ for all $k \geq k_0$, for certain $k_0 > 0$, $C > 0$.*

These three parts correspond to finite convergence, linear convergence, and sublinear convergence, respectively.

PROOF See the proof of Theorem 2.9 of Xu & Yin (2013). ■

B Further experimental results

We further conduct experiments for two different structures on the MNIST dataset (LeCun & Cortes, 2010) with 60K training and 10K test samples, namely a 784-2048-2048-2048-10 MLP and a 784-2048-784-2048-10 DNN with a residual connection in the second hidden layer (ResNet). The BCD algorithm (30 epochs for MLP; 20 epochs for ResNet) is implemented using MATLAB while backprop (SGD; 100 epochs) is implemented using Keras with TensorFlow backend. Squared losses, ReLUs are used without regularizations. All weight matrices are initialized from a Gaussian distribution with a standard deviation of 0.01 and the bias vectors are initialized as vectors of all 0.1, while $\boldsymbol{a}$ and $\boldsymbol{u}$ are initialized by a single forward pass. The hyperparameters in BCD ($\beta_i = 0.95, \gamma_i = 0.1, t = 0.1$) and the learning rate (0.05) in SGD are chosen by manual tuning. We report the training and test accuracies (the median of 5 runs) as follows:



(a) MLP; SGD

(b) ResNet; SGD

(c) MLP; BCD: $\alpha_1 = 10^{-3}, \alpha_{2\sim 7} = 10^{-2}, s = 1$

(d) ResNet; BCD: $\alpha_1 = 1, \alpha_{3/5/7} = 5, \alpha_{\text{even}} = 10, s = 0.1$

Figure 2: Training and test accuracies. Final test acc.: 2a & 2b: 0.9533; 2c: 0.9458; 2d: 0.9537.

From Figure 2, we observe that the BCD algorithms require much fewer epochs to achieve similar test accuracies. Thus, we say that the BCD method has high per epoch efficiency compared to backprop with SGD.