Neural Networks 139 (2021) 348–357

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

# Block-cyclic stochastic coordinate descent for deep neural networks

Kensuke Nakamura [a], Stefano Soatto [b], Byung-Woo Hong [a,b,*]

[a] Computer Science Department, Chung-Ang University, Seoul, Republic of Korea
[b] Computer Science Department, University of California Los Angeles, CA, USA

A R T I C L E I N F O

*Article history:*
Received 4 August 2020
Received in revised form 18 February 2021
Accepted 2 April 2021
Available online 19 April 2021

*Keywords:*
Coordinate descent
Deep neural network
Energy optimization
Stochastic gradient descent

A B S T R A C T

We present a stochastic first-order optimization algorithm, named block-cyclic stochastic coordinate descent (BCSC), that adds a cyclic constraint to stochastic block-coordinate descent in the selection of both data and parameters. It uses different subsets of the data to update different subsets of the parameters, thus limiting the detrimental effect of outliers in the training set. Empirical tests in image classification benchmark datasets show that BCSC outperforms state-of-the-art optimization methods in generalization leading to higher accuracy within the same number of update iterations. The improvements are consistent across different architectures and datasets, and can be combined with other training techniques and regularizations.

## 1. Introduction

The two workhorses of Deep Learning, responsible for much of the remarkable progress in traditionally challenging machine-learning problems, are SGD (stochastic gradient descent) and GSD (graduate student descent). The latter has produced an ever-growing body of neural network architectures, starting from basic shallow convolutional ones (LeCun, Bottou, Bengio, & Haffner, 1998) to non-Markov ones (Balduzzi, Frean, Leary, Lewis, Ma, & McWilliams, 2017; He, Zhang, Ren, & Sun, 2016a, 2016b), and still growing deeper (Chen, Li, Xiao, Jin, Yan, & Feng, 2017; Hu, Shen, & Sun, 2017; Huang, Liu, van der Maaten, & Weinberger, 2017). The former has been the subject of intense scrutiny, despite its simplicity, both in terms of unraveling the mysteries behind its unreasonable effectiveness, as well as fostering a cottage industry of modifications and improvements. Our work is squarely in the latter vein.

SGD (Robbins & Monro, 1951; Rumelhart, Hinton, Williams, et al., 1988; Zhang, 2004) is a simple variant of classical gradient descent where the stochasticity comes from employing a random subset of the measurements (mini-batch) to compute the gradient at each step of descent. This has the complexity of $\mathcal{O}(1)$ in the total example size, that is usually in the tens of thousands to millions. It also has implicit regularization effects, making it suited for highly non-convex loss functions, such as those entailed in training deep networks for classification.

The entire process of training is sensitive to outlier data such as erroneous labeling in the training set, as each mini-batch affects the update of the entire set of parameters. The mini-batch size is usually small, thus the relative impact of an outlier can be large compared to the full batch gradient. There are a number of techniques such as adaptive learning rate, regularization, and some gradient descent designed for weakening the impact of outliers, but they mainly aim to normalize the variation of mini-batches and cannot manipulate training outliers explicitly. Stochastic methods such as randomized block coordinate descent (SBC) (Wan, Zeiler, Zhang, Cun, & Fergus, 2013; Wang & Banerjee, 2014; Zhao, Yu, Wang, Arora, & Liu, 2014), on the other hand, trade off accuracy with robustness to noise. Our objective is to develop an accurate optimization algorithm for deep learning that is not subject to such a strict tradeoff.

In the proposed algorithm, we leverage randomized methods based on stochastic randomized block coordinate descent (Wan et al., 2013; Wang & Banerjee, 2014; Zhao et al., 2014), but introduce a cyclic constraint in the selection of both measurements and model parameters, so that different mini-batches of data are used to update different subsets of the unknown parameters. We perform numerical experiments using networks from shallow to recently developed deeper models based on popular image classification benchmark sets, and demonstrate that our algorithm consistently outperforms the state-of-the-art optimizations for all the network models under consideration.

In Section 2 we place our contribution in context, and provide the problem of interest and relevant works in Section 3. The details on the proposed algorithm are presented in Section 4. In Section 5 we report experiments to compare with the state-of-the-art, and discuss limitations and potential extensions in Section 6.

* Corresponding author at: Computer Science Department, Chung-Ang University, Seoul, Republic of Korea.
*E-mail address:* hong@cau.ac.kr (B.-W. Hong).

https://doi.org/10.1016/j.neunet.2021.04.001
0893-6080/© 2021 Elsevier Ltd. All rights reserved.

## 2. Related work

**Adaptive step size methods** In SGD, the current parameter estimate is updated by subtracting the (approximate) gradient multiplied by a factor, the *learning rate*. Since SGD does not converge to a point estimate, the learning rate usually decreases over iterations monotonically to reduce fluctuation of loss. While it is still common in practice to modulate the learning rate based on a fixed schedule, several adaptive learning functions have been studied to automate the scheduling (George & Powell, 2006). Some of the best known methods include AdaGrad (Duchi, Hazan, & Singer, 2011) and AdaDelta (Schaul, Zhang, & LeCun, 2013; Zeiler, 2012). They reduce the learning rate by accumulating the gradient of the loss function globally (Duchi et al., 2011) or parameter-wise (Schaul et al., 2013; Zeiler, 2012). For the adaptive scheduling of the learning rate, the interpolation with a random sampling technique has been used to compute the step size (De, Yadav, Jacobs, & Goldstein, 2016; Tan, Ma, Dai, & Qian, 2016). The adaptive change of learning rate has demonstrated numerical benefit favoring stable and faster convergence. However, as we will show empirically, our proposed algorithm outperforms the conventional SGD algorithms with the state-of-the-art scheme of the adaptive learning rate in terms of both training and testing losses. Moreover, our approach can be naturally integrated with the adaptive learning rate scheme and take its numerical advantage.

**Regularization methods** There are a number of ways to impose regularity to the model in order to improve generalization for better prediction, among which are data augmentation (An, 1996; Simonyan & Zisserman, 2014), batch normalization (Hoffer, Hubara, & Soudry, 2017; Ioffe & Szegedy, 2015), or dropout (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012; Huang, Sun, Liu, Sedra, & Weinberger, 2016; Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014; Wan et al., 2013). One can also incorporate regularization in the network architectures, including pooling (Krizhevsky, Sutskever, & Hinton, 2012), maxout (Goodfellow, Warde-Farley, Mirza, Courville, & Bengio, 2013), or skip connections (Huang et al., 2017; Long, Shelhamer, & Darrell, 2015). There is also an explicit regularization that is integrated with the objective function with classical weight decay (Lang & Hinton, 1990; Plaut et al., 1986), lasso (Tibshirani, 1996), group lasso (Yuan & Lin, 2006), or Hessian (Rifai, Glorot, Bengio, & Vincent, 2011). Our method acts in concert, not in alternative, to other forms of regularization. Yet, it can implicitly provides better generalization resulting in higher accuracy due to the way how it deals with the selection of example data in optimization.

**Variants of gradient descent** Stochastic average gradient (SAG) (Roux, Schmidt, & Bach, 2012) calculates the gradient using a randomly chosen subset of the examples and then averages their gradients in the estimation of the full gradient. Stochastic variance reduced gradient (SVRG) (Johnson & Zhang, 2013) considers the inherent variance of the gradient or the difference between the gradients of a mini-batch and the full gradient. Both SAG (Roux et al., 2012) and SVRG (Johnson & Zhang, 2013) are approximations of the standard gradient and would be subject to its same limitations in large scale optimization problems for non-convex objective functions. A variety of first-order stochastic algorithms has been developed for parallel computation (Zhang, Choromanska, & LeCun, 2015) or proximal operators (Duchi & Singer, 2009). Similar to SGD that randomly selects subsets of data, stochasticity has been applied in selecting subsets of parameters to update by randomized block coordinate descent (BCD) (Nesterov, 2012; Richtárik & Takáč, 2014). Such a technique has been used to train neural networks in Liu, Yan, Wang, and Zha (2016) utilizing parallel computation.

The term, block coordinate, is also used as the split variables in ADMM-like optimization for deep networks (Lau, Zeng, Wu, & Yao, 2018; Taylor, Burmeister, Xu, Singh, Patel, & Goldstein, 2016; Zeng, Lau, Lin, & Yao, 2019; Zhang & Brand, 2017). These methods are intended to update the model without the back-propagation that can cause the gradient vanishing. We consider to split the model parameters into blocks and update them using different data in order to impose a regularization effect.

Our algorithm is closely related to stochastic (randomized) block coordinate descent (SBC) (Wan et al., 2013; Wang & Banerjee, 2014; Zhao et al., 2014) which randomly chooses both parameters and examples in the optimization procedure. However, when the number of parameters is in the millions, there is a trade-off between accuracy and robustness to outliers. To mitigate this issue, we introduce a cyclic procedure such that a parameter is updated only once with each sample within an epoch. This is, however, different from classical cyclic coordinate descent (Saha & Tewari, 2010) and recent cyclic-block coordinate descents (Lee & Wright, 2016; Leventhal & Lewis, 2010; Richtárik & Takáč, 2016; Saha & Tewari, 2013), since we consider mini-batches of both the data and the parameters. Furthermore, our goal is not to approximate the full gradient, as in Wang and Banerjee (2014), Zhao et al. (2014). Instead, we aim to modify the stochastic procedure to achieve better regularization, hence higher accuracy.

## 3. Preliminaries

Let $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ be a set of training data where $x_i \in X$ is an input, typically an image, and $y_i \in Y$ is an output, typically a label. Let $h_w : X \to Y$ be a prediction function with the associated model parameters $w = (w_1, w_2, \ldots, w_m) \in \mathbb{R}^m$ where the dimension of the feature space is $m$. The discrepancy between the predicted output $h_w(x_i)$ and the true output $y_i$ is measured by a loss function $\ell(h_w(x_i), y_i)$ for each training sample $(x_i, y_i)$. The goal is to find optimal parameters $w^*$ that are typically obtained by minimizing the empirical loss $\mathcal{L}(w)$ on the dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$:

$$\mathcal{L}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(h_w(x_i), y_i) = \frac{1}{n}\sum_{i=1}^{n} f_i(w), \tag{1}$$

$$w^* = \arg\min_{w} \mathcal{L}(w), \tag{2}$$

where the loss incurred by the parameters $w$ with sample $(x_i, y_i)$ is denoted by $f_i(w) := \ell(h_w(x_i), y_i)$.

### 3.1. Stochastic gradient descent

The minimization of $\mathcal{L}(w)$ in Eq. (1), assuming $f_i(w)$ is differentiable, involves the computation of the gradient for a large number $n$ of training data. Stochastic gradient descent (SGD) (Robbins & Monro, 1951; Rumelhart et al., 1988; Zhang, 2004) achieves the dual objective of reducing the computational load as well as improving generalization due to the implicit regularization effect (Zhu, Wu, Yu, Wu, & Ma, 2019). The stochastic process of sampling subsets of data at each iteration leads to regularization in the estimation of the gradient for the expected loss. Let $\chi = \{1, 2, \ldots, n\}$ be the index set of the training data and $\beta \subset \chi$ be its random subset, called the mini-batch. SGD updates an initial estimate (typically random) of the weights recursively at each iteration $t$ via

$$w^{(t+1)} := w^{(t)} - \eta^{(t)} \frac{1}{|\beta^{(t)}|} \sum_{i \in \beta^{(t)}} \nabla f_i(w^{(t)}), \tag{3}$$

where $\eta^{(t)}$ is a positive scalar, called learning rate. Manual scheduling of the learning rate is typical, although adaptive scheduling schemes based on the gradient or the iteration are also considered (Duchi et al., 2011; Schaul et al., 2013; Zeiler, 2012).

### 3.2. Random coordinate descent

In the optimization of deep neural networks, it is often required to compute loss function $\{f_i(w)\}_{i=1}^n$ with respect to a large number $m$ of parameters $w \in \mathbb{R}^m$ in addition to dealing with a large number $n$ of data. Randomized block coordinate descent (BCD) (Nesterov, 2012; Richtárik & Takáč, 2014) selects a subset $c$ from the index set $\{1, 2, \dots, m\}$ of the feature space uniformly at random and computes gradients $\nabla_{w_c}\mathcal{L}(w)$ restricted to the selected subset $w_c$ of the coordinates using the set of loss functions $\{f_i\}_{i=1}^n$ on the whole data set. Then, the only selected parameters $w_c$ are updated based on the gradient $\nabla_{w_c}\mathcal{L}(w)$. The BCD algorithm proceeds at each iteration $t$ via

$$w_{c^{(t)}}^{(t+1)} := w_{c^{(t)}}^{(t)} - \eta^{(t)}\frac{1}{n}\sum_{i=1}^{n}\nabla_{w_{c^{(t)}}} f_i(w^{(t)}), \tag{4}$$

while keeping unselected parameters: $w_j^{(t+1)} = w_j^{(t)}, \forall j \notin c^{(t)}$.

### 3.3. Stochastic random coordinate descent

It is natural to consider combining the use of random mini-batches of data as done by SGD in Section 3.1 with random subsets of coordinates as done by BCD in Section 3.2. The resulting algorithm, called stochastic random block coordinate descent (SBC) (Wan et al., 2013; Wang & Banerjee, 2014; Zhao et al., 2014), selects subsets of the training data uniformly at random and computes the gradient of the objective function with respect to a randomly chosen subset of the parameters:

$$w_{c^{(t)}}^{(t+1)} := w_{c^{(t)}}^{(t)} - \eta^{(t)}\frac{1}{|\beta^{(t)}|}\sum_{i\in\beta^{(t)}}\nabla_{w_{c^{(t)}}} f_i(w^{(t)}). \tag{5}$$

While it is reasonable to expect that the regularizing effects of mini-batching would compound the computational benefits of block-descent, it is less obvious that connecting the random selections so that different sets of data are used to update different set of parameters would be beneficial. We present our proposed algorithm in the following section.

## 4. Block-cyclic stochastic coordinate descent

The essential motivation of our proposed algorithm is to combine the two types of algorithms, SGD and BCD, in such a way that SGD is designed to feed random subsets of data in the computation of gradient and BCD is designed to select random subsets of parameters to update. The combination of the two stochastic processes allows to use different subsets of data to update different subsets of parameters. We also introduce a constraint that allows the algorithm to end up using all the training example data to update each of model parameters and updating all the parameters at each epoch. We call the resulting algorithm block-cyclic stochastic coordinate descent (BCSC), which entails a doubly-stochastic process with randomization of both mini-batches of data and parameter blocks based on the cyclic block structure.

### 4.1. Cyclic block structure

We model the block structure of coordinates by decomposing the feature space $\mathbb{R}^m$ into $M$ subspaces. Let $P$ be an $m \times m$ permutation matrix and $P = [P_1|P_2|\cdots|P_M]$ be a decomposition of $P$ into a set of $M$ column blocks with $P_j$ of size $m \times m_j$, where $\sum_{j=1}^M m_j = m$. For a random selection of the elements from a feature vector with all the other elements being zero, we define a random selection matrix $Q_j$ with size $m \times m$ for a column block $P_j$ of the permutation matrix $P$ as follows: $Q_j = \left[P_j|O_j\right]$, where $O_j$ is a zero matrix with size of $m \times (m - m_j)$. For a given feature vector $w \in \mathbb{R}^m$, it can be uniquely written as $w = \sum_{j=1}^M Q_j Q_j^T w$. The iterative selection of a parameter block $w_{[j]} = P_j^T w$ from the elements in $w$ over $j$ considers all the elements in $w$ exhaustively with being mutually disjoint across blocks.

### 4.2. Doubly cyclic stochastic process

In the optimization procedure, one can consider a single stochastic process in the selection of mini-batch $\beta^{(t)}$ with a given cyclic block structure $P = [P_1|P_2|\cdots|P_M]$ for a random grouping of elements in parameter vector $w$, namely random block coordinate descent (RBC) algorithms, where the same mini-batch $\beta^{(t)}$ is used to update all the sequential blocks of parameters $w_{[j]} = P_j^T w$ in an iterative way. The RBC algorithm iterates over each $j$ with a fixed $t$ as follows:

$$G^{(t,j)} := \frac{1}{|\beta^{(t)}|}\sum_{i\in\beta^{(t)}}\nabla f_i(w^{(t,j)}), \tag{6}$$

$$w^{(t,j+1)} := w^{(t,j)} - \eta^{(t)} Q_j^T G^{(t,j)}, \tag{7}$$

where $G^{(t,j)}$ denotes the gradient of the objective function based on a mini-batch $\beta^{(t)}$, and it is assumed that $\cup_t \beta^{(t)}$ is the whole index set $\chi$ of the training data and mini-batches are mutually disjoint $\beta^{(t)} \cap \beta^{(s)} = \varnothing$ if $t \neq s$. However, this approach is ineffective in the presence of outliers that may corrupt the estimation of the gradient for the entire set of parameters.

The algorithm we propose, called block-cyclic stochastic coordinate descent (BCSC), is developed based on the doubly cyclic stochastic process within the selection of both mini-batch from the training set and coordinate block from the parameters. It is designed to ensure that each random block $w_{[j]} = P_j^T w$ of the parameters $w$ is updated following the independent stochastic selection of mini-batch $\beta^{(t)}$. In addition, each element in the training data ends up being used to update all the parameters within an epoch. Our BCSC algorithm proceeds with the doubly stochastic process to select both $\beta^{(t,j)}$ and $Q_j$ as follows:

$$G^{(t,j)} := \frac{1}{|\beta^{(t,j)}|}\sum_{i\in\beta^{(t,j)}}\nabla f_i(w^{(t,j)}), \tag{8}$$

$$w^{(t,j+1)} := w^{(t,j)} - \eta^{(t)} Q_j^T G^{(t,j)}, \tag{9}$$

subject to $\cup_t \beta^{(t,j)}$ is the whole index set of the training data and $\beta^{(t,j)} \cap \beta^{(s,j)} = \varnothing$ if $t \neq s$ at fixed $j$. Note that $P$ is randomly generated at each epoch and the index sets $\{\chi_j\}_{j=1}^M$ of the training data are also randomly shuffled at each epoch.

### 4.3. Our proposed algorithm

The central idea of the algorithm we propose is to use different subsets of data (mini-batches) to update different subsets (blocks) of parameters. This is graphically illustrated in Fig. 1 where SGD and RBC use the same mini-batch at each iteration, whereas our BCSC uses different mini-batches to update different blocks. In our algorithm, the number of partitions ($M$) in a sequence of mutually disjoint subsets in both the training data and the model parameters is related to the distributive effect of the undesired perturbations with unknown variances in the training data with which the stochastic gradients are computed. Thus, the effect of

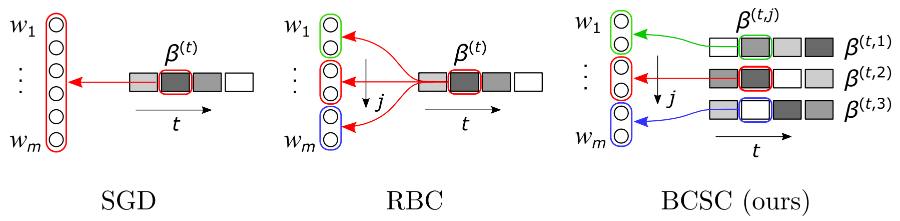

**Fig. 1. Graphical illustration of the algorithms.** SGD updates all the parameters $(w_1, w_2, \dots, w_m)$ at once using each mini-batch $\beta^{(t)}$. RBC updates sequential random subsets of parameters using the same mini-batch $\beta^{(t)}$. Our BCSC uses different mini-batches $\beta(t,j)$ to update different blocks $w_{[j]} = P_j^T w$ of parameters $w$ where $j$ denotes the index of parameter block and $t$ denotes the index of mini-batch. Here we illustrate our algorithm using the number of blocks $M = 3$ as an instance. Note that each of training example is used to update each parameter exactly once in an epoch. It is not guaranteed but usually different parameter blocks are updated using different data at each epoch due to the nature of the mini-batch procedure.

the noise process involved in the computation of gradients is more weakly distributed over the entire model parameter space leading to better accuracy due to the fact that incorrect gradients are only applied to a limited subset of model parameters while the entire model parameters are updated with the conventional stochastic gradient descent algorithm. The expected regularization effect of BCSC is related to SBC (Wan et al., 2013; Wang & Banerjee, 2014; Zhao et al., 2014) that randomly chooses both parameters and examples. Different from SBC, we update all the parameter blocks simultaneously using different mini-batches such that an outlier affects a small part of the model explicitly.

More details are described in Algorithm 1 where $M$ is a given number of partitions in the parameters. The algorithm proceeds with the initialization for the $M$ index sets $\{\chi_j\}_{j=1}^{M}$ of training data and for the permutation matrix $P$ at each epoch. Then, different mini-batches $\beta^{(t,j)}$ are taken from the data $\chi_j$ to update different blocks $w_{[j]} = P_j^T w$ of parameters $w$ followed by the update of the index set $\chi_j$ by excluding the mini-batch $\beta^{(t,j)}$ from $\chi_j$. At each iteration $t$ of mini-batches, the parameter updates are performed $M$ times for parameter blocks indexed by $j$.

**Algorithm 1** Block-Cyclic Stochastic Coordinate Descent (BCSC)

```
for all epoch do
  $\{\chi_j\}_{j=1}^{M}$ : $M$ index sets $\chi_j$ of data by random shuffling.
  $P = [P_1|P_2|\cdots|P_M]$ : random permutation matrix.
  for all $t$ : index for mini-batch do
    for all $j$ : index for parameter block do
      Take mini-batch $\beta^{(t,j)}$ from $\chi_j$.
      Take parameter block $w_{[j]} = P_j^T w$ using $P_j$.
      Compute gradient of the loss to $w_{[j]}$ using $\{f_i\}_{i \in \beta^{(t,j)}}$.
      Update parameter block $w_{[j]}$ using Eq. (9).
      Update index set $\chi_j := \chi_j \setminus \beta^{(t,j)}$.
    end for
  end for
end for
```

## 5. Experimental results

We provide quantitative and qualitative evaluation of our algorithm in comparison to the state-of-the-art optimization algorithms on the datasets including MNIST (LeCun et al., 1998), Cifar10 (Krizhevsky & Hinton, 2009) and Cifar100 (Krizhevsky & Hinton, 2009). MNIST consists of 60,000 training and 10,000 testing images with 10 labels. Cifar10 and Cifar100 are more challenging datasets that consist of 50,000 training and 10,000 test data with 10 and 100 labels, respectively. In order to provide better understanding on the effectiveness and robustness of our algorithm, we consider a variety of neural network architectures ranging from simple to deep and wide models; LeNet4 (LeCun et al., 1998), VGG19 (Simonyan & Zisserman, 2014), GoogLeNet (Szegedy et al., 2015), ResNet18 (He et al., 2016a, 2016b), ResNeXt29 (Xie, Girshick, Dollár, Tu, & He, 2016), MobileNet (Howard et al., 2017), ShuffleNet (Zhang, Zhou, Lin, & Sun, 2017), SENet18 (Hu et al., 2017), DPN92 (Chen et al., 2017), and DenseConv (Huang et al., 2017).

The performance of our BCSC algorithm is compared with other state-of-the-art optimization algorithms including stochastic gradient descent (SGD) (Robbins & Monro, 1951; Rumelhart et al., 1988; Zhang, 2004) that is the de-facto standard in deep learning optimization, stochastic randomized block-coordinate descent (SBC) (Wan et al., 2013; Wang & Banerjee, 2014; Zhao et al., 2014), and randomized block-coordinate descent (RBC) within the same number of update-iterations that is the essential measure of algorithmic time complexity. Note that the time complexity of our BCSC is the same as SGD. However, BCSC requires an additional computational load due to that implementation issue of the deep learning platforms that do not support the random choice of both model parameters and data in the back-propagation operation. We thus involve RBC with the same computational load in our experiments in order to highlight the benefit of our idea that updates different parameter blocks using different data.

For each experiment, we provide the learning curves that consist of the training loss of training mini-batches, and the test loss and the test accuracy for the validation set. In addition, the standard deviation of the training loss computed from the mini-batches within each epoch is also presented. The learning curves are shown in colors; training loss in blue, test loss in red, and test accuracy in green, and they are plotted with epoch in log scale. The train and test losses are displayed with respect to the left vertical axis and the percentile accuracy is displayed with respect to the right vertical axis. As quantitative comparison, the test accuracy is computed within the first half epochs, the last half epochs, all the epochs, and the final epoch.

For the selection of the hyper-parameters associated with the optimization algorithms, we use the customary values; mini-batch size is 128, momentum is 0.9, weight decay is $5 \times 10^{-4}$, and the total number of epochs is 200. For the learning rate, we employ a manual scheduling that is empirically optimized with respect to SGD; $\eta = 0.1$ for epochs 1–100, 0.01 for epochs 101–150, and 0.001 for epochs 151–200, so that the staircase effect appears in the learning curve in which it is noted that the horizontal axis for epoch iteration is in log-scale. These values are applied to all the algorithms throughout the experiments unless mentioned otherwise. The same values are used for the common hyper-parameters among all the algorithms.

**Effect of the number of parameter blocks** We initially design the experiment to validate the behavior of our algorithm as a function of the number of parameter blocks $M$. Thus, we compare our BCSC with varying $M = 2, 4, 8, 16$ against SGD ($M = 1$) using the models LeNet4, VGG19, and ResNet18, on the Cifar10 and Cifar100 datasets. In this experiment, we use a fixed learning rate of $\eta = 0.1$ across all the epochs to better understand the behavior of BCSC in comparison to SGD. The learning curves obtained from

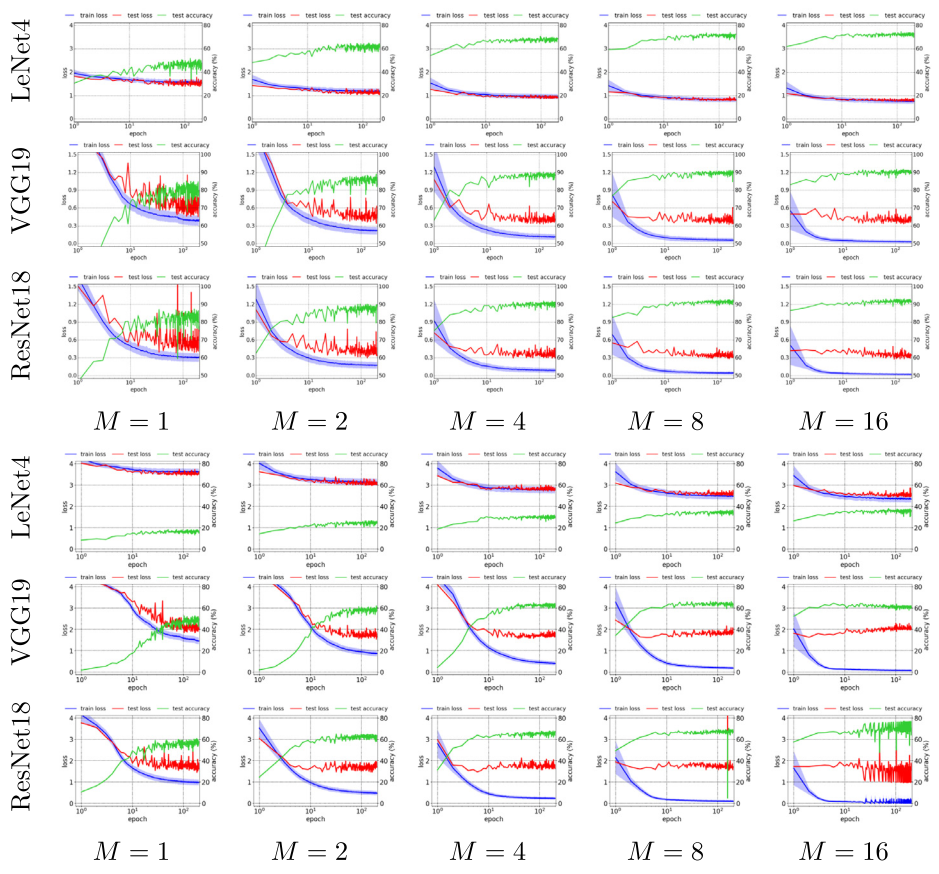

**Fig. 2. Effect of the number of parameter blocks** $M$. Learning curves obtained based on (top part) Cifar10 and (bottom part) Cifar100 datasets by our BCSC with varying number of parameter blocks $M$ for different network architectures. BCSC with $M = 1$ is equivalent to SGD. The train loss is indicated in blue, the test loss in red, and the test accuracy in green. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

**Table 1**
Test accuracy (%) obtained by BCSC with varying number of parameter blocks at different training outliers (%) based on Cifar10 using LeNet4 model. The learning rate is fixed at $\eta = 0.1$.

| Training outlier (%) | 0 | 5 | 10 | 15 |
|---|---|---|---|---|
| SGD ($M = 1$) | 57.91 | 53.43 | 52.80 | 51.79 |
| BCSC ($M = 2$) | 67.80 | 66.31 | 64.45 | 64.98 |
| BCSC ($M = 4$) | 71.72 | 71.32 | 70.73 | 70.12 |
| BCSC ($M = 8$) | **73.88** | **73.64** | **73.56** | **73.21** |

the models are presented with varying $M$ in Fig. 2 where it is clearly observed that both training and testing losses (red and blue lines) are significantly improved with increasing $M$, $M \leq 8$ in particular with deeper network models where the number of parameters is large, resulting in a notable improvement of the test accuracy (green line). The maximum of test accuracy and the minimum of test loss have reached a peak at $M = 8$ in some of the conditions. This observation implies that $M \geq 16$ may result in the under-fitting of the model due to the strong regularization effect. It is also observed that the variation of the training and testing losses decreases with increasing $M$ in most of the cases.

**Robustness to training outliers** To demonstrate the robustness of our BCSC to training outliers in comparison to SGD, we compute test accuracy with different number of parameter blocks $M = 2, 4, 8$ in the presence of arbitrarily corrupted training data with randomly assigned false labels at varying rates of outliers from 0% (original), 5%, 10%, 15% based on Cifar10 dataset using LeNet4 network model that is simple, yet illustrative. The training loss computed using a noise example can be extremely higher than the others. This means that the noise perturbs not only the step direction but also the step size of the model update. Our algorithm is designed to explicitly restrict the impact of such outliers. Table 1 presents the average test accuracy and clearly demonstrates that the test accuracy is improving with larger number of parameter partitions $M$ with the same amount of noise. Fig. 3 shows the learning curves obtained by SGD ($M = 1$) and our BCSC with ($M = 2, 4, 8$). SGD is shown to be more sensitive to outliers, whereas BCSC yields higher accuracy with increasing $M$ demonstrating its robustness against training outliers.

**Results with adaptive learning rate** In order to demonstrate that the benefits of BCSC are not diminished when using an adaptive learning rate, we compare BCSC with $M$ = 8 and SGD when integrated with the learning rate given by AdaGrad (Duchi et al., 2011) based on the basic models: VGG19 and ResNet18, using the Cifar10 dataset. The learning curves are presented in Fig. 4 where the training loss and the test loss are noticeably improved with BCSC in comparison to SGD. The results indicate that BCSC outperforms SGD consistently, regardless of whether the adaptive learning rate scheme is applied to the algorithm.

**Results with Dropout** In this experiment, we demonstrate that the regularization effects of BCSC persist if additional regularization is employed, for instance using Dropout. We employ a simple network model, LeNet4, in which we can easily observe the effect of Dropout using the MNIST dataset. Table 2 summarizes the average test accuracy of BCSC with parameter blocks $M = 2, 4, 8$ in comparison to SGD ($M = 1$) at different rates

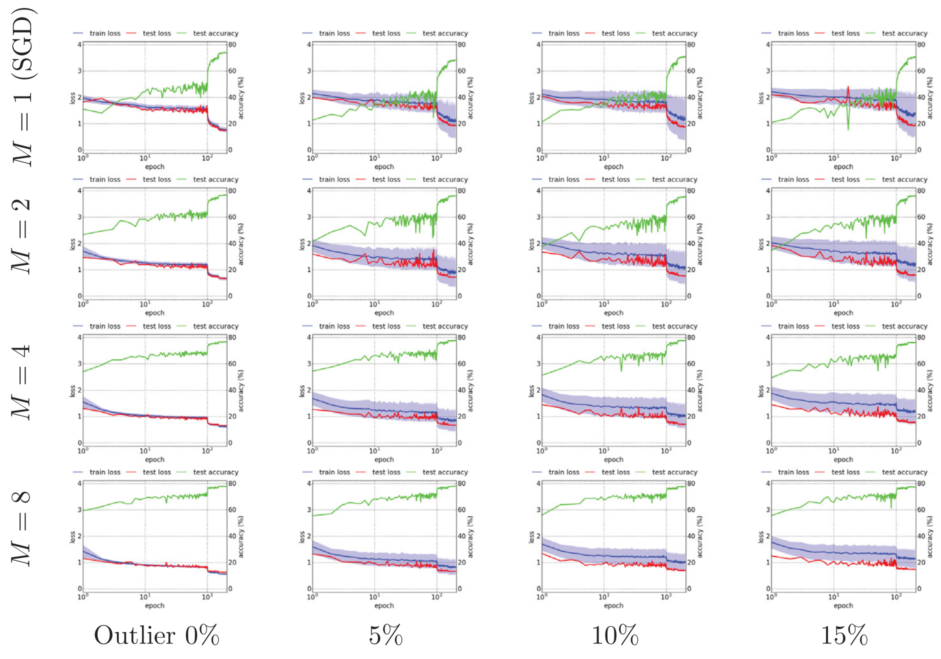

**Fig. 3. Robustness to training outliers**. Learning curves obtained based on Cifar10 by BCSC with varying number of parameter blocks $M$ at different degrees of training outliers (%) using LeNet4 network model. BCSC with $M = 1$ is equivalent to SGD.

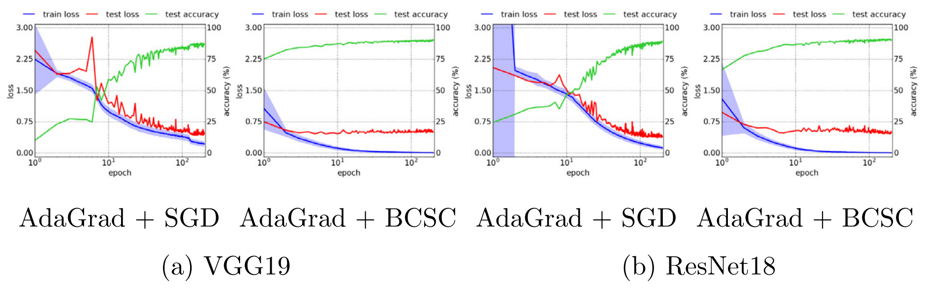

**Fig. 4. Comparison of BCSC with SGD when using with adaptive learning rate (AdaGrad)**. Learning curves obtained by SGD with AdaGrad and BCSC with AdaGrad using Cifar10. $M$ = 8 is used for BCSC.

**Table 2**
Test accuracy (%) obtained by BCSC with varying number of parameter blocks at different Dropout rates (%) based on MNIST dataset using LeNet4 model.

| Dropout rate (%) | 0 | 5 | 10 | 15 |
|---|---|---|---|---|
| SGD ($M = 1$) | 98.98 | 99.04 | 99.09 | 99.04 |
| BCSC ($M = 2$) | 99.00 | 99.10 | 99.14 | 99.13 |
| BCSC ($M = 4$) | **99.04** | 99.10 | 99.17 | 99.16 |
| BCSC ($M = 8$) | 99.02 | **99.13** | **99.17** | **99.19** |

of Dropout (0%, 5%, 10%, 15%). It is shown that BCSC outperforms SGD regardless of Dropout even though the effectiveness of a larger number of parameter blocks $M$ is shown to be weaker, which is due to the relatively small number of parameters in the network model.

**Results with deep models on Cifar10** We compare the performance of our BCSC against other state-of-the-art optimizations including stochastic gradient descent (SGD), stochastic randomized block-coordinate descent (SBC), and randomized block-coordinate descent (RBC). In this comparative analysis, we provide the learning curves and the test accuracy table based on the network models including LeNet4, VGG19, ResNet18, GoogLeNet and DPN92 using the Cifar10 dataset. The experimental results for BCSC are obtained with $M = 8$, which is chosen as an example, but the results with other values for $M$ agree with the effectiveness and robustness of the number of parameter blocks as demonstrated by the previous experiments. The learning curves obtained by different optimization algorithms, SGD, SBC, RBC and BCSC, based on different network models are presented in Fig. 5 where BCSC outperforms all the other algorithms in accuracy and stability regardless of the network models. Note that the effectiveness of our method is demonstrated in the learning curve even at the initial epoch where a number of iterations are performed across different mini-batches. In addition to the comparison by the learning curve, we provide quantitative evaluation of the test accuracy computed within (a) the first half epochs, (b) the last half epochs, (c) all the epochs and (d) the final epoch in Table 3 where the first half epochs means those in the range between 1st epoch and 100th epoch in our experimental condition for example. These experimental results indicate that our BCSC algorithm outperforms all the state-of-the-art optimization

**Table 3**
Test accuracy (%) obtained by SGD, SBC, RBC, and BCSC with $M = 8$ based on Cifar10 using different network models.

| | (a) First half epochs | | | | (b) Last half epochs | | | |
|---|---|---|---|---|---|---|---|---|
| | SGD | SBC | RBC | BCSC | SGD | SBC | RBC | BCSC |
| LeNet4 | 46.98 | 64.32 | 54.34 | **70.49** | 70.33 | 72.90 | 72.74 | **77.17** |
| VGG19 | 75.28 | 85.40 | 80.09 | **89.22** | 92.33 | 92.58 | 92.57 | **93.70** |
| ResNet18 | 79.43 | 87.01 | 81.34 | **90.64** | 93.89 | 93.76 | 93.74 | **95.12** |
| GoogLeNet | 77.66 | 86.40 | 80.48 | **89.97** | 94.04 | 94.01 | 94.04 | **95.56** |
| DPN92 | 80.53 | 86.62 | 70.78 | **91.15** | 94.50 | 93.99 | 89.58 | **95.24** |
| | (c) All epochs | | | | (d) Final epoch | | | |
| | SGD | SBC | RBC | BCSC | SGD | SBC | RBC | BCSC |
| LeNet4 | 58.66 | 68.61 | 63.54 | **73.83** | 73.24 | 73.80 | 75.25 | **77.61** |
| VGG19 | 83.81 | 88.99 | 86.33 | **91.46** | 93.62 | 92.69 | 93.58 | **94.09** |
| ResNet18 | 86.66 | 90.38 | 87.54 | **92.88** | 94.90 | 93.87 | 94.34 | **95.19** |
| GoogLeNet | 85.85 | 90.21 | 87.26 | **92.77** | 94.78 | 94.02 | 94.36 | **95.61** |
| DPN92 | 87.51 | 90.30 | 80.18 | **93.20** | 95.38 | 94.20 | 91.70 | **95.46** |

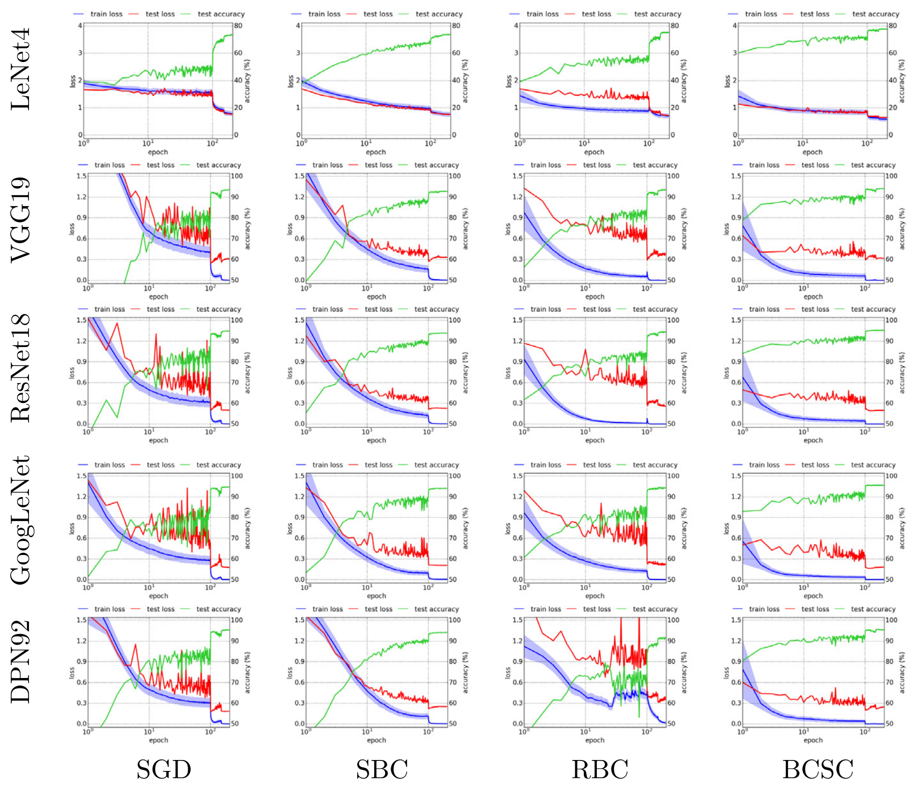

**Fig. 5. Evaluation on Cifar10**. Learning curves obtained by (SGD) stochastic gradient descent, (SBC) stochastic randomized block-coordinate descent, (RBC) randomized block-coordinate descent, and (BCSC) our algorithm with $M = 8$.

methods irrespective of the architecture and the depth of the models by the final accuracy.

**Results with deep models on Cifar100** We now further validate the performance of our BCSC in comparison with the state-of-the-art optimizations, SGD, SBC, and RBC using the more challenging Cifar100 based on the deep models including MobileNet, ShuffleNet, VGG19, ResNet18, SENet18, DenseConv, ResNeXt29, GoogLeNet, and DPN92. In this experiment, we use $M = 2$ due to the heavy computational cost required to optimize deep models using the Cifar100 dataset. The learning curves are obtained by SGD, SBC, RBC and BCSC with $M = 2$, based on different network models, and they are presented in Fig. 6 where better and more stable results are observed with BCSC irrespective of the architecture albeit the minimum partition number $M = 2$ is used. It is clearly observed that the train loss computed by BCSC decreases faster than all the other algorithms, which leads to significant improvement in particular at early epochs. The quantitative evaluation of BCSC with $M = 2$ in comparison to the other optimization methods is provided in Table 4 where the test accuracy is computed at (a) first half epochs, (b) last half epochs, (c) all epochs, and (d) final epoch. The test accuracy with BCSC is shown to be better than the one with all the other methods under comparison at the final epoch and over the all epochs. These experiments further confirm that BCSC outperforms the state-of-the-arts optimizations irrespective of the network architectures in accuracy and stability. The effectiveness of our BCSC algorithm can be naturally demonstrated with larger number of partitions in the model parameters.

## 6. Discussion

We have presented a first-order optimization algorithm for large scale problems in the deep learning framework when both

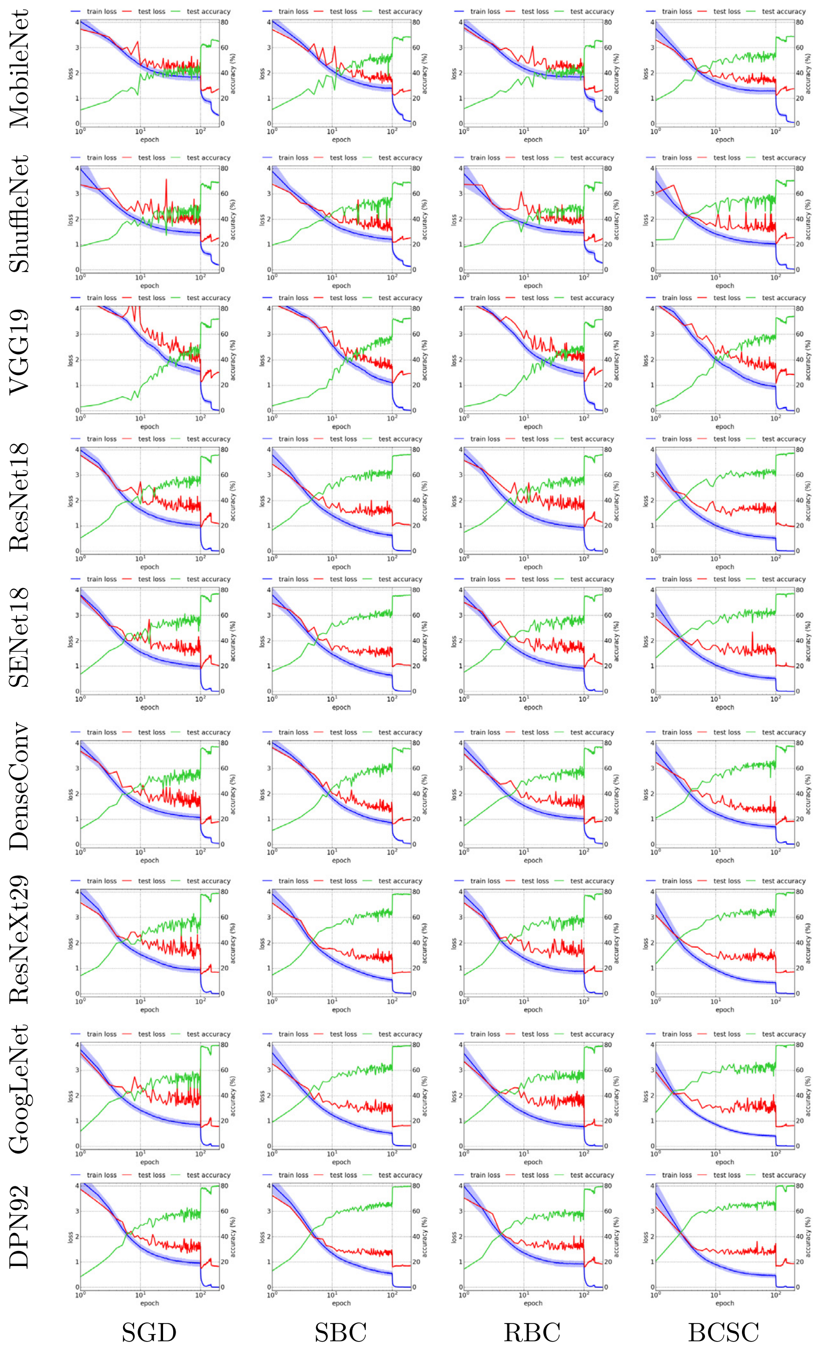

**Fig. 6. Evaluation on Cifar100**. Learning curves obtained by SGD, SBC, RBC, and our BCSC with $M = 2$ for different network architectures.

the number of training data and the number of model parameters are large, and when the training data is polluted with outliers. The proposed algorithm, named BCSC, is based on the intuition that different subsets of data being used for updating different subsets of parameters is beneficial in achieving better generalization and handling outliers. The experimental results based on the state-of-the-art network models with the standard image classification datasets indicate that the proposed doubly stochastic process with the block-cyclic constraint leads to improved model generalization and robustness to outliers in the training phase. In addition, it has been empirically demonstrated that our algorithm outperforms the state-of-the-arts in optimizing a number of recent deep models in terms of accuracy and stability of the learning curve over the update-iteration. The implementation issue of our algorithm is that we use $M$ of the individual mini-batch sequences that increase the computational load due to the back-propagation. We have demonstrated that BCSC has outperformed RBC that updates the parameters in discrete time and requires the same computational load. Moreover our algorithm could overtake SGD within the same number of parameter updates. Our algorithm can be naturally extended to distributed and parallel computation, so as to mitigate the added computational cost due to the doubly stochastic process. Additional variants to the sampling and circulant schemes, as well as hyper-parameter tuning and determination of the optimal parameter-batch sizes, are also subject of future work.

**Table 4**
Test accuracy (%) obtained based on Cifar100 dataset by SGD, SBC, RBC, and our BCSC with $M = 2$ at different ranges of epochs.

| | (a) First half epochs | | | | (b) Last half epochs | | | |
|---|---|---|---|---|---|---|---|---|
| | SGD | SBC | RBC | BCSC | SGD | SBC | RBC | BCSC |
| MobileNet | 39.36 | 47.24 | 39.09 | **51.47** | 63.80 | 67.89 | 63.24 | **67.79** |
| ShuffleNet | 43.57 | 50.12 | 45.26 | **53.75** | 67.77 | 68.53 | 67.22 | **69.57** |
| VGG19 | 38.47 | 47.37 | 41.42 | **51.48** | 69.48 | 71.53 | 69.46 | **72.38** |
| ResNet18 | 52.14 | 58.32 | 52.56 | **60.21** | 74.35 | 75.90 | 74.71 | **76.79** |
| SENet18 | 52.90 | 57.74 | 53.40 | **60.09** | 75.38 | 75.83 | 75.11 | **76.98** |
| DenseConv | 51.91 | 55.15 | 53.86 | **60.02** | 75.68 | 75.36 | 75.52 | **76.84** |
| ResNeXt29 | 52.65 | 59.78 | 54.79 | **62.39** | 77.52 | 78.33 | 77.42 | **78.97** |
| GoogLeNet | 51.14 | 58.86 | 53.83 | **60.97** | 78.33 | 79.17 | 78.20 | **79.68** |
| DPN92 | 54.58 | 61.65 | 55.92 | **63.88** | 78.30 | **79.50** | 77.97 | 79.48 |
| | (c) All epochs | | | | (d) Final epoch | | | |
| | SGD | SBC | RBC | BCSC | SGD | SBC | RBC | BCSC |
| MobileNet | 51.58 | 57.57 | 51.17 | **59.63** | 65.21 | 68.57 | 65.04 | **68.88** |
| ShuffleNet | 55.67 | 59.32 | 56.24 | **61.66** | 69.12 | 68.77 | 68.56 | **70.36** |
| VGG19 | 53.98 | 59.45 | 55.44 | **61.93** | 72.14 | 72.62 | 71.82 | **74.19** |
| ResNet18 | 63.24 | 67.11 | 63.64 | **68.50** | 76.08 | 76.32 | 76.23 | **77.28** |
| SENet18 | 64.14 | 66.78 | 64.26 | **68.53** | 77.28 | 76.11 | 76.50 | **77.30** |
| DenseConv | 63.79 | 65.25 | 64.69 | **68.43** | 77.22 | 75.53 | 76.91 | **77.46** |
| ResNeXt29 | 65.09 | 69.05 | 66.11 | **70.68** | 78.88 | 78.21 | 78.47 | **79.37** |
| GoogLeNet | 64.73 | 69.02 | 66.01 | **70.33** | 79.51 | 79.42 | 79.04 | **80.20** |
| DPN92 | 66.44 | 70.57 | 66.95 | **71.68** | 79.98 | 79.77 | 79.45 | **80.23** |

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF), Republic of Korea: 2017R1A2B4006023 and 2019K1A3A1A77074958, and the Office of Naval Research (ONR), USA: N00014-19-1-2229.

## References

An, G. (1996). The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation*, *8*(3), 643–674.

Balduzzi, D., Frean, M., Leary, L., Lewis, J., Ma, K.-D., & McWilliams, B. (2017). The shattered gradients problem: If resnets are the answer, then what is the question? In *Proceedings of machine learning research*: *Vol. 70*, *Proceedings of the 34th international conference on machine learning* (pp. 342–350). PMLR.

Chen, Y., Li, J., Xiao, H., Jin, X., Yan, S., & Feng, J. (2017). Dual path networks. In *Advances in neural information processing systems* (pp. 4470–4478).

De, S., Yadav, A., Jacobs, D., & Goldstein, T. (2016). Big batch SGD: Automated inference using adaptive batch sizes. arXiv preprint arXiv:1610.05792.

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

Duchi, J., & Singer, Y. (2009). Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, *10*(Dec), 2899–2934.

George, A. P., & Powell, W. B. (2006). Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine Learning*, *65*(1), 167–198.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. (2013). Maxout networks. In *ICML'13*, *Proceedings of the 30th international conference on international conference on machine learning* (pp. III–1319–III–1327). JMLR.org, http://dl.acm.org/citation.cfm?id=3042817.3043084.

He, K., Zhang, X., Ren, S., & Sun, J. (2016a). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

He, K., Zhang, X., Ren, S., & Sun, J. (2016b). Identity mappings in deep residual networks. In *European conference on computer vision* (pp. 630–645). Springer.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

Hoffer, E., Hubara, I., & Soudry, D. (2017). Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in neural information processing systems* (pp. 1729–1739).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., et al. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

Hu, J., Shen, L., & Sun, G. (2017). Squeeze-and-excitation networks. arXiv preprint arXiv:1709.01507.

Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Huang, G., Sun, Y., Liu, Z., Sedra, D., & Weinberger, K. Q. (2016). Deep networks with stochastic depth. In *European conference on computer vision* (pp. 646–661). Springer.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448–456).

Johnson, R., & Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems* (pp. 315–323).

Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images*: *Technical report*, University of Toronto.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Lang, K. J., & Hinton, G. E. (1990). Dimensionality reduction and prior knowledge in e-set recognition. In *Advances in neural information processing systems* (pp. 178–185).

Lau, T. T.-K., Zeng, J., Wu, B., & Yao, Y. (2018). A proximal block coordinate descent algorithm for deep neural network training. arXiv preprint arXiv:1803.09082.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lee, C.-P., & Wright, S. J. (2016). Random permutations fix a worst case for cyclic coordinate descent. arXiv preprint arXiv:1607.08320.

Leventhal, D., & Lewis, A. S. (2010). Randomized methods for linear constraints: convergence rates and conditioning. *Mathematics of Operations Research*, *35*(3), 641–654.

Liu, X., Yan, J., Wang, X., & Zha, H. (2016). Parallel randomized block coordinate descent for neural probabilistic language model with high-dimensional output targets. In *Chinese conference on pattern recognition* (pp. 334–348). Springer.

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3431–3440).

Nesterov, Y. (2012). Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, *22*(2), 341–362.

Plaut, D. C., et al. (1986). Experiments on learning by back propagation. *ERIC*.

Richtárik, P., & Takáč, M. (2014). Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, *144*(1–2), 1–38.

Richtárik, P., & Takáč, M. (2016). Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, *156*(1–2), 433–484.

Rifai, S., Glorot, X., Bengio, Y., & Vincent, P. (2011). Adding noise to the input of a model trained with a regularized objective. arXiv preprint arXiv:1104.3250.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 400–407.

Roux, N. L., Schmidt, M., & Bach, F. (2012). A stochastic gradient method with an exponential convergence rate for finite training sets. In *NIPS'12, Proceedings of the 25th international conference on neural information processing systems (Vol. 2)* (pp. 2663–2671).

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive Modeling*, *5*(3), 1.

Saha, A., & Tewari, A. (2010). On the finite time convergence of cyclic coordinate descent methods. arXiv preprint arXiv:1005.2146.

Saha, A., & Tewari, A. (2013). On the nonasymptotic convergence of cyclic coordinate descent methods. *SIAM Journal on Optimization*, *23*(1), 576–601. http://dx.doi.org/10.1137/110840054.

Schaul, T., Zhang, S., & LeCun, Y. (2013). No more pesky learning rates. In *International conference on machine learning* (pp. 343–351).

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., et al. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1–9).

Tan, C., Ma, S., Dai, Y.-H., & Qian, Y. (2016). Barzilai-borwein step size for stochastic gradient descent. In *Advances in neural information processing systems* (pp. 685–693).

Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., & Goldstein, T. (2016). Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning* (pp. 2722–2731).

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, 267–288.

Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., & Fergus, R. (2013). Regularization of neural networks using dropconnect. In *Proceedings of the 30th International conference on machine learning (ICML-13)* (pp. 1058–1066).

Wang, H., & Banerjee, A. (2014). Randomized block coordinate descent for online and stochastic optimization. arXiv preprint arXiv:1407.0107.

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2016). Aggregated residual transformations for deep neural networks. arXiv preprint arXiv:1611.05431.

Yuan, M., & Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, *68*(1), 49–67.

Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

Zeng, J., Lau, T. T.-K., Lin, S., & Yao, Y. (2019). Global convergence of block coordinate descent in deep learning. In *International conference on machine learning* (pp. 7313–7323). PMLR.

Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on machine learning* (p. 116). ACM.

Zhang, Z., & Brand, M. (2017). Convergent block coordinate descent for training tikhonov regularized deep neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Vol. 30*, *Advances in neural information processing systems* (pp. 1721–1730). Curran Associates, Inc., https://proceedings.neurips.cc/paper/2017/file/6a2feef8ed6a9fe76d6b3f30f02150b4-Paper.pdf.

Zhang, S., Choromanska, A. E., & LeCun, Y. (2015). Deep learning with elastic averaging SGD. In *Advances in neural information processing systems* (pp. 685–693).

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2017). Shufflenet: An extremely efficient convolutional neural network for mobile devices. arXiv preprint arXiv:1707.01083.

Zhao, T., Yu, M., Wang, Y., Arora, R., & Liu, H. (2014). Accelerated mini-batch randomized block coordinate descent method. In *NIPS'14, Proceedings of the 27th international conference on neural information processing systems (Vol. 2)* (pp. 3329–3337).

Zhu, Z., Wu, J., Yu, B., Wu, L., & Ma, J. (2019). The anisotropic noise in stochastic gradient descent: Its behavior of escaping from sharp minima and regularization effects. In *ICML* (pp. 7654–7663).