TU Delft

Delft University of Technology

SlimML: Removing Non- critical Input Data in Large-scale Iterative Machine Learning

Han, Rui; Liu, Chi Harold; Li, Shilin; Chen, Lydia; Wang, Guoren; Tang, Jian; Ye, Jieping

DOI
10.1109/TKDE.2019.2951388

Publication date
2019

Document Version
Accepted author manuscript

Published in
IEEE Transactions on Knowledge and Data Engineering

Citation (APA)
Han, R., Liu, C. H., Li, S., Chen, L., Wang, G., Tang, J., & Ye, J. (2019). SlimML: Removing Non- critical Input Data in Large-scale Iterative Machine Learning. *IEEE Transactions on Knowledge and Data Engineering*, *33*(5), 2223-2236. Article 8890886. Advance online publication. https://doi.org/10.1109/TKDE.2019.2951388

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2951388, IEEE Transactions on Knowledge and Data Engineering

SlimML: Removing Non-critical Input Data in Large-scale Iterative Machine Learning

Rui Han, Chi Harold Liu, *Senior Member, IEEE*, Shilin Li, Lydia Y. Chen, *Senior Member, IEEE*, Guoren Wang, *Senior Member, IEEE*, Jian Tang, *Fellow, IEEE*, Jieping Ye

Abstract—The core of many large-scale machine learning (ML) applications, such as neural networks (NN), support vector machine (SVM), and convolutional neural network (CNN), is the training algorithm that iteratively updates model parameters by processing massive datasets. From a plethora of studies aiming at accelerating ML, being data parallelization and parameter server, the prevalent assumption is that all data points are equivalently relevant to model parameter updating. In this paper, we challenge this assumption by proposing a criterion to measure a data point's effect on model parameter updating, and experimentally demonstrate that the majority of data points are non-critical in the training process. We develop a slim learning framework, termed SlimML, which trains the ML models only on the critical data and thus significantly improves training performance. To such an end, SlimML efficiently leverages a small number of aggregated data points per iteration to approximate the criticalness of original input data instances. The proposed approach can be used by changing a few lines of code in a standard stochastic gradient descent (SGD) procedure, and we demonstrate experimentally, on NN regression, SVM classification, and CNN training, that for large datasets, it accelerates model training process by an average of 3.61 times while only incurring accuracy losses of 0.37%.

Index Terms—Iterative machine learning, large input datasets, model parameter updating, MapReduce.

1 Introduction

Machine learning (ML) has become ubiquitous in recent years and its success can be attributed to its ability to extract knowledge and make decisions by learning the underlying structures of large input datasets [17], [27], [36]. To train learning models, ML applications often adopt the iterative optimization process [12]. It specifically minimizes the cost (loss) function that quantifies the penalty paid for the difference between estimated and actual values for input data. The trained model is characterized by its *parameters* and evaluated by an *accuracy* metric such as prediction error or classification accuracy on test sets. In many real-life applications, the training algorithm has to process a tremendous number of input data instances and takes a significantly long time, tending to be the bottleneck of ML. The outstanding challenge still remains of how to efficiently use ML systems on massive input data points [17], [28] and commodity hardware [24], [44].

To enable fast processing of large datasets, one major category of techniques exploits data parallelism to improve the performance of data processing at each iteration [22], [40], [54], or samples important data to improve the convergence speed of iterative training (optimization) process [38], [45]. In a data-parallel environment, large datasets also result in tremendous amount of local variables (e.g. intermediate results to compute gradients) during model training [43], in particular for deep neural networks (e.g. convolutional neural networks (CNNs) [41]) with millions of model parameters. Another category of techniques, therefore, is developed to improve the performance of accessing (storing and retrieving) and synchronizing local variables and global parameters [20], [31], [32], [43], [53], or apply compression methods (e.g. vector quantization [33] and weight pruning [29]) to reduce the model size. At each iteration of model training, these techniques typically use stochastic optimization methods to select a subset/mini-batch of input data for model training [14]. In particular, latest batch size control techniques address large mini-batch instability and significantly increase training speed by using larger batch sizes [23], [52]. Most of these techniques treat the *selected data points in each batch equally* and implicitly assume the *equal effect* on model parameter updating.

However, when applying ML in real datasets, there exists a considerable proportion of **non-critical input data**, namely the data that has *little influence on model parameter updating* during the iterative training process. For example, Fig. 1 illustrates critical and non-critical data points in three typical iterative ML algorithms: neural network (NN) regression [56], SVM classifier [19], and CNN [41], [42]. We can see that *critical* points are those whose gradients are larger than 0 (NN regression), are incorrectly classified (inside the two hyperplanes of the SVM classifier), and have the largest error signal (CNN), respectively. The remaining points are *non-critical* because they do not influence model parameter updating. Our empirical evaluations on massive and real datasets (Section 2.3) show that the percentages of non-critical input data range from 48.77% to 93.59% (the average is 75.27%) across these three ML algorithms and across their iterations of training. Based on the observation on the presence of non-critical data, coresets [13], [34] and importance sampling [48], [49], [55], [61] techniques select part of important input data points for model training. However, the former uses fixed subsets of input data and cannot reflect the changes in model parameters, and the

- *Rui Han, Chi Harold Liu, Shilin Li, Guoren Wang are with the Beijing Institute of Technology, P.R. China. Lydia Y. Chen is with the TU Delft, Netherlands. Jian Tang and Jieping Ye are with DiDi AI Labs, P.R. China.*

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2951388, IEEE Transactions on Knowledge and Data Engineering

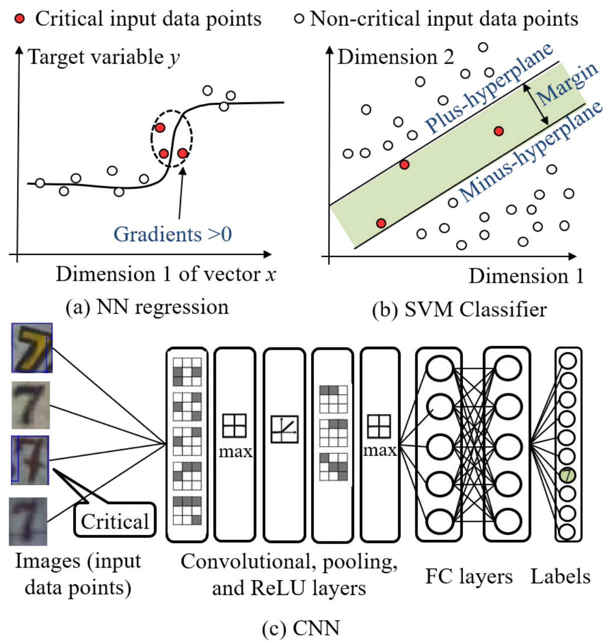


Fig. 1. Examples of critical and non-critical data in three ML algorithms

later is computationally expensive to compute the gradient norms of all input data points for sampling.

Motivated by these problems, this paper proposes S-limML, a non-critical input data removal framework for iterative and large-scale ML. The basic approach taken by SlimML is to generate a small number of **aggregated** data points at the pre-training stage, where each point preserves the averaged attribute values over a subset of similar **original** input data points. At each iteration of the model training stage, SlimML uses aggregated data points to estimate the effects between different parts of the input data and model parameter updating, thus removing the non-critical parts before training. In particular, the contributions of this paper are as follows:

- We propose an effective criterion to quantify a data point's effect on *model parameter updating*. The experimental evaluations on widely-used ML algorithms and datasets show that this criterion distinguishes critical and non-critical input data during iterative model training (Section 2).
- We design two SlimML modules for efficient non-critical input data removal (Section 3). First, the *input data aggregator* module quickly generates aggregated data points for high-dimensional and massive datasets using incremental singular value decomposition (SVD) [26] based dimensionality reduction and locality sensitive data division. Second, the *input data remover* module enables fast estimation of input data's effect using aggregated data points, while providing precise estimation by selecting and processing a small number of aggregated data points that possibly contain critical data.
- We implemented SlimML on Spark [1] (a mainstream platform to support large-scale ML using the MapReduce paradigm) and incorporated it with standard and deep ML algorithms using Intel BigDL [2]: NN regression, SVM, and CNN (Section 3.4). Specifically, we apply S-limML on the prevalent training methods (mini-batch gradient descend and stochastic gradient descent (SGD)) of these algorithms, and also demonstrate the general applicability of of SlimML to other optimization methods, i.e. Adam [39] and Adadelta [60], and in combination with the importance sampling [11], [45].
- We evaluate the effectiveness of SlimML on large datasets of millions of instances (Section 4), some of which requires training time upto two days for SVM on commodity hardware. The evaluation results show: (1) in SlimML, the times of generating and processing aggregated data points take an average of 0.16% and 2.62% of the total model training time; (2) by removing non-critical input data during the training process, SlimML reduces input data by an average of 71.31% and speedups model training by an average of 3.61 times for all models and by 4.63 times for the large AlexNet model [41], while only incurring negligible accuracy losses of 0.37% and 0.10%.

2 Background and Motivation

To show our motivation behind focuing on removing non-critical input data during the iterative training process, we demonstrate its prevalence in three popular iterative ML algorithms: NN regression [56], SVM classifier [19], and CNN [41]. This section first explains their model updating process as the background of this work (Section 2.1). Subsequently, it formally defines a data point's effect on model parameter updating and non-critical data (Section 2.2) and presents the results of a measurement study on non-critical input data using concrete cases and real datasets (Section 2.3).

2.1 Model Parameter Updating in Iterative ML Algorithms

This work studies the gradient descent based supervised learning algorithms (e.g. regression or classification), which represent the dominant iterative optimization techniques in the ML community [14], [47], [50]. Given a model (function $f(\vec{x}, \Theta)$) characterized by its parameters Θ, an algorithm trains it to minimize a specified cost function $c(\Theta, I)$ (also called loss or error function) on a training dataset I. Such a cost function essentially represents a surrogate of error on unknown samples, namely the *model accuracy* on test datasets such as the prediction error in regression problems or the accuracy of identifying correct categories in classification problems.

In a typical training process, the algorithm starts from randomly generated initial parameters Θ_0 of the model and iteratively updates them until convergence. At iteration i ($i > 1$), the algorithm takes a set I_i of data points and the parameters Θ_{i-1} from the previous iteration as input and updates each parameter $\theta_i \in \Theta_i$ as follows:

$$\theta_i = \theta_{i-1} - \lambda \cdot g(I_i, \Theta_{i-1}, \theta_{i-1}) \tag{1}$$

where λ is the learning rate and gradient $g(I_i, \Theta_{i-1}, \theta_{i-1})$ represents the steepest direction of updating θ_{i-1} in order to have the largest reduction in the cost function. This gradient is calculated as:

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2951388, IEEE Transactions on Knowledge and Data Engineering

$$g(I_i, \Theta_{i-1}, \theta_{i-1}) = \frac{1}{|I_i|} \sum_{j=1}^{|I_i|} \left(\frac{\partial c(\Theta_{i-1}, \vec{x}^{(j)})}{\partial \theta_{i-1}} \cdot (f(\vec{x}^{(j)}, \Theta_{i-1}) - y^{(j)}) \right) \quad (2)$$

where $\frac{\partial c(\Theta_{i-1}, \vec{x}^{(j)})}{\partial \theta_{i-1}}$ is the partial derivative of the cost function on parameter θ_{i-1} and $(f(\vec{x}^{(j)}, \Theta_{i-1} - y^{(j)}))$ is the prediction error on data point ($\vec{x}^{(j)}$,$y^{(j)}$). We now introduce three examples following the above iterative training process.

NN regression. The first example represents one important class of regression applications. The algorithm trains a neural network model to predict $\vec{x}$'s target value y [56]. The *cost function* (i.e. mean squared error (MSE)) measures prediction errors on I: $c(\Theta, I) = \frac{1}{N}\sum_{i=1}^{N}(f(\vec{x}^{(i)}, \Theta) - y^{(i)})^2 + \lambda\frac{1}{v}\sum_{i=1}^{v} w_i^2$, where v is the number of weights in Θ and λ is the regularization parameter.

Model parameter updating. Mini-batch gradient descent [14] is a dominant training method for neural network models. The algorithm has several epochs, each epoch first shuffles the input data and then sequentially divides it into multiple subsets. At each iteration, the algorithm uses a set of input data points to update model weights.

SVM classifier. The second example constitutes another important application in supervised ML, and we study the binary soft-margin SVM classifier [19]. This classifier constructs a plus-hyperplane (y=+1) and a minus-hyperplane (y=-1), classifying data points into positive and negative categories. Given input data I, the *cost function* measures the error of the classifier on the training data: $c(\Theta, I) = \frac{1}{2}\|\vec{w}\|^2 + \lambda\frac{1}{N}\sum_{i=1}^{N}\max\{0, 1 - y^{(i)}f(\vec{x}^{(i)}, \Theta)\}$, where $\vec{w}$ is a d-dimensional vector of weights.

Model parameter updating. *SGD* is a widely applied method to train SVM classifiers [51]. At each iteration, this algorithm randomly takes one or multiple input data points and uses them to calculate the *gradient* for model parameter updating.

CNN. The final example is one major type of deep learning applications. CNNs [41] are widely used in visual recognition and take images as input data. A CNN architecture (the whole network model such as LeNet-5 [42] and AlexNet [41]) uses a series of layers of different types (that is, convolutional, pooling, Rectified Linear Unit (ReLU), and fully-connected (FC) layers) to extract an image's characteristics and predicts its class score in the last CF layer (the output layer). The pooling/ReLU layers implement fixed functions, and each convolutional/FC layer consists of multiple neurons. Each neuron has trainable parameters (weights and bias) and an activation function that defines the output, given an input combined with the parameters.

Model parameter updating. Similar to regular multi-layer neural networks, the parameters in a CNN architecture are trained using gradient descent and backpropagation. For every neuron in the convolutional/FC layer, its parameters are updated according to Eqn. (1) and (2), in which the error signal $(f(\vec{x}^{(j)}, \Theta_{i-1}) - y^{(j)})$ is backpropagated from the output layer to the input layer. Mini-batch gradient descent is a common training method for CNNs.

2.2 Effect on Model Parameter Updating

Herein, we focus on critical and non-critical data over the iterations of model training. At each iteration, a training algorithm starts from the model parameters obtained in the previous iteration and adjusts them according to gradients calculated using the input data. Hence, if processing a data point $(\vec{x}, y)$ triggers an update in the model parameters, we consider that point **critical** and introduce a criterion to measure the effect of processing it on model parameter updating.

Definition 1 (Effect on model parameter updating). *An input data point* $(\vec{x}, y)$*'s effect on updating the model parameters* Θ*, denoted by* $e(\vec{x}, y, \Theta)$*, is the sum of the absolute value of every parameter's gradient calculated using this data point:*

$$e(\vec{x}, y, \Theta) = \sum_{\theta \in \Theta} |g((\vec{x}, y), \Theta, \theta)| \quad (3)$$

According to Eqn. (2), the calculation of parameter θ's gradient depends on both the data point $(\vec{x}, y)$ and the parameters Θ obtained from the previous iteration. Note that in CNN, we only consider the parameters in *the first layer* (the input layer) when estimating an input data point's effect, because only this layer takes this point as input.

Discussion of input data selection/sampling. The identification and removal of non-critical input data (with small effects on model parameter updating) is not intended to replace, but rather complement the existing input data selection/sampling techniques. That is, only the input data points that are **used** by the training algorithm at each iteration are considered. For example, the SGD based training technique uses a mini-batch of sampled input data points (random or importance sampling [38], [61]) at every iteration, our work focuses on removing non-critical ones from these points.

Discussion of overfitting. Existing supervised learning algorithms usually employ regularization parameters to prevent overfitting. For example, the NN regression and the SVM classifier use parameters λ in their cost functions to avoid overfitting. The estimation of effects in Eqn. (3) is based on the cost functions with regularization parameters, thereby keeping the regularized objective of the training algorithms.

Discussion of other iterative optimization techniques. The definition of effect on parameter updating (Eqn. (3) is based on the gradient-based optimization techniques. This definition can be directly extended to other iterative optimization techniques (e.g. Newton and BFGS) by substituting the gradient in Eqn. (2).

2.3 Experimental Evaluation

The measurement in this section focuses on the *iterative* aspect of model training. We first take CNN (AlexNet architecture [41]) as an example and train this model using the mini-batch method on the 32×32×3 Cifar10 dataset [3] with 60k data points and 10 object classes. The whole training process takes 10k iterations. Figure 2 illustrates the probability distributions of input data points' effects on parameter updating at five different iterations. We can observe that: (i) in all iterations, the input data points have a long tail in the distribution such that most of the data points have much

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2951388, IEEE Transactions on Knowledge and Data Engineering

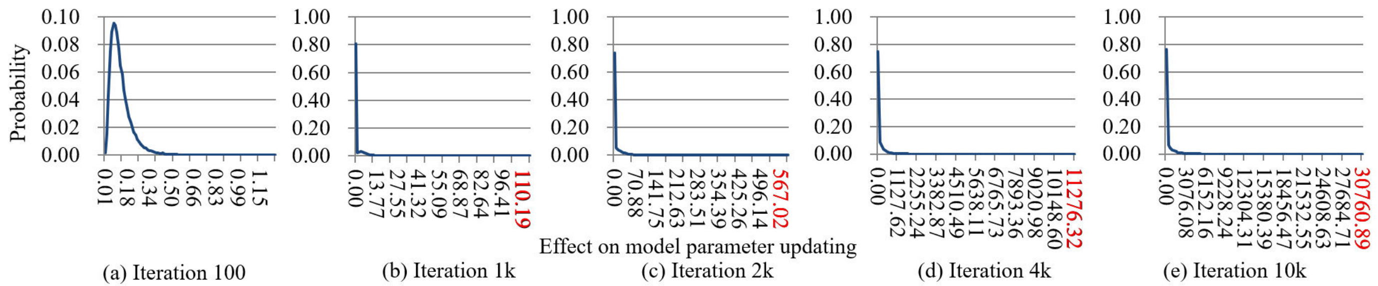


Fig. 2. Probability distributions of effect on parameter updating in CNN

smaller effect values than a small percentage of data points with large effects; (ii) the data points' effect values increase over the course of iterations. For example, the maximum effect value at iteration 10k is 300 times larger than that at iteration 1k.

Based on the observation of long tail values of effect, we set a *threshold* to divide input data points into critical and non-critical ones (Definition 2). We further set the threshold ϵ to 0.01 (that is, the data points whose cumulative effects are smaller than 1% of the total effects have negligible influence on model parameter updating) and extended our evaluation to the three ML algorithms in Section 2.1: the NN regression is trained using the NASA Earth Exchange (NEX) [7] dataset with 12.11 million points; the SVM classifier is trained using the Gas sensor array (GSA) [4] dataset with 8.39 million points; and the CNN (AlexNet architecture [41]) is trained using the Cifar10 dataset [3].

Definition 2 (Non-critical data points). *Given a set of m input data points $\{(\vec{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$ with increasing values of effect, the first n ones belong to non-critical input data if their cumulative effects divided by the total effects of all points is smaller than a threshold ϵ:*

$$\sum_{i=1}^{n} e(\vec{x}^{(i)}, y^{(i)}, \Theta) / \sum_{i=1}^{m} e(\vec{x}^{(i)}, y^{(i)}, \Theta) \leq \epsilon \tag{4}$$

Fig. 3(a), (c) and (e) show model training times (x axis) and the accuracies on test sets (y axis) in NN regression (number of neurons in hidden layer is 50), SVM classifier (Gaussian radial basis function (RBF) kernel is used and γ is 1), and CNN (AlexNet), respectively. We can observe that it takes several hours to dozens of hours to complete the training process, during which the model accuracy constantly improves across the iterations. During this iterative training process, Fig. 3(b), (d) and (f) show that *considerable proportions* of non-critical input data exist at each iteration: the average percentages of non-critical input data are 55.57%, 86.71%, and 81.70% in the NN regression, SVM, and CNN, respectively. In addition, we can observe that these proportions *gradually increase* over the iterations in all three algorithms. This indicates that the discrepancy of effects among data points becomes larger when more iterations are conducted. In other words, more data points have *negligible* impact on model parameter updating when the training process is nearing to converge. For example, in the SVM classifier, more data points fall between the two hyperplanes and thus do not influence the updating of model parameters.

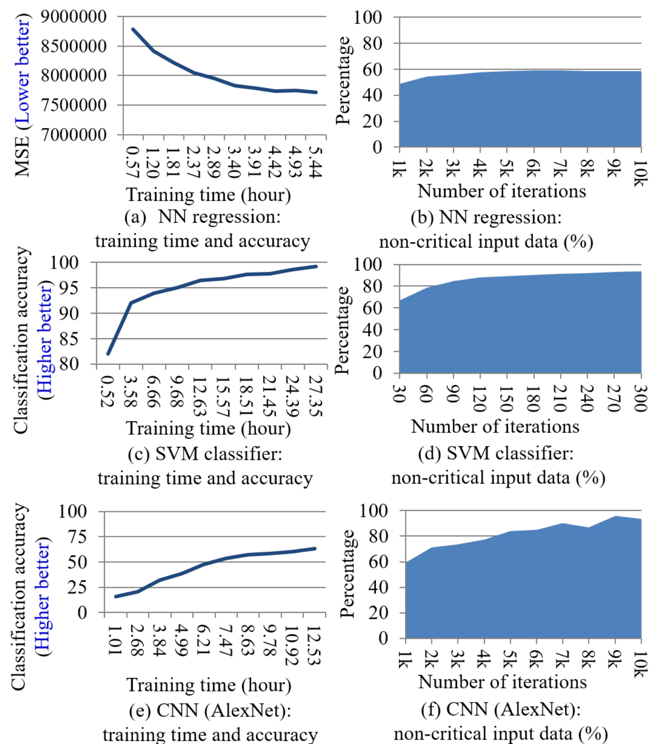


Fig. 3. Quantitative measurement of NN regression, SVM, and CNN

Challenge. The common observation from those three ML algorithm is that removing large amounts of non-critical input data can significantly reduce model training time (both in terms of computation and communication times), while causing negligible impact on model parameter updating and accuracy if the removal is done correctly. To slim down the training process by using a smaller amount of critical data, it is critical to estimate input data's effect in a *much shorter time* compared to the model training time. Nonetheless, directly computing all data points' effects takes prohibitively long time. Hence, the benefit and challenge motivate the design of SlimML, explained in the subsequent section.

3 SlimML

In this section, we first present an overview of SlimML in Section 3.1 and then explain its modules in Sections 3.2 and 3.3.

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.

3.1 Overview

For a ML application, SlimML is built upon the parallel data processing framework and enables the efficient removal of non-critical input data in model training using two modules, as shown in Fig. 4.

Input data aggregator. Given an input dataset, this module transforms it into multiple *aggregated data points*, each of which represents a part (subset) of the original input data points with similar feature values, and aggregates their attribute information. Note that the aggregated data generation is only performed once before the iterative model training stage.

Input data remover. Based on the aggregated data points, this model is designed to remove non-critical input data during the iterative training process. At each iteration, the module is applied in each parallel task to estimate the aggregated data points' effects on model parameter updating, and outputs the critical parts of input data. Note that in estimation, the number of aggregated points is sufficiently small such that the processing only takes a small proportion of the training time.

3.2 Input Data Aggregator

The *input data aggregator* module generates aggregated data points for two purposes. (1) *Data similarity preservation*: it groups similar data points of input data and stores their averaged attribute information to preserve data similarity. This tries to guarantee similar correlations to model parameter updating between an aggregated data point and the original data points it represents. (2) *Fast generation*: it completes the generation process quickly even when handling large datasets. To this effect, three steps are used in input data aggregation.

Step 1. Dimensionality reduction. This step operates on the attribute part (the $N \times d$ dataset X) of input data. To handle high-dimensional data efficiently, this step employs the incremental SVD method [26] to transform dataset X (either sparse or dense) into a dense $N \times v$ data set X', where $v << d$. This method is an *approximation* of standard SVD and it is used for two reasons. First, it minimizes the difference (i.e. the Frobenius norm between X and X') between the two data sets in transformation. Second, its execution time is independent of the data dimensionality d: the transformation is an iterative optimization process whose time complexity is O($v \times e \times a$), where e is the number of iterations for each dimensionality and a is the number of attributes used in each iteration (that is, a finite number of attributes is used at each iteration). Hence the transformation can be completed quickly when dealing with high-dimensional datasets.

Step 2. Locality sensitive division. This step operates on the $N \times v$ reduced dataset X' and divides its N points into 2^v parts, where each part consists of $\frac{N}{2^v}$ points that are similar in attribute values. Specifically, the division starts from the whole dataset and splits them recursively using v splits. Each split first sorts all the data points in each part in an ascending order according to their attribute values in a particular dimension j ($1 \leq j \leq v$) and then divides the ranked points into two equal parts, guaranteeing that the points with similar attribute values are grouped in the same part under the split.

Note that compared to the other grouping methods such as hash, clustering and index tree, our approach is straight forward and has weaker guarantee of dividing related data points. On the other hand, this method is employed for two reasons. (1) *Uniform data division*. To make fair comparison of aggregated data points effects on model parameter updating, these points are generated at the same level of granularity. An aggregated data point's level of granularity represents the number of input data points whose attribute information is summarized by this point. Hash methods such as locality sensitive hashing (LSH) [21] provide *non-uniform grouping* of data points: the number of data points in a group can be 100 times larger than another groups number of data points. (2) *Fast data division*. Our approach completes the division much faster than transitional data grouping methods such as clustering and index tree. Take k-means clustering and R-tree as examples to compare. The time complexities of our method is $O(v \times N)$. In contrast, the time complexity of k-means is $O(2^v \times i \times N)$, where v is the number of cluster centers and i is the number of iterations ($2^v \times i >> v$), and the time complexity of constructing a R-tree is $O(logN \times N)$ ($logN >> v$). Let v=5, i=30, and N=1,000,000, our method is 480 and 200 times faster than k-means and R-tree, respectively.

Step 3. Information aggregation. According to the division result, the final step summarizes the information of the original input data points in each part and generates an aggregated data point (Definition 3) and an *index file*, which records its mapping relationship to the original data points. Note that step 2 divides data points in the *reduced* attribute space while step 3 compute the averages of the *original* data points' attributes and category values.

Definition 3 (Aggregated data points). *Suppose an aggregated data point* ($\vec{a}$,$\hat{y}$) ($\vec{a} = (a_1, a_2, ..., a_d)$) *corresponds to a set of* m *input data points* $\{(\vec{x}^{(i)}, y^{(i)})\}_{i=1}^m$ ($\vec{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_d^{(i)})$)*, then*

$$a_j = \frac{w_j \times \Sigma_{i=1}^m x_j^{(i)}}{m}, \quad \hat{y} = \frac{\Sigma_{i=1}^m y_j^{(i)}}{m} (1 \leq j \leq d) \quad (5)$$

Where w_j represents feature j's weight (importance). Typically, there are two ways to calculate the feature importance in a ML algorithm. First, a simple way is to estimate feature importance before training according to the characteristics of the dataset itself. A possible method is to calculate each features associated information gain using decision trees. Moreover, in many iterative ML algorithms, the importance of a feature varies when the model changes across the iterations of training. The permutation feature importance method [46] is commonly used to calculate a feature's importance as the difference of model errors between the unchanged feature and a feature with a shuffled value.

Fig. 5 shows an example of the input data aggregation process. Step 1 transforms a 12×5 dataset X into a 12×2 dataset X'. We can see that data points with similar attribute values (e.g., points $\vec{x}^{(1)}$ and $\vec{x}^{(2)}$ in X) still have similar attributes in X'. Step 2 operates on the 2-dimensional dataset X' and divides the 12 data points into four equal parts (subsets), where each part has three data points of similar

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2951388, IEEE Transactions on Knowledge and Data Engineering

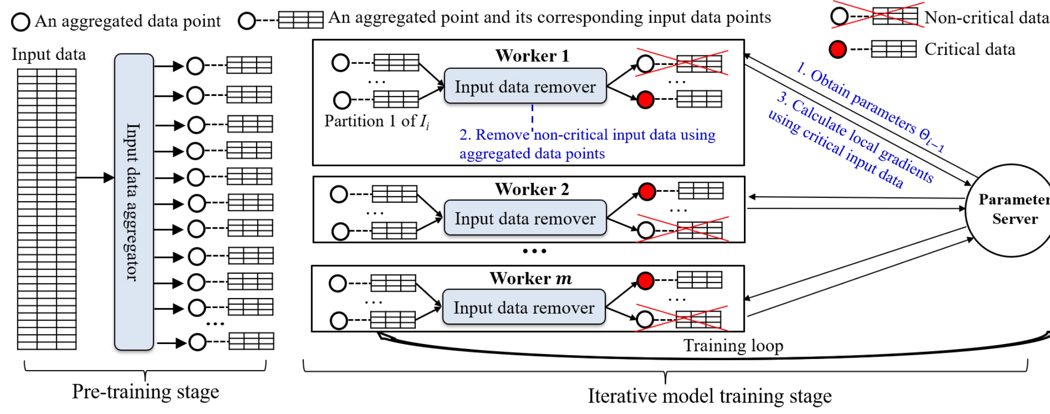
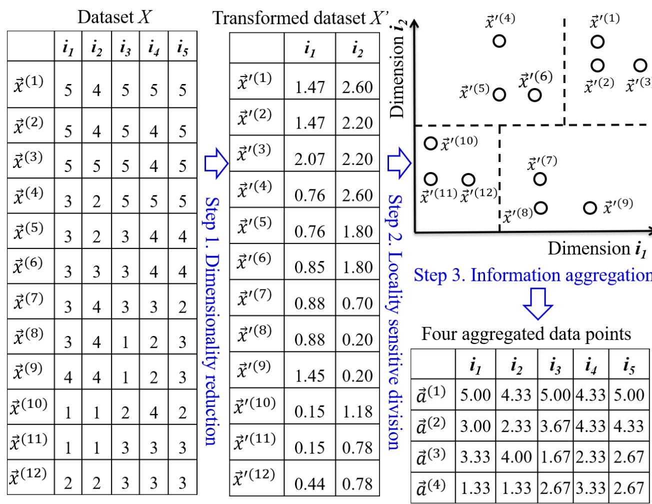


Fig. 4. Overall process of SlimML



Dataset X

	i_1	i_2	i_3	i_4	i_5
$\vec{x}^{(1)}$	5	4	5	5	5
$\vec{x}^{(2)}$	5	4	5	4	5
$\vec{x}^{(3)}$	5	5	5	4	5
$\vec{x}^{(4)}$	3	2	5	5	5
$\vec{x}^{(5)}$	3	2	3	4	4
$\vec{x}^{(6)}$	3	3	3	4	4
$\vec{x}^{(7)}$	3	4	3	3	2
$\vec{x}^{(8)}$	3	4	1	2	3
$\vec{x}^{(9)}$	4	4	1	2	3
$\vec{x}^{(10)}$	1	1	2	4	2
$\vec{x}^{(11)}$	1	1	3	3	3
$\vec{x}^{(12)}$	2	2	3	3	3

Transformed dataset X'

	i_1	i_2
$\vec{x}'^{(1)}$	1.47	2.60
$\vec{x}'^{(2)}$	1.47	2.20
$\vec{x}'^{(3)}$	2.07	2.20
$\vec{x}'^{(4)}$	0.76	2.60
$\vec{x}'^{(5)}$	0.76	1.80
$\vec{x}'^{(6)}$	0.85	1.80
$\vec{x}'^{(7)}$	0.88	0.70
$\vec{x}'^{(8)}$	0.88	0.20
$\vec{x}'^{(9)}$	1.45	0.20
$\vec{x}'^{(10)}$	0.15	1.18
$\vec{x}'^{(11)}$	0.15	0.78
$\vec{x}'^{(12)}$	0.44	0.78

Four aggregated data points

	i_1	i_2	i_3	i_4	i_5
$\vec{a}^{(1)}$	5.00	4.33	5.00	4.33	5.00
$\vec{a}^{(2)}$	3.00	2.33	3.67	4.33	4.33
$\vec{a}^{(3)}$	3.33	4.00	1.67	2.33	2.67
$\vec{a}^{(4)}$	1.33	1.33	2.67	3.33	2.67

Fig. 5. An example of input data aggregator

attribute value. Finally, step 3 generates four aggregated data points $\vec{a}^{(1)}$ to $\vec{a}^{(4)}$.

3.3 Input Data Remover

The *input data remover* module is designed for both the parameter server and the MapReduce architectures. As shown in Figure 6, each map task processes one partition of input data points to compute gradients for one partition of model parameters using two steps. First, the module removes non-critical input data points according to the model parameters obtained in the previous iteration. Secondly, it uses the remaining input data to update these parameters.

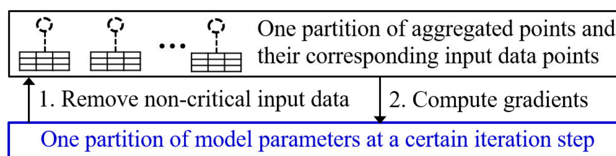


Fig. 6. Input data remover in a map task

Specifically, this module uses aggregated data points as an approximation of input data and estimates these points' effects on model parameter updating. If an aggregated data point's effect is lower than a removal threshold ϵ, its corresponding input data points are removed according to its index file. Hence the **number** of aggregated data points determines both the **overheads** and **effectiveness** of the input data removal. On the one hand, using aggregated data points to remove non-critical input data requires extra processing of these data points. The number of aggregated data points, therefore, should be much smaller (e.g. 10 or 100 times smaller) than the number of original input data points, guaranteeing this processing time only takes a small proportion of the model training time. On the other hand, a sufficient number of aggregated data points allows the fine-grained differentiation of the different parts of the input data represented by these points, thus enabling the accurate computation of these parts' effects.

Based on these aggregated data points, the steps of input data removal in a parallel worker are detailed in Algorithm 1. At each iteration of model training, this module first computes the effects of all m aggregated data points (line 1 to 3). It then employs a ranking method to sort all the aggregated points in an ascending order according to their effects on model parameter updating (line 4) and add them to set C (line 5). Subsequently, the first i points are identified as non-critical ones if their accumulated effects divided by the accumulated effects of all m points is smaller than or equal to the threshold ϵ, and they are removed from set C (line 6 to 10). Finally, the algorithm returns the critical input data according to the aggregated data points in C (line 11).

3.4 Implementation

SlimML is implemented in Scala and it is currently targeted for ML applications running on Spark [1]. Its *input data aggregator* module is implemented based on the open source packages of SVD, and its *input data remover* module is incorporated with typical iterative ML algorithms.

The *Input data aggregator* module operates on a ML algorithm's input data and it is independent of the iterative training process. Among its three steps, step 1 (dimensionality reduction) is the most computationally expensive one that takes most of the generation time. We therefore use the incremental SVD method [5] to approximate the matrix transformation. Step 1 treats the dimensionality reduction

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2951388, IEEE Transactions on Knowledge and Data Engineering

Algorithm 1 Input data removal in a parallel worker

Require: $(\vec{a}, y)$: an aggregated data point that corresponds to multiple input data points;
$\{(\vec{a}^{(1)}, y^{(1)}, (\vec{a}^{(2)}, y^{(2)}, ..., (\vec{a}^{(m)}, y^{(m)}\}$: the partition (set) of m aggregated data points in the task;
e: an aggregated data point's effect on parameter updating;
Θ: the partition of model parameters in the task; ϵ: the threshold of non-critical input data;
C: the set of critical aggregated data points.

1. **for** i=1 to m **do**
2. Compute $(\vec{a}^{(i)}, y^{(i)}$'s effect $e^{(i)}$ using Eqn. (3);
3. **end for**
4. Sort the m aggregated data points in ascending order according to their effects;
5. $C=\{\vec{a}^{(1)},\vec{a}^{(2)},...,\vec{a}^{(m)}\}$;
6. **for** i=1 to m **do**
7. **if** $(\sum_{j=1}^{i} e^{(j)} / \sum_{j=1}^{m} e^{(j)} \leq \epsilon)$ **then**
8. $C = C \setminus \{(\vec{a}^{(i)}, y^{(i)})\}$;
9. **end if**
10. **end for**
11. Return the set of input data points represented by the aggregated data points in C.

process as a gradient descent optimization problem whose running time can be controlled by setting the number of iterations and the number of attributes used at each iteration.

The advantages of generating aggregated data points are two fold: (i) removing the non-critical input data before the training of each iteration, and more importantly (ii) incorporating the *input data remover* module into a ML algorithm does not require any modification in its model training process. We incorporated this module into three ML algorithms introduced in Section 2.1: (1) **NN regression** is implemented as a network of sigmoid activation functions [56]: $f_{NN}(\vec{x}, \Theta) = \Sigma_{i=1}^{n_h}(w_{i(d+2)+d+1} \cdot s(\Sigma_{j=0}^{d-1} w_{i(d+2)+j} \cdot x_j + w_{i(d+2)+d}))$ ($w_i \in \Theta$), where $s(x)$ is the sigmoid activation function $\frac{1}{1+e^{-x}}$, n_h is the number of neurons in the hidden layer, and d is the data dimensionality. (2) **SVM classifier** [19] is implemented as: $f_{SVM}(\vec{x}, \Theta) = \vec{w}^T\vec{x} + b = \sum_{i=1}^{N} \alpha_i y^{(i)} \kappa(\vec{x}^{(i)}, \vec{x}) + b$ ($\vec{w}, b \in \Theta$), where α_1 to α_N are the Lagrange multipliers and $\kappa(\vec{x}^{(i)}, \vec{x})$ is a kernel function such as Gaussian RBF kernel $\exp(-\gamma\|\vec{x}^{(i)} - \vec{x}\|^2$. (3) **CNN** is implemented as two representative architectures: LeNet-5 [42] and AlexNet [41]. In addition to the input and output layers, all two architectures have multiple convolutional layers (2 for LeNet-5, and 5 for AlexNet), 2 pooling layers, and 1 or 2 FC layers (1 for LeNet-5, and 2 for AlexNet).

To support the processing of large datasets, the distributed versions of all three algorithms are built upon the MapReduce paradigm and the resilient distributed dataset (RDD) data structure in Spark [59], and they are implemented on Spark Machine Learning Library (MLlib) [8] (supporting the synchronous SGD) and BigDL [2] (supporting the asynchronous SGD [50] and the parameter server paradigm [18], [43]). The implementation modifies the Distributed optimizer function, which conducts the model training in parallel workers. Figure 7 lists the four steps of this function: step 1 initializes the model parameters and distributes their replication to workers. Step 2 samples training data from the input data, transforms it into the mini-batch RDD structure, and partitions it to workers. Subsequently, step 3 calculates gradients using the partitions of model parameters and training samples in each worker. At the end of each iteration, step 4 aggregates gradients from these workers and updates the model. Our implementation modifies steps 2 and 3 of this function. Specifically, the mini-batch data structure at step 2 is modified to describe aggregated data points and their corresponding input data points. At step 3, Algorithm 1's non-critical input data removal is added before the gradient calculation. The implementation of each ML algorithm requires changing less than 100 lines of code.

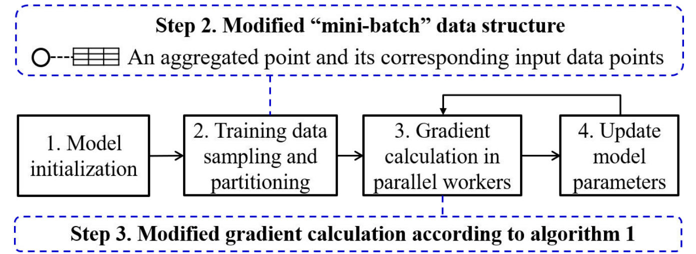


Fig. 7. The Distributed optimizer function in MLLib/BigDL

4 EVALUATION

In this section, we first evaluate SlimML's salient feature in removing non-critical input data by quickly generating and processing aggregated data points (Section 4.2), and then highlight its effectiveness in reducing model training times with negligible accuracy losses (Section 4.3). Finally, we discuss SlimML's applicability in different optimization and sampling methods (Section 4.4).

4.1 Experimental Settings

Experiment platform. The experiments were conducted on a YARN cluster with one master node and 10 worker nodes connected through 1 GB ethernet network cards. Each node is equipped with two Intel Xeon E5645 processor cores, 32 GB of DRAM, and one 1 TB 7200RPM SATA disk drive, and runs Linux Ubuntu 14.04.1. The Spark cluster has 40 executors, each executor has two vcores and 4 GB memory, and the driver memory is 20GB. The JDK, Spark, Scala versions are 1.8.0, 2.0.0, and 2.11.8, respectively.

Tested workloads and datasets. We test workloads of three ML applications (NN regression, SVM, and CNN) based on the implementations in Section 3.4. For each application, different model complexities are tested: (1) *NN regression*: the number of neurons in the hidden layer is set to 50 or 100; (2) *SVM*: the Gaussian RBF kernel is used and the gamma parameter is set to $\frac{1}{18}$ (i.e. 1 over the number of features) or $\frac{1}{9}$ (representing a more complex model); and (3) *CNN*: LeNet-5 and AlexNet architectures are used. As listed in Table 1, NN regression, SVM, LeNet-5 and AlexNet are tested using NEX [7], Wearable Stress and Affect Detection (WESAD) [10], GSA [4], MNIST [6], TinyImages (extracted from the 100 classes of the Tiny Images dataset [9]) and Cifar10 [3] datasets, in which the

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2951388, IEEE Transactions on Knowledge and Data Engineering

numbers of training/testing points are 9,691,154/2,422,788, 50,400,000/12,600,000, 6,709,412/1,677,353, 50,000/10,000, 200,000/50,000 and 50,000/10,000, respectively.

TABLE 1
Summary of workloads and datasets

Workload	NN regression	SVM	CNN	
			LeNet-5	AlexNet
Dataset	NEX,WESAD	GSA	MNIST, Tiny-Images	Cifar10

Compared techniques. The stardard SGD method and two sampling techniques widely used in todays large-scale ML are compared. (1) *Coreset* [13], [34]. This technique constructs a weighted subset of the input data (called a coreset) that only contains 25% of the original input data and uses this coreset in model training. The current coreset construction algorithm is developed for NN regression. (2) *Importance sampling* [48], [49], [55], [61]. This technique skews the sampling towards important input data points according to the distribution of their losses, thus reducing the variance of gradient estimates. It can be applied in both NN regression [16] and CNN [37], [38].

Model training settings. We set the hyperparameters of model training according to the commonly used ones or default values in current ML libraries and BigDL [2]. In detail, the mini-batch gradient descent method [14] is used to train NN regression and CNN. Each epoch divides the input data into 100 subsets, the learning rate is set to 0.01, and the regularization parameter λ=0.1. The *SGD* method [51] is used to train SVM. Each iteration takes 10k training samples, the learning rate is set to 1.49×10^{-7} (i.e. 1 divided by the number of training instances), and the regularization parameter λ=1. For all compared techniques, we use the same hyperparameters, batch size, and initial model parameters in comparative evaluations.

Evaluation Metrics. Both performance and accuracy metrics are used to evaluate the ML process across a number of iterations. At a certain iteration, the *performance* metric is the model training time. In SlimML, this time is the sum of the generation and processing times of aggregated data points and the model training time on critical input data. In addition, the *accuracy* metric is MSE on the test set for NN regression, and its classification accuracy on the test set for SVM and CNN. We report the top-1 accuracy: the top predicted class (the one having the highest probability) is the same as the target class label.

4.2 Overheads and Effectiveness of Input Data Removal

The effectiveness of input data removal relies on quickly generating aggregated data points and using them to compute effects of different parts of input data.

Generation of aggregated data points. We tested the three steps of generating aggregated data points in SlimML. At step 1, the input dataset is transformed to a low-dimensional dataset using the incremental SVD, In transformation, the number of iterations for each dimensionality is 10 and the number of attributes in each iteration is 5% of the data attributes. At step 2, the dataset is divided into different subsets according to the aggregation ratios. At step 3, the information of each subset is aggregated to generate an aggregated data point. Step 1 takes a majority (over 95%) of generation time, and thus the generation time is mainly determined by the running time of incremental SVD rather than the aggregation ratio. Table 2 reports the generation time of each dataset and it is two to three orders of magnitude shorter than that of model training time listed in Table 3. We also compared SlimML to the coreset approach [13], [34] when processing the same datasets (note that the current coreset techniques cannot be applied to image datasets), because both approaches are conducted at the pre-training stage. We can see in Table 2 that our approach is 3.44 times faster than coreset.

TABLE 2
Comparison of generation time (seconds) at the pre-training stage

Dataset	NEX	WESAD	GSA	MNIST	Cifar10	TinyImage
SlimML	148.11	943.38	6.44	11.73	35.64	106.09
Coreset	510.82	3205.41	20.04			

TABLE 3
Comparison of whole model training time (hours)

Dataset	NEX	WESAD	GSA	MNIST	Cifar10	TinyImage
SlimML	2.95	4.04	12.16	1.10	2.70	16.30
Coreset	5.44	24.61	27.35			

Processing aggregated data points. To evaluate the computation costs of processing points, we divide each iteration's model training process into two parts: processing aggregated data points and removing non-critical input data (part 1), and model training using critical input data (part 2). For either part, we report its **percentage computation time**, which denotes the execution time of this part divided by the total execution time. For each ML algorithm and dataset, we test three cases of input data aggregation, which are denoted by three **aggregation ratios** (the number of original input data points divided by the number of aggregated data points), as shown in Fig. 8. Some algorithms process larger datasets, so larger aggregation ratios (50, 100 and 200) are used. We can see that the execution time of part 1 is inversely proportional to the aggregation ratio. That is, a larger compression ratio (e.g. 200) means shorter execution times of part 1. Overall, SlimML quickly processes aggregated data points such that it takes an average of 2.46% of the total model training time.

Effectiveness of removing non-critical input data. Following the setting of the previous evaluation (aggregation ratios are 100, 100 and 10 for NN regression, SVM and CNN, respectively), we evaluate the effectiveness of input data removal at each iteration of model training. The threshold of non-critical input data is 1% of the total effect, and the NEX, GAS, MNIST, and Cifar10 datasets are tested. In evaluation, we report the **percentage of input data removal**, which denotes the amount of removed input data in a worker by applying SlimML divided by that amount without our approach. Fig. 9(a) to (c) first show the execution time of the iterative training process. For NN regression and SVM, the training processes take 10k and 300 iterations to converge, respectively. For the LeNet-5 and AlexNet architectures of

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2951388, IEEE Transactions on Knowledge and Data Engineering

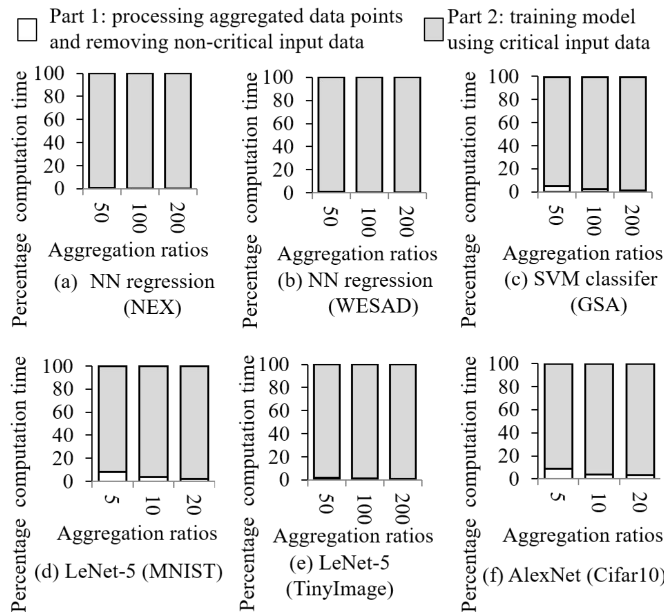


Fig. 8. Percentage computation time breakdown under three aggregation ratios.

CNN, the training processes take 100k and 10k iterations to converge, respectively.

Fig. 9(d) to (f) show this percentage in each model during its iterative training process. With SilmML, 55.33%, 74.14%, and 84.45% of input data are reduced in the models of NN regression, SVM, and CNN, respectively. This reduction is primarily determined by three factors: (1) *the characteristic of input data*: the NEX dataset in NN regression contains the smallest proportion of non-critical input data, so the percentage of reduction is also the lowest in this algorithm; (2) *the current model parameters*: the values of model parameters continuously change over the iterations, which cause more input data points becoming non-critical and hence more computational cost can be saved; (3) *the aggregated data points*: SlimML uses aggregated data points' effects to approximately represent those of input data points and decide the removal of non-critical ones.

4.3 Acceleration of Model Training

In this section, we first evaluate the effectiveness of SlimML in accelerating the standard SGD training approach following the experimental setting of the previous section. Fig. 10 shows the comparative results in terms of training time (x axis) and accuracy on test sets (y axis). First, Fig. 10(a) to Fig. 10(c) show the comparison results when training models of lower complexity. We can see that during the iterative training process, applying SlimML to remove non-critical input data achieves considerable reductions in training times when obtaining the same accuracies. Hence the training algorithms with SlimML converge much faster, while obtaining very similar accuracies in most of the cases. These results verify that SlimML correctly removes input data that has a much lower level of influence on model parameter updating than the retained input data, thus resulting in negligible accuracy losses.

Fig. 10(d) to Fig. 10(g) show the results when training models of higher complexities. We can observe that SlimML displays more obvious superiority over algorithms without input data removal by reducing model training by an average of 3.72 times (this reduction is 2.45 times for models of lower complexity). In particular, for the AlexNet with the most numbers of model parameters, SlimML achieves 4.63 times reduction in model training times with negligible accuracy loss of 0.10%. The above results verify SlimML's applicability to different ML algorithms and indicate that it has more advantages in complex models.

We further compare SlimML to the two sampling techniques using large datasets (WESAS and TinyImage). Figures 11(a) and (c) demonstrate the training time (x axis) and accuracy (y axis) of the four techniques. For SlimML (coreset), the training time includes both the generation time of aggregated data points (coresets) and the iterative model training time. We can see that the *importance sampling* approach needs the longest training time. This indicates that although this approach requires less number of iterations to converge, it also needs to constantly calculate all data points' losses in order to update the distribution. Hence each iteration takes much longer time than that of the standard SGD approach (e.g. 40% longer in CNN). In contrast, the *coreset* approach considerably decreases the training time because it only uses a smaller subset of input data for training. However, this technique also incurs the largest accuracy losses: 2.64 times larger than those of SlimML and importance sampling approaches. The results show SlimML achieves both the shortest training time and small accuracy losses, because its input data removal (see Figures 11(b) and (d)) relies on the processing of aggregated data points according to the latest model parameters, and the number of aggregated data points is much smaller than that of the input data points.

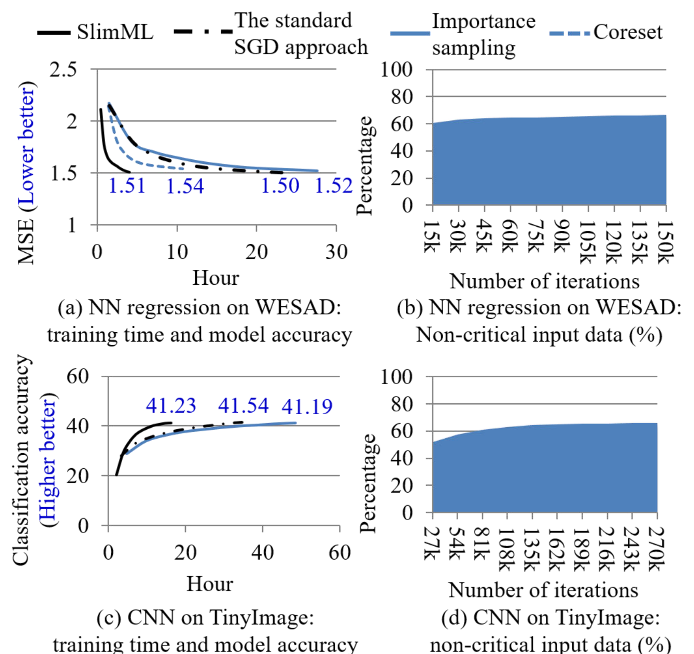


Fig. 11. Comparison of training time and accuracy with baseline techniques

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2951388, IEEE Transactions on Knowledge and Data Engineering

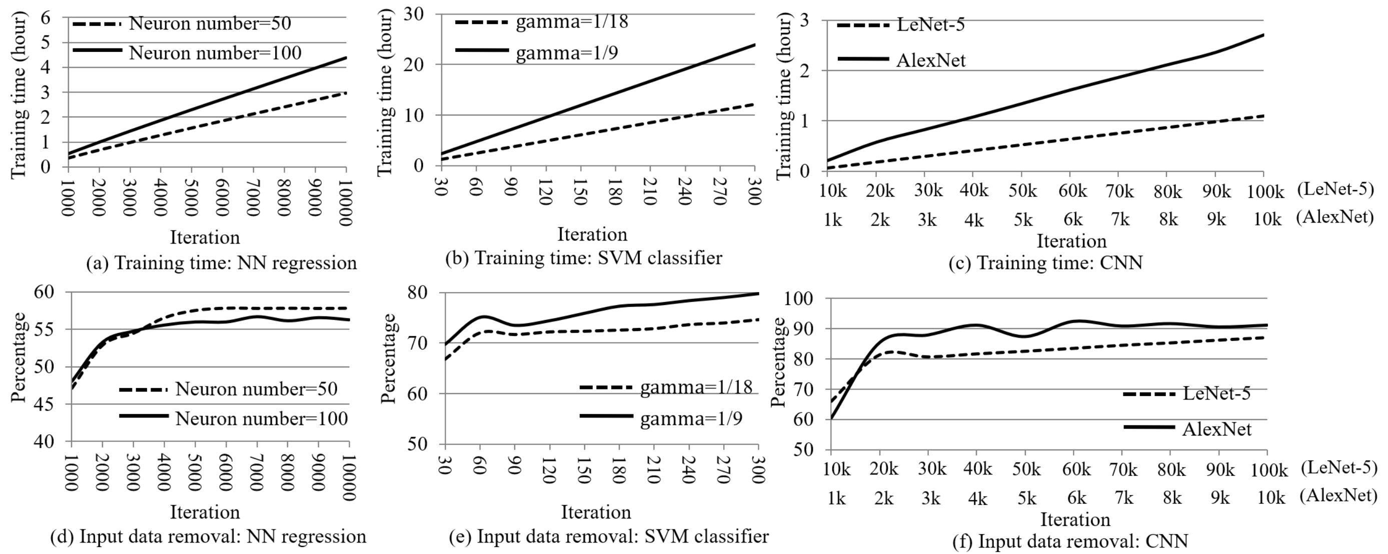


Fig. 9. Applying SlimML to the standard SGD approach under different model complexities. The top figures ((a) to (c)) show the iterative model training times across iterations and bottom figures ((d) to (f)) shows the percentages of input data removal.

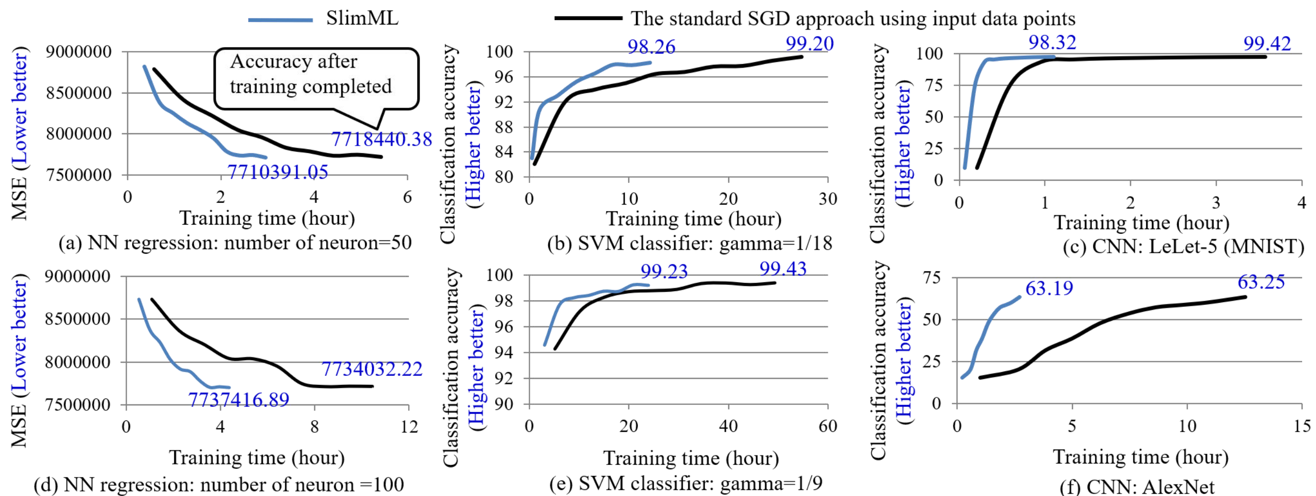


Fig. 10. Comparison of training time and accuracy with and without removal of non-critical input data in standard SGD approach

Results. *When applying SlimML to remove non-critical input data, the model training is accelerated by an average of 3.61 times with small accuracy losses of 0.37% in the evaluations of NN regression, SVM, and CNN workloads.*

4.4 Discussion of the Applicability in Other Optimization and Sampling Methods

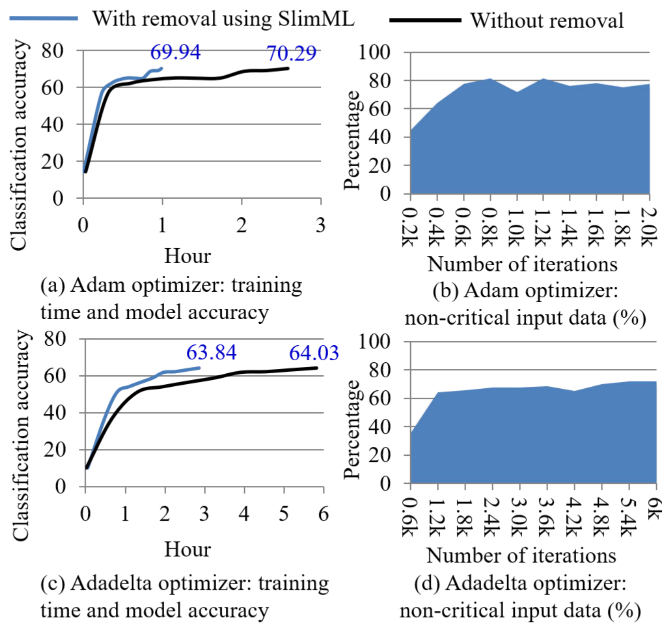
Discussion 1: Optimization methods. We take *AlexNet* and *Cifar-10* dataset as an example, and design two experiments to demonstrate the applicability of SlimML with other optimization methods in addition to the SGD method shown in the previous evaluations. Two methods considered are: Adam [39] (a first-order gradient-based optimization method) and Adadelta [60] (an adaptive learning rate technique). The detailed model setting of these methods follow the benchmark provided in [2].

Evaluation results. Figures 12(a) and (c) show the model training time and its test accuracy of either standard optimization method (without input data removal) and the one with removal using SlimML. We can observe that compared to SGD (Figure 10(f)), both optimization methods achieve better model accuracies using less training times. This is because they optimize the model parameter updating process, needing less iterations in training (2k iterations in Adam and 6k iterations in Adadelta, as shown in Figures 12(b) and (d)). SlimML is orthogonal to these methods such that it reduces the training time of each iteration by removing non-critical input data against all three optimization methods. The evaluation results show that SlimML further reduces training time by half while achieving the same model accuracy (Figures 12(a) and (c)), because it reduces large proportions of input data (72.75% in Adam and 64.78% in Adadelta, as shown in Figures 12(b) and (d)).

Discussion 2: Integration with Importance sampling. We implemented the representative importance sampling method for deep learning [11], [45], which creates a multino-

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2951388, IEEE Transactions on Knowledge and Data Engineering

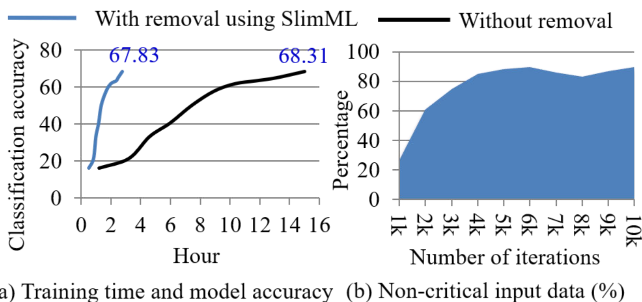


(a) Adam optimizer: training time and model accuracy
(b) Adam optimizer: non-critical input data (%)
(c) Adadelta optimizer: training time and model accuracy
(d) Adadelta optimizer: non-critical input data (%)

Fig. 12. Applying SlimML in the Adam and Adadelta optimization methods

mial distribution using the input data points' losses. At each iteration, we first sampled the training points based on the distribution and remove the non-critical input data using SlimML. In evaluation, the same mini-batch is used as the previous sections and the data points' losses are computed at each epoch.

Evaluation results. Figure 13(a) shows that compared to SGD with random sampling (Figure 10(f)), the basic importance sampling method (without removal) improves model accuracy by about 5% via reducing the variance of the gradient estimates in training. However, this method also needs 19.95% longer time to complete the training process because its expensive computation of input data points' losses. In contrast, SlimML mitigates this phenomenon by only calculating the losses of the aggregated points (for data sampling), whose number is much smaller than that of input data points. Furthermore, our approach removes an average 77.00% non-critical ones from the sampled points (Figure 13(b)), and reduces model training time by an average of 4.83 times with accuracy losses of less than 1% (Figure 13(a)).



(a) Training time and model accuracy (b) Non-critical input data (%)

Fig. 13. Applying SlimML in AlexNet training using the importance sampling method

Discussion 3: removal threshold. In our approach, the removal threshold is a key parameter that determines the number of original data points to be removed from model training. The variations in the threshold therefore influence both training time and model accuracy. In previous evaluations of the CNN workload, the threshold is set to 1%, which allows the data points whose cumulative effects are smaller than 1% of the total effects of all points to be removed. In this evaluation, we first tested three thresholds: 0.8%, 1%, and 1.25%, which lead to considerable differences in the percentages of removed input data during the iterative training process, as shown in Figure 14(a). We further tested six thresholds for two different workloads and datasets, Figure 14(b) shows the percentages of input data removal at the end of model training. In both evaluations, we can see that lower thresholds always bring smaller numbers of input data points to be removed, thus resulting in longer training time. On the other hand, a larger threshold achieves more performance improvement, but also incurs larger accuracy losses (e.g. the accuracy loss is 7.49% if the threshold is 1.25%). We can also observe that when the thresholds are small (e.g. 0.6% and 0.8%), an increase in the threshold value brings considerable improvement. However, the improvement becomes negligible when the threshold is larger than a value (e.g. 1.5% in TinyImage or 1.25% in Cifar10). This is because the input data points' effects on model parameter updating have a long tail in the distribution (as shown in Figure 2), and most of them are identified as non-critical data when the threshold is larger than the value. In conclusion, SlimML can make trade-offs between training time and model accuracy with the threshold, and it is possible to introduce an automatic method to dynamically adjust the removal threshold during the iterative training process, while maintaining the accuracy loss below a user-specified value.

5 Related Work

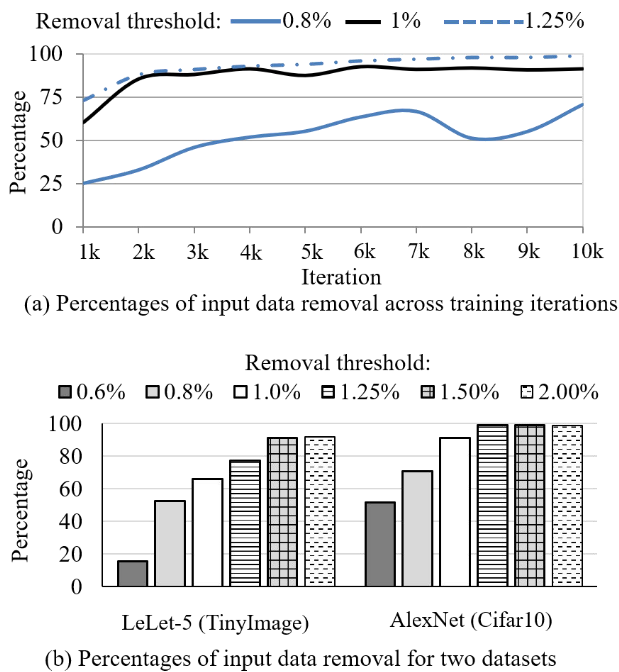
We can summarise the key methodologies currently used in large-scale ML into two camps as described below.

5.1 Massive Input Data Points

When handling massive data points, **data parallelization** techniques process them in a parallel and distributed fashion, thus accelerating the training process of ML algorithms (e.g. mini-batch based neural network or SGD based SVM). These techniques mainly concentrate on how to address the challenges in data parallelization such as stragglers [30] and data locality [58], or combine data parallelization and algorithm parallelization to create an optimal execution plan [15]. SlimML is built upon the dominant data parallel paradigm - MapReduce [22], and it focuses on removing non-critical input data in map tasks.

The most related approach to SlimML is coresets and importance sampling that also select a part of more important (distinctive) input data points for model training. Specifically, the **coreset** approach [13], [34] constructs a smaller, weighted subset of the input data to approximate the full dataset and then uses this subset (i.e. coreset) in model training. This construction process is completed at the pre-training stage based on the input dataset itself. However, the

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.



(a) Percentages of input data removal across training iterations

(b) Percentages of input data removal for two datasets

Fig. 14. Percentages of input data removal under different removal thresholds.

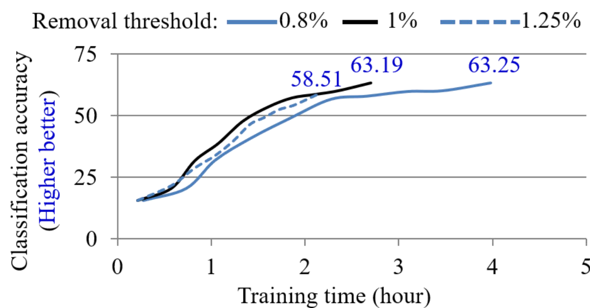


Fig. 15. Comparison of training time and accuracy under different removal thresholds.

model is constantly updated during the iterative training process, hence the importances of data points fluctuate and the fixed coreset may poorly reflect the current situation.

For such an issue, recent **importance sampling** techniques create a distribution to sample the most important data points, and update the distribution at each iteration to reflect the latest changes in model parameters [48], [49], [55], [61]. In recent years, importance sampling has been applied in many iterative ML algorithms such as regression [16] and deep neural networks [37], [38]. However, it suffers from being computationally expensive because each update of the distribution needs to compute all input data points' gradient norms or losses with respect to the model parameters [38]. In contrast, SlimML incurs smaller computational footprint using aggregated data points. First, its removal process only calculates the gradient norms of aggregated data points, whose number is much smaller than that of input data points. Second, it is integrated with the MapReduce and parameter server paradigms and thus can process partial input data and model parameters in each worker in parallel.

5.2 Massive Model Parameters

In parallel and distributed model training, large datasets result in tremendous amount of local variables (e.g. intermediate results to compute gradients in Eqn. (2)) used to update the global parameters [43]. *Parameter server* is another important category of techniques that improve the performance of accessing (reading and updating) and synchronizing local variables and global parameters [18], [20], [43], [53]. This work is implemented based on the efficient parameter synchronization of parameter server.

Model compression techniques apply lossy compression techniques such as pruning unimportant connections [35], vector Quantization [25], [33], huffman coding [29]) in massive model parameters, thus reducing computation and communication costs in model training. Within the context of large-scale ML, these techniques has been applied in both approximate nearest neighbour (ANN) search of stream data [57] and deep neural networks [29]. During the iterative training process, they treat the selected input data as *equally critical* to model accuracy and hence are complementary to the input data removal approach in this work.

6 Conclusion

In this paper, we presented SlimML to accelerate the training process of iterative ML applications, and demonstrated its effectiveness using both three popular ML algorithms. SlimML is based on two key ideas: (1) it aggregates information of input data to create small aggregated data points at the pre-learning stage; (2) using these points, it quickly estimates the input data's effects on model parameter updating at each training iteration, using the data with high estimated effects in model training. Evaluation results using real workloads and massive datasets demonstrate the effectiveness of SlimML in bringing considerable reductions in model training times while only causing negligible accuracy losses.

Acknowledgments

We thank the anonymous reviewers for careful review of our paper. This work has been supported in part by the National Key Research and Development Plan of China (Grant No. 2018YFB1003701 and 2018YFB1003700), in part by the National Natural Science Foundation of China (Grant No. 61872337), and in part by the Swiss National Science Foundation NRP75 project 407540_167266.

References

[1] Apache spark. http://spark.apache.org/.
[2] Bigdl: Distributed deep learning on apache spark.
[3] Cifar-10 database. http://yann.lecun.com/exdb/mnist/.
[4] Gsa dataset. https://archive.ics.uci.edu/ml/datasets/Gas+sensor+array+\under+dynamic+gas+mixtures.
[5] Incremental svd. http://sifter.org/~simon/journal/20061211.html.
[6] Mnist handwritten digit database. http://yann.lecun.com/exdb/mnist/.
[7] Nex dataset. https://nex.nasa.gov/nex/resources/322/.
[8] Spark machine learning library (mllib). https://spark.apache.org/docs/1.0.1/mllib-guide.html.
[9] Tinyimages dataset. http://groups.csail.mit.edu/vision/TinyImages/.

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.

[10] Wesad dataset. http://archive.ics.uci.edu/ml/datasets/WESAD+%28Wearable+Stress+and+Affect+Detection%29.
[11] Guillaume Alain, Alex Lamb, Chinnadhurai Sankar, Aaron C Courville, and Yoshua Bengio. Variance reduction in sgd by distributed importance sampling. *arXiv: Machine Learning*, 2015.
[12] Yuichiro Anzai. *Pattern Recognition & Machine Learning*. Elsevier, 2012.
[13] Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coreset constructions for machine learning. *stat*, 1050:4, 2017.
[14] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.
[15] Matthias Boehm, Shirish Tatikonda, and et al. Hybrid parallelization strategies for large-scale machine learning in systemml. In *VLDB'14*, volume 7, pages 553–564. VLDB Endowment, 2014.
[16] Olivier Canevet, Cijo Jose, and Francois Fleuret. Importance sampling tree for large-scale empirical expectation. In *ICML'16*, pages 1454–1462, 2016.
[17] Vineet Chaoji, Rajeev Rastogi, and Gourav Roy. Machine learning in the real world. volume 9, pages 1597–1600. VLDB Endowment, 2016.
[18] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *OSDI'14*, volume 14, pages 571–582, 2014.
[19] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
[20] Wei Dai, Jinliang Wei, Jin Kyu Kim, Seunghak Lee, Junming Yin, Qirong Ho, and Eric P Xing. Petuum: A framework for iterative-convergent distributed ml. In *NIPS'13*, 2013.
[21] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG'04*, pages 253–262. ACM, 2004.
[22] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
[23] Aditya Devarakonda, Maxim Naumov, and Michael Garland. Adabatch: Adaptive batch sizes for training deep neural networks. *arXiv: Learning*, 2017.
[24] Xiaopeng Fan, Jiannong Cao, and Haixia Mao. A survey of mobile cloud computing. *ZTE Communications*, 2011.
[25] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir D Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv: Computer Vision and Pattern Recognition*, 2014.
[26] Genevieve Gorrell. Generalized hebbian algorithm for incremental singular value decomposition in natural language processing. In *EACL'06*, volume 6, pages 97–104, 2006.
[27] Rui Han, Lizy Kurian John, and Jianfeng Zhan. Benchmarking big data systems: A review. *IEEE Transactions on Services Computing*, 11(3):580–597, 2017.
[28] Rui Han, Chi Harold Liu, Zan Zong, Lydia Y Chen, Wending Liu, Siyi Wang, et al. Workload-adaptive configuration tuning for hierarchial cloud schedulers. *IEEE Transactions on Parallel and Distributed Systems*, 2019.
[29] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. pages 1–14. 2016.
[30] Aaron Harlap, Henggang Cui, Wei Dai, Jinliang Wei, Gregory R Ganger, Phillip B Gibbons, Garth A Gibson, and Eric P Xing. Addressing the straggler problem for iterative convergent parallel ml. In *SoCC'16*, pages 98–111, 2016.
[31] Aaron Harlap, Alexey Tumanov, Andrew Chung, Gregory R Ganger, and Phillip B Gibbons. Proteus: agile ml elasticity through tiered reliability in dynamic resource markets. In *EuroSys'17*, pages 589–604, 2017.
[32] Yuzhen Huang, Tatiana Jin, Yidi Wu, Zhenkun Cai, Xiao Yan, Fan Yang, Jinfeng Li, Yuying Guo, and James Cheng. Flexps: Flexible parallelism control in parameter server architecture. *Proceedings of the VLDB Endowment*, 11(5):566–579, 2018.
[33] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran Elyaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18(187):1–30, 2016.
[34] Jonathan Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable bayesian logistic regression. In *Advances in Neural Information Processing Systems*, pages 4080–4088, 2016.
[35] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size. *Computer Vision and Pattern Recognition (CVPR'17)*, 2017.
[36] Yichao Jin and Yonggang Wen. When machine learning meets media cloud: Architecture, application and outloo. *ZTE Communications*, 16(3):30–39, 2018.
[37] Angelos Katharopoulos and François Fleuret. Biased importance sampling for deep neural network training. *arXiv preprint arXiv:1706.00043*, 2017.
[38] Angelos Katharopoulos and Francois Fleuret. Not all samples are created equal: Deep learning with importance sampling. *ICML'18*, pages 2525–2534, 2018.
[39] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLP'15*, 2015.
[40] Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. Mlbase: A distributed machine-learning system. In *CIDR'13*, volume 1, pages 2–8, 2013.
[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105, 2012.
[42] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
[43] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *OSDI'14*, volume 14, pages 583–598, 2014.
[44] Lei Liu, Shengjie Yang, Lu Peng, and Xinyu Li. Hierarchical hybrid memory management in os for tiered memory systems. *IEEE Transactions on Parallel and Distributed Systems*, 2019.
[45] Ilya Loshchilov and Frank Hutter. Online batch selection for faster training of neural networks. *arXiv: Learning*, 2015.
[46] Christoph Molnar et al. Interpretable machine learning: A guide for making black box models explainable. *E-book at¡ https://christophm. github. io/interpretable-ml-book/¿, version dated*, 10, 2018.
[47] Yadong Mu, Wei Liu, Xiaobai Liu, and Wei Fan. Stochastic gradient made stable: A manifold propagation approach for large-scale optimization. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):458–471, 2017.
[48] Deanna Needell and Rachel Ward. Batched stochastic gradient descent with weighted sampling. In *International Conference Approximation Theory*, pages 279–306. Springer, 2016.
[49] Deanna Needell, Rachel Ward, and Nati Srebro. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in neural information processing systems*, pages 1017–1025, 2014.
[50] Chengjie Qin, Martin Torres, and Florin Rusu. Scalable asynchronous gradient descent optimization for out-of-core models. *Proceedings of the VLDB Endowment*, 10(10):986–997, 2017.
[51] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.
[52] Samuel L Smith, Pieterjan Kindermans, and Quoc V Le. Don't decay the learning rate, increase the batch size. 2018.
[53] Alexander Smola and Shravan Narayanamurthy. An architecture for parallel topic models. *VLDB'10*, 3(1-2):703–710, 2010.
[54] Yiheng Wang, Xin Qiu, Ding Ding, Yao Zhang, Yanzhang Wang, Xianyan Jia, Yan Wan, Zhichao Li, Jiao Wang, Shengsheng Huang, et al. Bigdl: A distributed deep learning framework for big data. *arXiv preprint arXiv:1804.05839*, 2018.
[55] Rachel Ward. Stochastic gradient descent with importance sampling. *The University of Texas at Austin*, 2014.
[56] Bogdan M Wilamowski and Hao Yu. Improved computation for levenberg–marquardt training. *IEEE Transactions on Neural Networks*, 21(6):930–937, 2010.
[57] Donna Xu, Ivor W Tsang, and Ying Zhang. Online product quantization. *IEEE Transactions on Knowledge and Data Engineering*, 30:2185–2198, 2018.
[58] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys'10*, pages 265–278. ACM, 2010.
[59] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th*

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2951388, IEEE Transactions on Knowledge and Data Engineering

USENIX conference on Networked Systems Design and Implementation, pages 2–2. USENIX Association, 2012.

[60] Matthew D Zeiler. Adadelta: An adaptive learning rate method. *arXiv: Learning*, 2012.

[61] Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *ICML'15*, pages 1–9, 2015.

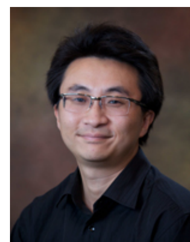
Rui Han is an Associate Professor at the School of Computer Science and Technology, Beijing Institute of Technology, China. Before joining BIT, He received MSc with honor in 2010 from Tsinghua University, China, and obtained his PhD degree in 2014 from the Department of Computing, Imperial College London, UK. His research interests are system optimization for cloud data center workloads (in particular highly parallel services, machine learning and deep learning applications). He has over 40 publications in these areas, including papers at TPDS, INFOCOM, ICDCS, ICPP, CCGrid, and CLOUD.

Chi Harold Liu (SM'15) receives the Ph.D. degree from Imperial College, UK in 2010, and the B.Eng. degree from Tsinghua University, China in 2006. He is currently a Full Professor and Vice Dean at the School of Computer Science and Technology, Beijing Institute of Technology, China. He is also the Director of IBM Mainframe Excellence Center (Beijing), Director of IBM Big Data Technology Center. Before moving to academia, he joined IBM Research - China as a staff researcher and project manager, after working as a postdoctoral researcher at Deutsche Telekom Laboratories, Germany, and a visiting scholar at IBM T. J. Watson Research Center, USA. His current research interests include the Big Data analytics, mobile computing, and deep learning. He received the Distinguished Young Scholar Award in 2013, IBM First Plateau Invention Achievement Award in 2012, and was interviewed by EEWeb.com as the Featured Engineer in 2011. He has published more than 200 prestigious conference and journal papers and owned more than 14 EU/U.S./U.K./China patents, with Google H index 26. He currently serves as the Symposium Chair for IEEE ICC 2020 Next Generation Networking, an Area Editor for KSII Trans. on Internet and Information Systems and the book editor for six books published by Taylor & Francis Group, USA and China Machinery Press. He also has served as the general chair of IEEE SECON'13 workshop on IoT Networking and Control, IEEE WCNC'12 workshop on IoT Enabling Technologies, and ACM UbiComp'11 Workshop on Networking and Object Memories for IoT. He served as the consultant to Asian Development Bank, Bain & Company, and KPMG, USA, and the peer reviewer for Qatar National Research Foundation, and National Science Foundation, China. He is a Fellow of IET, and a Senior Member of IEEE.

Shilin Li is a undergraduate student at the School of Computer Science and Technology, Beijing Institute of Technology. His work focuses on optimization of big data system for machine learning and deep learning workloads.

Lydia Y. Chen is an Associate Professor in the Department of Computer Science at the Technology University Delft. Prior to joining TU Delft, she was a research staff member at the IBM Zurich Research Lab from 2007 to 2018. She received Ph.D. from the Pennsylvania State University and B.A from National Taiwan University Her research interests center around dependability management, resource allocation and privacy enhancement for large scale data processing systems and services. She has published more than 80 papers in journals, e.g., IEEE Transactions on Distributed Systems, IEEE Transactions on Service Computing, and conference proceedings, e.g., INFOCOM, Sigmetrics, DSN, and Eurosys. She was a co-recipient of the best paper awards at CCgrid15 and eEnergy15. She is a senior IEEE member.

Guoren Wang received the BSc, MSc, and Ph.D. degrees from the Department of Computer Science, Northeastern University, Shenyang, China, in 1988, 1991, and 1996, respectively. He is now a Full Professor and Director of Institute of Data Science and Knowledge Engineering at the Department of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. He was an Assistant President for Northeastern University, China. His research interests include XML data management, query processing and optimization, bioinformatics, high dimensional indexing, parallel database systems, and cloud data management. He has published more than 150 research papers in top conferences and journals like IEEE TKDE, ICDE, SIGMOD, VLDB, etc.



Dr. Jian Tang is the Chief Scientist of Intelligent Control at DiDi AI Labs. He is also a professor in the Electrical Engineering and Computer Science Department at Syracuse University. He received his Ph.D degree in Computer Science from Arizona State University in 2006. His research interests lie in the areas of Machine Learning, Big Data, Cloud Computing, Wireless Networking and Mobile Computing. Dr. Tang has published over 130 papers in premier journals and conferences. He received an NSF CAREER award in 2009, the 2016 Best Vehicular Electronics Paper Award from IEEE Vehicular Technology Society, and Best Paper Awards from the 2014 IEEE International Conference on Communications (ICC) and the 2015 IEEE Global Communications Conference (Globecom) respectively. He has served as an editor for a few IEEE journals, including IEEE Transactions on Big Data, IEEE Transactions on Mobile Computing, IEEE Transactions on Network Science and Engineering, IEEE Transactions on Wireless Communications, etc. In addition, he served as a TPC co-chair for the 2018 International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuituous), the 2015 IEEE International Conference on Internet of Things (iThings) and the 2016 International Conference on Computing, Networking and Communications (ICNC); as the TPC vice chair for the 2019 IEEE International Conference on Computer Communications (INFOCOM); and as an area TPC chair for INFOCOM 2017-2018. He is also an IEEE Fellow, an IEEE VTS Distinguished Lecturer, and the Vice Chair of the Communications Switching and Routing Committee of IEEE Communications Society.

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2951388, IEEE Transactions on Knowledge and Data Engineering

Jieping Ye is head of DiDi AI Labs, a VP of Didi Chuxing and a Didi Fellow. He is also an associate professor of University of Michigan, Ann Arbor. His research interests include big data, machine learning and data mining with applications in transportation and biomedicine. He has served as a Senior Program Committee/Area Chair/Program Committee Vice Chair of many conferences including NIPS, ICML, KDD, IJCAI, ICDM, and SDM. He serves as an Associate Editor of Data Mining and Knowledge Discovery and IEEE Transactions on Knowledge and Data Engineering. He has won best paper awards at ICML and KDD. He won the NSF CAREER Award in 2010. He earned his Ph.D. in Computer Science and Engineering from the University of Minnesota in 2005.

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: TU Delft Library. Downloaded on February 26,2020 at 09:35:02 UTC from IEEE Xplore. Restrictions apply.