A Survey on Large-Scale Machine Learning

Meng Wang, *Senior Member, IEEE*, Weijie Fu, Xiangnan He,
Shijie Hao, and Xindong Wu, *Fellow, IEEE*

Abstract—Machine learning can provide deep insights into data, allowing machines to make high-quality predictions and having been widely used in real-world applications, such as text mining, visual classification, and recommender systems. However, most sophisticated machine learning approaches suffer from huge time costs when operating on large-scale data. This issue calls for the need of Large-scale Machine Learning (LML), which aims to learn patterns from big data with comparable performance efficiently. In this paper, we offer a systematic survey on existing LML methods to provide a blueprint for the future developments of this area. We first divide these LML methods according to the ways of improving the scalability: 1) model simplification on computational complexities, 2) optimization approximation on computational efficiency, and 3) computation parallelism on computational capabilities. Then we categorize the methods in each perspective according to their targeted scenarios and introduce representative methods in line with intrinsic strategies. Lastly, we analyze their limitations and discuss potential directions as well as open issues that are promising to address in the future.

Index Terms—Large-scale machine learning, efficient machine learning, big data analysis, efficiency, survey

✦

1 INTRODUCTION

MACHINE learning endows machines the intelligence to learn patterns from data, eliminating the need for manually discovering and encoding the patterns. Nevertheless, many effective machine learning methods face quadratic time complexities with respect to the number of training instances or model parameters [70]. With the rapidly increasing scale of data in recent years [207], these machine learning methods become overwhelmed and difficult to serve for real-world applications. To exploit the gold mines of big data, Large-scale Machine Learning (LML) is therefore proposed. It aims to address the general machine learning tasks on available computing resources, with a particular focus on dealing with large-scale data. LML can handle the tasks with nearly linear (or even lower) time complexities while obtaining comparable accuracies. Thus, it has become the core of big data analysis for actionable insights. For example, self-driving cars such as Waymo and Tesla Autopilot apply convolutional networks in computer vision to perceive their surroundings with real-time images [115]; online media and E-commerce sites such as Netflix and Amazon build efficient collaborative filtering models from users' histories to make product recommendations [18]. All in all, LML has been playing a vital and indispensable role in our daily lives.

- *Meng Wang, Weijie Fu, Shijie Hao, and Xindong Wu are with the Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology), Ministry of Education, Hefei, Anhui 230601, China, and also with the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, Anhui 230601, China.*
  E-mail: {eric.mengwang, fwj.edu, hfut.hsj}@gmail.com, xwu@hfut.edu.cn.
- *Xiangnan He is with the University of Science and Technology of China, Hefei, Anhui 230031, China. E-mail: xiangnanhe@gmail.com.*

Given the increasing demand for learning from big data, a systematic survey on this area becomes highly scientific and practical. Although some surveys have been published in the area of big data analysis [12], [33], [54], [193], they are less comprehensive in the following aspects. First, most of them only concentrate on one perspective of LML and overlook the complementarity. It limits their values for understanding this area and promoting future developments. For example, [12] focuses on predictive models without covering the optimization, [33] reviews stochastic optimization algorithms while ignoring the parallelization, and [193] only pays attention to processing systems for big data and discusses the machine learning methods that the systems support. Second, most surveys either lose the insights into their reviewed methods or overlook the latest high-quality literature. For example, [12] lacks discussions on the computational complexities of the reviewed models, [33] neglects the optimization algorithms that address the data with high dimensionality, and [120] limits its investigation to distributed data analysis in the Hadoop ecosystem.

In this paper, we thoroughly review over 200 papers on LML from computational perspectives with more in-depth analysis and discuss future research directions. We provide practitioners lookup tables to choose predictive models, optimization algorithms, and processing systems based on their demands and resources. Besides, we offer researchers guidance to develop the next generation of LML more effectively with the insights of current strategies. We summarize the contributions as follows.

First, we present a comprehensive overview of LML according to three computational perspectives. Specifically, it consists of: 1) model simplification, which reduces computational complexities by simplifying predictive models; 2) optimization approximation, which enhances computational efficiency by designing better optimization algorithms; and 3) computation parallelism, which improves computational capabilities by scheduling multiple computing devices.

Second, we provide an in-depth analysis of existing LML methods. To this end, we divide the methods in each perspective into finer categories according to targeted scenarios. We analyze their motivations and intrinsic strategies for accelerating the machine learning process. We then introduce the characteristics of representative achievements. In addition, we review the hybrid methods that jointly improve multiple perspectives for synergy effects.

Third, we analyze the limitations of the LML methods in each perspective and present the potential directions based on their extensions. Besides, we discuss some open issues in related areas for the future development of LML.

The paper is organized as follows. We first present a general framework of machine learning in Section 2, followed by a high-level discussion on its effectiveness and efficiency. In Section 3, we comprehensively review state-of-the-art LML methods and provide in-depth insights into their benefits and limitations. Lastly, we discuss the future directions to address the limitations and other promising open issues in Section 4, before concluding the paper in Section 5.

2 FROM EFFECTIVENESS TO EFFICIENCY

In this section, we present three perspectives to quantify the efficiency based on an acknowledged error decomposition on effectiveness. Then we provide a brief of LML.

2.1 Overview of Machine Learning

Notations. We consider a general setting of machine learning tasks: given n instances $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ sampled from a d-dimension space, and $\{\mathbf{y}_i\}_{i=1}^{n_L}$ indicate to the labels of the first n_L instances $(n_L \geq 0)$ within c distinct classes. The goal of machine learning is to learn an instance-to-label mapping model $f : \mathbf{x} \rightarrow \mathbf{y}$ from a family of functions $\mathcal{F}$, which can handle both existing and future data.

Effectiveness With Error Decomposition. Let $Q(f)_n$ denote the empirical risk of a machine learning model trained over n instances, and $Q(f)$ be the corresponding expected risk where the number of instances is infinite. Then we introduce the following specific functions obtained under different settings. Specifically, let $f^* = \text{argmin}_f Q(f)$ be the optimal function that may not belong to the function family $\mathcal{F}$, and $f_{\mathcal{F}}^* = \text{argmin}_{f \in \mathcal{F}} Q(f)$ be the optimal solution in $\mathcal{F}$. Suppose $f_n = \text{argmin}_{f \in \mathcal{F}} Q(f)_n$ is the optimal solution that minimizes the empirical risk $Q(f)_n$, and $\tilde{f}_n$ refers to its approximation obtained by iterative optimization.

Let $T(\mathcal{F}, n, \rho)$ be the computational time for an expected tolerance ρ with $Q(\tilde{f}_n)_n - Q(f_n)_n < \rho$. Based on the above definitions, the excess error $\mathcal{E}$ obtained within an allotted time cost T_{max} can be decomposed into three terms [32]:

$$\text{argmin}_{\mathcal{F},n,\rho} \ \mathcal{E}_{app} + \mathcal{E}_{est} + \mathcal{E}_{opt}, \ \text{s.t.} \ T(\mathcal{F}, n, \rho) \leq T_{max},$$
(1)

where $\mathcal{E}_{app} = \mathbb{E}[Q(f_{\mathcal{F}}^*) - Q(f^*)]$ denotes the approximation error, measuring how closely the functions in $\mathcal{F}$ can approximate the optimal solution beyond $\mathcal{F}$; $\mathcal{E}_{est} = \mathbb{E}[Q(f_n) - Q(f_{\mathcal{F}}^*)]$ is the estimation error, which evaluates the effect of minimizing the empirical risk instead of the expected risk; $\mathcal{E}_{opt} = \mathbb{E}[Q(\tilde{f}_n) - Q(f_n)]$ indicates the optimization error, which measures the impact of the approximate optimization on the generalization performance.

Efficiency From Three Perspectives. Based on a low degree of reduction of the above decomposition, we show the three perspectives for improving machine learning efficiency.

First, we focus on the effect of $\mathcal{E}_{app}$, which is predefined by the function family $\mathcal{F}$ of predictive models. By tuning the size of $\mathcal{F}$, we can make a trade-off between $\mathcal{E}_{app}$ and the computational complexity. Second, we consider the influence of $\mathcal{E}_{est}$. According to the probably approximately correct theory, the required number of instances for optimizing models in a large-size function family can be much larger [91]. To make use of available data and reduce the estimation error $\mathcal{E}_{est}$, we suppose all n instances are traversed at least once during optimization. Thus, we ignore $\mathcal{E}_{est}$ and the factor n. Third, we pay attention to $\mathcal{E}_{opt}$, which is affiliated with optimization algorithms and processing systems. Specifically, the algorithms play a crucial role in computational efficiency, which attempts to increase the reduction in the optimization error per computation unit. The processing systems determine the computational capacity based on the hardware for computing and the software for scheduling. With a powerful system, plenty of iterations in optimization can be performed within allotted time costs.

Above all, the machine learning efficiency can be improved from three perspectives, including 1) the computational complexities of predictive models, 2) the computational efficiency of optimization algorithms, and 3) the computational capabilities of processing systems.

2.2 Brief of Large-Scale Machine Learning

Now we present a brief of LML based on the above analysis.

1) Reduce the computational complexity based on model simplification. Machine learning models can be optimized based on matrix operations, which generally take cubic computational complexities for square matrices. Besides, most non-linear methods require extra quadratic complexities for estimating the similarity between pairs of instances to formulate their models. If these models are simplified by being constructed and optimized with smaller or sparser involved matrices, we can reduce their complexities significantly.

2) Improve the computational efficiency based on optimization approximation. Gradient descent algorithms cannot always compensate for their huge computations by selecting larger learning rates [33]. Therefore, a more effective manner is splitting data or parameters into multiple subsets and then updating models with these small subsets. As these approximate algorithms can obtain relatively reliable gradients with fewer computations, the reduction in the optimization error can be increased per computation unit.

3) Enhance the computational capacity based on computation parallelism. During the procedure of model construction and optimization, a high number of computation-intensive operations can be performed simultaneously, such as the calculations that are repeated on different mini-batches. Built upon parallel processing systems, we can break up these intensive computations and accomplish them on multiple computing devices with a shorter runtime.
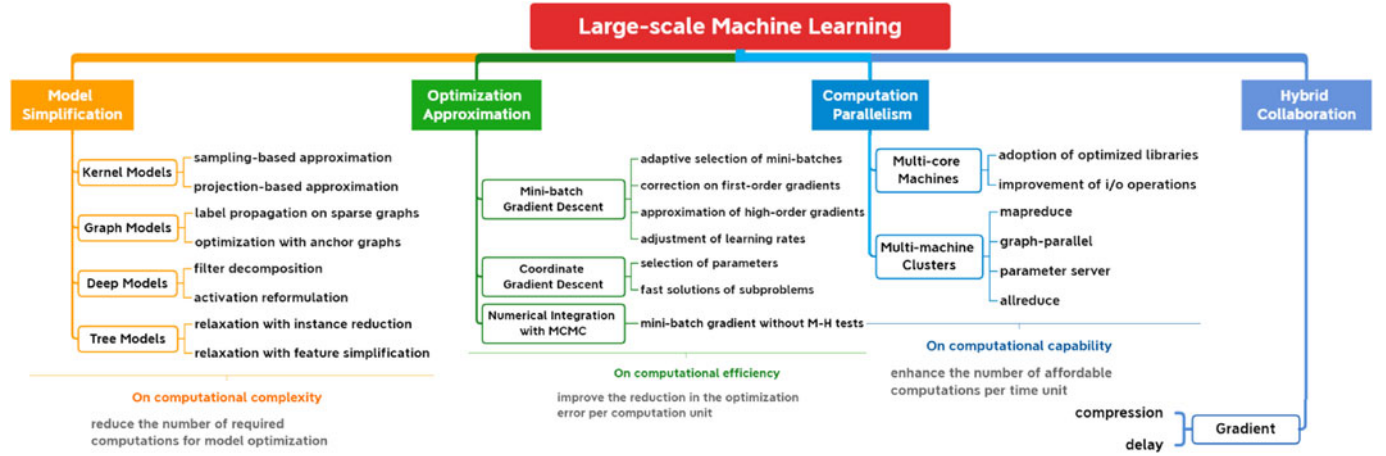
Fig. 1. The framework of Section 3, which shows the ways of scaling up machine learning to large-scale machine learning from three perspectives.

TABLE 1
A Brief Lookup Table for LML Methods Based on Model Simplification

Categories	Strategies	Representative Methods
Kernel-based Models	sampling-based approximation projection-based approximation	uniformly sampling [118], [214], incremental sampling [34], [73]. Gaussian [139], orthonormal transforms [215], ramdom features [134], [161].
Graph-based Models	label propagation on sparse graph optimization with anchor graph	approximate search [110], [227], [232], division and conquer [45], [197]. single layer [132], [201], [228], hierarchical layers [77], [200].
Deep Models	filter decomposition activation reformulation	on channels [97], [117], [181], [189], on spatial fields [182], [190], [218]. at hidden layers [44], [128], [137], [149], at the output layer [143], [144].
Tree-based Models	relaxation with instance reduction relaxation with feature simplification	random sampling [76], sparse-based [52], importance-based [112]. random sampling [36], histogram-based [21], [52], exclusiveness-based [112].

3 REVIEW ON LARGE-SCALE MACHINE LEARNING

In this section, we review LML in detail. Specifically, we present the methods in the above three perspectives in Sections 3.1, 3.2, and 3.3, and discuss their collaboration in Section 3.4. For each part, we categorize the related methods according to targeted scenarios and introduce the methods based on their intrinsic strategies. We also provide experimental evidence to demonstrate the effectiveness of these strategies and summarize their pros and cons. For convenience, Fig. 1 provides a coarse-to-fine overview for the structure of this section.

3.1 Model Simplification

Model simplification methods improve machine learning efficiency from the perspective of computational complexities. To reformulate predictive models and lower computational complexities, researchers introduce reliable domain knowledge for a small $\mathcal{E}_{app}$, such as the structures or distributions of instances and the objectives of learning tasks [24]. According to targeted scenarios, the reviewed scalable predictive models consist of four categories, including kernel-based, graph-based, deep-learning-based, and tree-based. For convenience, a brief overview is provided in Table 1.

3.1.1 For Kernel-Based Models

Kernel methods play a central role in machine learning and have demonstrated huge success in modeling real-world data with highly complex, nonlinear distributions [146]. The key element of these methods is to project instances into a kernel-induced Hilbert space $\phi(\mathbf{x})$, where dot products between instances can be computed equivalently through the kernel evaluation as $K_{ij} = <\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)>$.

Motivation. Given n instances, the computational complexity of constructing a kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ scales as $O(n^2 d)$. Supposing all instances are labeled with a label matrix of $\mathbf{Y} \in \mathbb{R}^{n \times c}$ ($n = n_L$), most kernel-based methods can be solved based on matrix inversion as

$$(\mathbf{K} + \sigma \mathbf{I})^{-1} \mathbf{Y}, \tag{2}$$

which requires a computational complexity of $O(n^3 + n^2 c)$. Examples include the Gaussian process, kernel ridge regression, and least-square support vector machine [81].

To alleviate the above computational burdens for a large n, a powerful solution is performing low-rank approximation based on SPSD sketching models [83] and solving the matrix inversion with Woodbury matrix identity [94]. Specifically, let $\mathbf{S} \in \mathbb{R}^{n \times m}$ with $n \geq m$ indicate the sketching matrix. Take $\mathbf{C} = \mathbf{KS}$, and $\mathbf{W} = \mathbf{S}^T \mathbf{KS}$. Then $\mathbf{CW}^\dagger \mathbf{C}^T$ becomes a low-rank approximation to $\mathbf{K}$ with the rank at most m. Based on the matrix inversion lemma [94], Eq. (2) can be rewritten into $\frac{1}{\sigma}[\mathbf{Y} - \mathbf{C}(\sigma \mathbf{W} + \mathbf{C}^T \mathbf{C})^{-1}(\mathbf{C}^T \mathbf{Y})]$, which reduces the computational complexity to $O(m^3 + nmc)$.

For scaling up kernel-based models without hurting the performance, the choice of $\mathbf{S}$ or the efficient construction of $\mathbf{C}$ becomes crucial. Below we introduce two strategies for low-rank approximation based on sketching matrices.

Sampling-Based Approximation. With this strategy, **S** represents a sparse matrix that contains one nonzero in each column. A direct method is sampling columns of kernel matrices at random with replacement [118], [214], which is equivalent to the Nyström approximation. By assuming that potential clusters are convex, one can select columns corresponding to kmeans centers with the cost of $O(nmdt)$ [228], where t is the number of iterations. To weigh the complexity and the performance, a few incremental sampling methods are proposed. At each iteration, these methods first randomly sample a subset of columns and then pick the column either with the smallest variance of the similarity matrix between the sampled columns and the remaining ones or the lowest sum of squared similarity between selected ones [34], [73], [229]. In general, the computational complexities of these methods scale as $O(nmp)$, where p is the size of the subset. Besides, the structures of clustered blocks can be exploited to reduce storage costs [180].

Projection-Based Approximation. Based on this strategy, **S** is a dense matrix, which consists of random linear combinations of all columns of kernel matrices [90]. For example, one can introduce Gaussian distributions to build a data-independent random projection [139]. Besides, the sketch can be improved with orthonormal columns that span uniformly random subspaces. To do this, we can randomly sample and then rescale the rows of a fixed orthonormal matrix, such as Fourier transform matrices, and Hadamard matrices [215]. These orthonormal sketches can additionally speed up the matrix product to $O(n^2 \log m)$, as opposed to $O(n^2 m)$ required for the same operation with general dense sketches [215]. On the other hand, some methods construct kernel spaces directly by mapping data non-linearly to new low-dimension subspaces [134], [148], [161].

Discussion Based on Experimental Results. Benefit from matrix identity lemma on the low-rank approximation, time costs can be significantly reduced. For example, [210] sped up the matrix inversion nearly 100+ times without scarifying the accuracy for COREL images. Below we discuss the choice of the above methods in practical applications.

First, for sampling-based methods, uniform sampling is the fastest solution. Although its performance is generally worse than others with a small m, it can be mitigated with the increase of m [118]. Besides, the effectiveness and efficiency of incremental sampling are weighed based on q. However, its time cost could be 50 percent lower than kmeans, especially when the selected number of columns was small [73]. Second, for projection-based methods, Gaussian matrices lead to comparable or better performance than orthonormal sketches [83], [215]. Third, projection-based sketches are consistently better than uniformly sampling, while the latter outperforms naive random features [134], [214]. Finally, since sampling-based methods only need to compute the involved similarity, the computation of a full kernel matrix is avoided. However, if data comes in streaming with varying distributions, these methods are no longer suitable [134]. In contrast, random features can be solved in the primal form through fast linear solvers, thereby enabling to handle large-scale data with acceptable performance [161], [186].

3.1.2 For Graph-Based Models

Graph-based methods define a graph where the nodes represent instances in the dataset, and the weighted edges reflect their similarity. For classification tasks, their underlying assumption is label smoothness over the graph [146].

Motivation. Given n instances, a graph is first constructed to estimate the similarity between all instances, measured as $W_{ij}=\text{RBF}(x_i, x_j)$. Then graph-based models constrain the labels of nearby instances to be similar and the predicted labels towards the ground truths. Let **Y** be the label matrix where only n_L rows contain nonzero elements. Consequently, the soft label matrix **F** can be solved by

$$\mathbf{F} = (\mathbf{I} + \alpha \mathbf{L})^{-1}\mathbf{Y}, \tag{3}$$

where $\mathbf{L}=\text{diag}(\mathbf{1}^T\mathbf{W})-\mathbf{W}$ denotes the graph Laplacian matrix. The computational cost of graph models comes from two aspects: the graph construction with the cost of $O(n^2 d)$ and the matrix inversion with the cost of $O(n^3)$. Thus, there are two types of strategies for improving scalability, including label propagation on sparse graphs and optimization with anchor graphs.

Label Propagation on Sparse Graphs. Unlike kernels, most graphs prefer the sparse similarity, which has much less spurious connections between dissimilar points [132]. Thus, we can conduct iterative label propagation to accelerate the spread of labels, represented as $\mathbf{F}^{t+1}=\alpha(\mathbf{W})\mathbf{F}^t+(1-\alpha)\mathbf{Y}$, where $\mathbf{F}^0=\mathbf{Y}$. Let k denote the average nonzero element in each row of **W**. Then the number of necessary computations in each iteration scales as $O(nkC)$, taking up a tiny part of the original amount $O(n^2 C)$. As a result, a huge part of the computational complexity now comes from graph construction, which can be efficiently solved based on approximate sparse graph construction.

These graph construction methods introduce a hierarchical division on datasets to find the neighbors of each instance and estimate their similarity, which reduces the cost to $O(n \log(n)d)$. For example, approximate nearest neighbor search (ANNS) first builds a structured index with all instances [110] and then searches the approximate neighbors of each instance on the obtained index, such as hierarchical trees [200] and hashing tables [199], [227]. To enhance the quality, we can repeat the above procedure to generate multiple basic graphs and combine them to yield a high-quality one [232]. Besides, as sparse graph construction is much easier than the nearest neighbor search, divide-and-conquer methods become more popular. These methods first divide all instances into two or three overlapped subsets and then unite the sub-graphs constructed from these subsets multiple times with neighbor propagation [45], [197].

Optimization With Anchor Graphs. Anchor graphs sample m instances as anchors and measure the similarity between all instances and these anchors as $\mathbf{Z} \in \mathbb{R}^{n \times m}$. Then the labels of instances are inferred from these anchors, leading to a small set of to-be-optimize parameters. They use anchors as transition nodes to build the similarity between instances for label smoothness, and their soft label matrix can be obtained as $\mathbf{F}=\mathbf{Z}(\mathbf{Z}^T\mathbf{Z}+\alpha\tilde{\mathbf{L}})^{-1}\mathbf{Z}^T\mathbf{Y}$, where $\tilde{\mathbf{L}}\in\mathbb{R}^{m \times m}$ is a reduced Laplacian over anchors. Clearly, the cost of graph construction is reduced to $O(nmd)$ or even $O(nd \log(m))$ with ANNS, and the cost of optimization scales as $O(nm^2 + m^3)$.

The original anchor graph models introduce sparse adjacency between instances and anchors [132]. After that, hierarchical anchor graphs propose to retain sparse similarities over all instances while keeping a small number of anchors for label inference [200]. In case that the smallest set of anchors still needs to be large and brings considerable computations, FLAG develops label optimizers for further acceleration [77]. Besides, EAGR proposes to perform label smoothness over anchors with pruned adjacency [201].

Discussion Based on Experimental Results. Now we show the advantages of the two classes of methods. On the one hand, label propagation is more efficient than the original matrix inversion. Without accuracy reduction, it could obtain $10\times$ to $100\times$ acceleration over the matrix inversion [79]. As for approximate graph construction, although hierarchical trees and hashing tables are easy to be used for ANNS [198], they generally ignore the fact that each query must be one of the instances in the graph construction. Thus, they put redundancy efforts on giving a good result for unnecessary future queries. In contrast, divide-and-conquer methods can obtain higher consistency to the exact graphs with less time cost, which was demonstrated on both Caltech101, Imagenet, and TinyImage [45]. On the other hand, although the accuracy of anchor graphs is lightly worse than sparse graphs [132], they are more potent on handling very large datasets. The reason is that, once a set of anchors can be stored in the memory, anchor graphs can be efficiently constructed with the memory cost of $O(md + nk)$ rather than $O(nd)$. For example, with hierarchical anchor graphs, the classification on 8 million instances could be implemented on a personal computer within 2 mins [77], [200].

3.1.3 For Deep Models

Deep models introduce layered architectures for data representation [66]. Instead of fully-connected networks (FCNs) where any pair of input features in each grid-like data are relevant, convolutional neural networks (CNNs) and recurrent neural networks (RNNs) utilize the structures of data by using small-size filters on local receptive fields [117].

Motivation. Take CNN as an example, where each layer of convolution kernels can be viewed as a 4D tensor. Consider input feature maps with the spatial size of $d_I \times d_I$ and the channel number of m_I. A standard convolutional layer on the input can be parameterized by convolution kernels with the size of $d_K \times d_K \times m_I \times m_O$, where d_K is the spatial dimension of the square kernel and m_O is the number of output channels. Suppose the stride equals 1. The computational complexity of this convolutional layer becomes

$$d_I \times d_I \times d_K \times d_K \times m_I \times m_O. \tag{4}$$

Generally, deep models involve a large number of hidden layers. Although the sizes of filters are much smaller than the sizes of input features, CNNs still contain millions of parameters [117], and the convolutions contribute to the bulk of most computations. Besides, deep models contain two types of bounded activation functions, including those used in hidden layers for feature extraction, and the ones used at output layers for probability prediction. The former type is used for all neurons, and the latter type involves normalization operations. When handling big models or a large number of output neurons, e.g., NLP tasks, both can lead to huge computations during back-propagation [141].

To address these issues, we introduce the filter decomposition for simplifying convolution layers and improving the overall speedup. After that, we review efficient activation functions where gradients can be easily estimated.

Filter Decomposition. Filter decomposition is derived by the intuition that there is a significant amount of redundancy in 4D tensors. Thus, some methods reduce computations by gathering information from different channels hierarchically. For example, Alexnet develops group convolutions to avoid the computations between two groups of channels [117]. Mobilenet introduces separable depthwise convolutions [97], which factorize the filters into purely spatial convolutions followed by a pointwise convolution along with the depth variable [181], leading to 70 percent reduction of parameters. Inception v1 in turn first implements the dimensionality reduction on the channels of input features [189], followed by the filters with the original receptive fields. After that, Shufflenet generalizes group convolutions and depthwise convolutions based on the channel shuffle to further reduce the redundancy [231].

One the other hand, some methods reduce the computations based on the hierarchical information integration over the spatial side. For example, VGG replaces one layer of large-size filters with the two layers of smaller-size filters [182], reducing nearly 28 percent computations. Besides, Inception v3 introduces asymmetric convolutions that decompose the convolution with 3×3 filters into two cascaded ones with the spatial dimensions of 1×3 and 3×1 [190], and nearly 33 percent parameters can be saved. Dilated convolutions further support the exponential expansion of receptive fields without the loss of resolution [218]. It shows that a 7×7 receptive field could be explored by two dilated convolutions with 3×3 parameters, reducing 80 percent computations.

Activation Reformulation. The bounded activation functions such as sigmoid and tanh bring expensive exponential operations at each neuron. Besides, both of them face vanishing gradient problems. To address these issues, ReLU releases the bound by simply picking the outputs as max(0, x_{input}) [149]. Since too many activations being below zero will make ReLU neurons inactive, leaklyReLU additionally introduces small gradients over the negative domain [137]. Besides, to improve the training stability by constraining the outputs of activations, a few hard bounded functions upon ReLU were developed, such as bounded ReLU and bounded leakyReLU [128]. Moreover, [44] proposes to replace all the multiplications with adds to reduce the computations.

For probability prediction, the implementation of softmax needs to compute a normalization factor based on all output neurons that could be in millions or billions. To reduce the computations, hierarchical softmax reorganizes output probabilities based on a binary tree. Specifically, each parent node divides the own probability to its children nodes, and each leaf corresponds to a probabilistic output [144]. When an input-output pair is available, one can only maximize the probability of the path in the binary tree, which reduces the computational complexity logarithmically. When Huffman coding is adopted for an optimal hierarchy, the speedup can be further enhanced [143].

Discussion Based on Experimental Results. Gradient descent is widely used for training various networks. Thus, filter decomposition successfully reduces the number of computations by lowering the scale of parameters. For example, Inception v1 was able to become $2\times$ to $3\times$ faster than those without dimensionality reduction on channels [189]. Mobilenet with depthwise convolutions could build a $32\times$ smaller and $27\times$ less compute-intensive networks than VGG16, which still obtained comparable accuracies [97]. Benefit from spatial decomposition, Inception v3 used at least $5\times$ fewer parameters and achieved $6\times$ cheaper computationally [190]. Meanwhile, by improving the activations, the computations can be significantly reduced. For example, by using hierarchical softmax on vocabulary with 10,000 words, [144] sped up network training more than $250\times$. If there was no overhead and no constant term, the speedup could be $750\times$. Besides, the network with ReLU could reach a 25 percent training error rate on CIFAR-10 while being $6\times$ faster than the one with tanh [117].

Above all, it is necessary to combine the above strategies to reduce the computations in iterative optimization. However, to maintain the effectiveness with filter decomposition, it is essential to analyze the complexity of each operation carefully and modify a model progressively [49], [92]. We also note that, model compression also plays a vital role in deep learning. More details can be found in [55], [230].

3.1.4 For Tree-Based Models

Tree-based models build hierarchical trees from a root to leaves by recursively splitting the instances at each node using different decision rules [172], such as Gini index and entropy. After that, random forest (RF) and gradient boosting decision trees (GBDT) introduce ensemble learning to improve the robustness of classifiers and enhance their accuracies. The former trains each tree independently. The latter learns trees sequentially by correcting errors, and often applies second-order approximation for custom losses [75].

Motivation. Given n instances with d features, finding the best split for each node generally leads to the computational complexity of $O(nd)$ by going through all features of each instance. Thus, the datasets with millions of instances and features will lead to huge computational burdens when growing trees. To address this issue, the relaxation of decision rules becomes significantly important, which can be performed from the views of instances and features.

Relaxation With Instance Reduction. Implementing instances sampling while growing trees is the simplest method of relaxing the rules and reducing costs [65], which benefits both trees, RF, and GBDT. Besides, one can take advantage of sparse features to ignore invalid instances when evaluating each split, making the complexity linear to the number of non-missing instances [52]. To improve the representation of sampled instances for GBDT, GOSS prefers the instances with large gradients [112].

Relaxation With Feature Simplification. Feature sampling is an alternative for rule relaxation, which considers only a subset of features at each split node [36]. Besides, histogram-based boosting groups features into a few bins using quantile sketches in either a global or local manner and then perform the splitting based on these bins directly [21], [52].

Since a part of features rarely take nonzero values simultaneously, EFB develops a greedy bundling method to locate these features and then merges them by only extending the range of exclusive features [112].

Discussion Based on Experimental Results. Below we discuss the strengths and weaknesses of the relaxation of decision rules. First, randomly sampling 30 to 20 percent instances could speed up GBDT $3\times$ to $5\times$ while improving its performance [76]. Sparsity-aware splitting also ran $50\times$ faster than the naive version on Allstate-10K [52]. On the other hand, local histogram aggregation resulted in smaller numbers of iterations than the global one by refining the histogram strategy [52]. Meanwhile, EFB additionally merged implicitly exclusive features into much fewer features, leading to better performance than sparsity-aware splitting [112]. Second, although the relaxed rules significantly speed up the training of tree-based models, it may result in extra costs. For example, GOSS requires the cost of computing gradients, and sparsity-aware splitting has to maintain a nonzero data table with additional memory costs.

3.1.5 Summary

We have reviewed various LML methods from the perspective of computational complexities. Now we discuss both the advantages and disadvantages of the above methods.

First, kernel and graph-based models can be scaled up and optimized more efficiently than deep models. Besides, experts can introduce their domain knowledge on input features and develop specific similarities for these two models. Since the sum of positive semidefinite matrices is still positive semidefinite, it is also easier to merge the similarities of different types of features into a single model [211]. Of note, although graphs generally lead to smaller memory costs than kernels, they are only able to handle the data that is satisfied with the cluster assumption [228].

Second, benefit from the hierarchical feature extraction on structural instances, deep models can obtain much higher classification accuracies [17], [171]. However, these methods in turn require huge time costs for training the over-parameterized models. Although filter decomposition methods reduce the computations remarkably, the architectures demand careful designs [190]. Besides, some deep models are mathematically equivalent to kernelized ridge regressions that learn their own kernels from the data [170]. However, they can only build kernels in finite-dimension spaces, and their representations are generally uninterpretable, making predictions hard to be explained [20]. In contrast, tree-based models with hierarchical splitting are more interpretable. Besides, these models can be directly integrated into many other methods for acceleration, such as label trees with binary classifiers [22].

3.2 Optimization Approximation

Optimization approximation scales up machine learning from the perspective of computational efficiency. In each iteration, these methods only compute the gradients over a few instances or parameters to avoid most useless computations [220]. As a result, they increase the reduction in the optimization error per computation unit and obtain an approximate solution with fewer computations. For a small

TABLE 2
A Brief Lookup Table for LML Methods Based on Optimization Approximation

Categories	Strategies	Representative Methods
Mini-batch Gradient Descent	adaptive sampling of mini-batches	dynamic batch sizes [183], proportional instance sampling [13], [86].
	correction of first-order gradients	momentum [152], [160], weighted historical gradients, [64], [108], [174].
	approximation of higher-order gradients	mini-batch Quasi-Newton [38], mini-batch Gauss-Newton [72].
	adjustment of learning rates	refer to iterations [212], to gradients [69], [225], to moments [116], [164].
Coordinate Gradient Descent	selection of parameters	Gauss-Seidel manner [41], [98], [151], Gauss-Southwel rules [67], [153], [178].
	fast solutions of subproblems	using extrapolated points [19], [124], [129], [151], [175], caching [35], [178].
Numerical Integration	mini-batch gradient without M-H tests	1st-order Langevin dynamics [158], [202], 2nd-order Langevin dynamics [51].

$\mathcal{E}_{opt}$, advanced mathematical techniques must be used to guarantee the effectiveness of approximation. According to targeted scenarios, we further categorize them into mini-batch gradient descent, coordinate gradient descent, and numerical integration based on Markov chain Monte Carlo. For convenience, an overview is provided in Table 2.

3.2.1   For Mini-Batch Gradient Descent

The methods of mini-batch gradient descent (MGD)[1] aim to solve the problems with a modest number of parameters but a large number of instances. Compared with stochastic gradient descent, MGD utilizes better gradients estimated over more instances per iteration and generally obtains fast local convergence with lower variances.

Motivation. We first review the basic formulation of MGD. Suppose $\mathcal{X}_t$ refers to a mini-batch of instances with the size of m_t, and Q denotes the objective function built upon the parameter matrix $\mathbf{W}$. Let $\partial Q(\mathbf{W}; \mathbf{x}_i)$ be the stochastic gradient on $\mathbf{x}_i$ and $\mathbf{G}_t = \frac{1}{m_t} \sum_{i \in \mathcal{X}_t} \partial Q(\mathbf{W}^t; \mathbf{x}_i)$ indicates the aggregated stochastic gradient on $\mathcal{X}_t$. Similar to gradient descent, the parameter $\mathbf{W}$ can be updated by

$$\mathbf{W}^{t+1} = \mathbf{W}^t - \eta \mathbf{G}_t, \tag{5}$$

with a learning rate of η. For large-scale datasets, updating parameters based a few instances leads to large variances of gradients and makes optimization unstable. Although we can estimate gradients with large and fixed batch sizes, it remarkably increases per-iteration costs [33].

To address this issue, many LML methods have been proposed by improving gradient information in each iteration with a few extra computations. According to the roles in MGD, we review the methods from four complementary aspects. An illustrative diagram is shown in Fig. 2.

Adaptive Sampling of Mini-Batches. We take account of both the sizes of mini-batches and the sampling of instances. First, since the naive algorithms that partially employ gradient information sacrifice local convergence, we can increase batch sizes gradually via a prescribed sequence [183]. Besides, linear scaling is effective for large mini-batches [87]. Second, as randomly selected instances are independent of optimization, it is worth considering both data distributions and gradient contributions. Specifically, we can enforce the sampling weights of instances to be proportional to the L2 norm of their gradients [13]. Besides, by maintaining a distribution over bins and learning the distribution per t iterations, computations can be further reduced [86].

Correction of First-Order Gradients. Performing the correction on mini-batch gradients provides an alternative to improve the quality of search directions with lower variances, which enables a larger learning rate for accelerations. On the one hand, gradient descent with momentum stores the latest gradients and conducts the next update based on a linear combination of the gradient and previous updates [160]. Gradients descent with Nesterov momentum first performs a simple step towards the direction of the previous gradient and then estimates the gradient based on this lookahead position [152]. On the other hand, SAG utilizes the average of its gradients over time to reduce the variances of current gradients [174], and SVRG develops a memory-efficient version which only needs to reserve the scalars to constrict the gradients at subsequent iterations [108].

Approximation of Higher-Order Gradients. When the condition numbers of objective functions become larger, the optimization can be extremely hard owing to ill-conditioning [61]. To solve this issue, MGD methods thus introduce the approximation of second-order information with successive re-scaling [122], [135], [187]. Similar to inexact Newton algorithms, a basic solution is to employ conjugate gradient algorithms to estimate Hessian matrices. However, as the mini-batch size is much smaller, these algorithms implemented in a nearly stochastic manner can lead to very noisy results. To this end, mini-batch-based Quasi-Newton (MQN) and Gauss-Newton algorithms (MGN) are proposed, which improve the approximation by only using first-order information. In particular, MQN introduces online L-BFGS to approximate the inverse of Hessian based on the latest parameters and a few gradients at previous mini-batches [38]. MGN builds the Hessian approximation based on the Jacobian of the predictive function inside quadratic objective functions [72]. When logarithmic losses are used in probability estimation, it enables faster
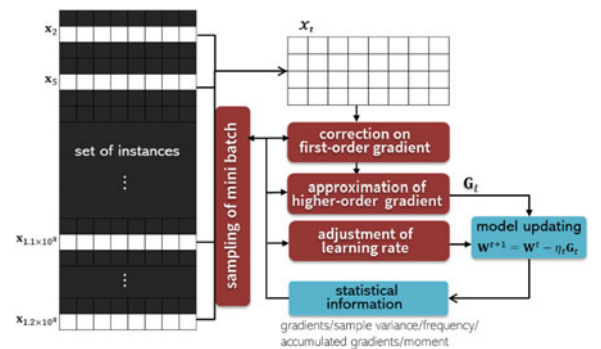

Fig. 2. The diagram of mini-batch gradient descent, where the steps in red are key points for improving the computational efficiency.

1. MGD in this paper is equal to mini-batch SGD in other papers.

implementation without the explicit Jacobian matrices. It is worthwhile noting that we usually can offset the costs of these approximations when the size of mini-batches is not too small [33].

Adjustment of Learning Rates. The learning rate plays an important role in the convergence [169]. Specifically, a small rate may slow down the convergence, while a large rate can hinder convergence and cause the objective function to fluctuate around its minimum. Although the rate decayed by the number of iterations can alleviate this issue [212], recent studies propose to adjust it more carefully according to optimization processes. For example, Adagrad prefers smaller rates for the parameters associated with frequently occurring dimensions and larger rates for the ones associated with infrequent dimensions [69]. Besides, Adadelta takes account of the decaying average over past squared gradients [225]. The motivation is that, by restricting the length of accumulated gradients to some fixed sizes, we can reduce the aggressive when decreasing learning rates [23]. Meanwhile, learning rates can also be updated based on the second moment. For example, Adam prefers flat minima in error surfaces [116]. Besides, Nadam and AMSGrad introduce Nesterov-accelerated gradients and the maximum of past squared gradients into Adam, respectively [68], [164].

Discussion Based on Experimental Results. Now we discuss the effectiveness of the above strategies. First, adaptive sampling of mini-batch can clearly improve computational efficiency. By doubling batch sizes during training, [183] reduced the time cost from 45 mins to 30 mins on ImageNet. Besides, [86] introduced adaptive sampling and only used 30 percent epochs to obtain the same accuracy of uniform sampling. Since updating the statistic information does not need to be frequent, its extra computational cost is insignificant. Second, by correcting gradients for lower variance, SVRG converged faster than the MGD using learning rate scheduling [108]. Third, by approximating the second-order information, [135] achieved $6\times$ to $35\times$ faster when solving eigensystems. MQN also reached a lower objective value with competitive computing time to the original MGD [38]. Finally, by taking the frequency of parameters and the decaying of gradients into account for learning rates, Adagrad and Adadelta could only use 20 percent epochs to achieve the same result of normal optimizers [225]. Based on adaptive moment estimation, Adam further reduced more than 50 percent computations [116]. A comprehensive comparison of learning rate schemes can be found in [113]. Of note, since these strategies play different roles in optimization, one can apply all for further acceleration [80].

3.2.2 For Coordinate Gradient Descent

The LML methods built upon coordinate gradient descent (CGD) aim at addressing the problems with a modest number of instances in a high dimension. These problems frequently arise in areas like natural language processing [220] and recommender systems [18], [93].

Motivation. We first present the basic formulation of CGD [204]. Let $Q(\mathbf{W})$ denote the objective function with the parameter matrix of $\mathbf{W}$, and $\mathcal{I}_t$ contain the indexes of the selected parameters in the tth iteration. As only a few parameters need to be updated, CGD can take all instances
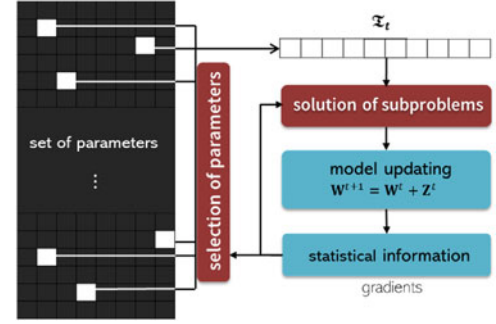

Fig. 3. The diagram of coordinate gradient descent, where the steps in red are key points for improving computational efficiency.

into account and optimize the selected parameters to optimal solutions in each iteration. Specifically, let $\mathbf{W} + \mathbf{Z}^t$ refer to the expected solution on selected parameters, namely $\mathbf{Z}^t$ only contains non-zero values at the indexes in $\mathcal{I}_t$. In each iteration, CGD minimizes the following subproblem

$$g(\mathbf{Z}^t) = Q(\mathbf{W} + \mathbf{Z}^t) - Q(\mathbf{W}), \qquad (6)$$

where $Z_{i,j}^t = 0, \forall \{i, j\} \notin \mathcal{I}_t$. Although the scales of the reduced subproblems are small and many available solvers can be directly used for the optimization [11], CGD can still lead to an expensive time cost for exact solutions during iterative optimization. Besides, updating each parameter with the same number of iterations causes a massive amount of redundant computations.

To address these issues, we review LML methods from two aspects: the selection of parameters and the fast solution of subproblems. For convenience, an illustrative diagram showing their roles is displayed in Fig. 3.

Selection of Parameters. For convergence with a smaller number of iterations, the indexes must be selected carefully. Typically, the selection of these parameters follows a Gauss-Seidel manner [217], namely, each parameter is updated at least once within a fixed number of consecutive iterations. As features may be correlated, performing the traversal with a random order of parameters in each iteration has empirically shown the ability of acceleration [41], [98], [151]. On the other hand, Gauss-Southwell (GS) rules propose to take advantage of gradient information for the selection, which can be regarded as performing the steepest descent [178]. To avoid the expensive cost in GS rules, we can connect them to the nearest neighbor search and introduce a tree structure to approximate the rules [67], [153]. Besides, by introducing the quadratic approximation on objective functions and performing the diagonal approximation on Hessian matrices, the selection of multiple indexes can be reduced to separable problems [221].

Fast Solutions of Subproblems. Similar to accelerated gradient descent, extrapolation steps can be employed for accelerated coordinate descent methods [124], [151]. Suppose $Q(\mathbf{W}) = p(\mathbf{W}) + q(\mathbf{W})$, where $p(\mathbf{W})$ and $q(\mathbf{W})$ are smooth and nonsmooth, respectively. Accelerated proximal gradient (APG) first replaces $p(\mathbf{W})$ with the first-order approximation regularized by a trust region penalty and then uses the information at an extrapolated point to update the next iterate [129]. With appropriate positive weights, it can improve the convergence remarkably while remaining almost the

same per-iteration complexity to the proximal gradient [19], [175]. To address nonconvex problems, the monotone APG method proposed in [125] introduces sufficient descent conditions with a line search. It reduces the average number of proximal mappings in each iteration and speeds up the convergence. Moreover, if both $p(\mathbf{W})$ and $q(\mathbf{W})$ are nonsmooth, ADMM introduces Douglas-Rachford splitting to solve the problem, which is extremely useful when the proximal operators can be evaluated efficiently [35]. Of note, similar to standard iterative optimization, common heuristics can be used for speedup, such as caching variable-dependent eigendecomposition [35] and pre-computation of non-variable quantities [178].

Discussion Based on Experimental Results. Now we discuss the efficiency of the above strategies for solving primal and dual problems. On the one hand, when optimizing L2-SVM with correlated features, the CGD based on random permutation could obtain the same relative difference to the optimum with $2\times$ to $8\times$ fewer training costs [41]. Besides, due to the sparsity of the solutions of L1 and L2 regularized least squares problems, CGD based on GS rules not only outperformed random and cyclic selections with $5\times$ to $10\times$ fewer epochs for faster convergence but also used $2\times$ less running time including the estimation of statistical information [153]. On the other hand, for solving lasso, APG only required 20 percent time of the basic proximal gradient, and ADMM further saved 50 percent time costs [155]. Besides, when solving linear inverse problems like image deblurring, the result of the proximal algorithm ISTA after 10,000 iterations could be obtained by its accelerated version with 275 iterations [19], [60]. For nonconvex logistic regression with capped L1 penalties, nonmonotone APG with released conditions obtained a higher accuracy with 75 percent fewer costs than monotone APG [125]. In addition, by caching eigendecomposition, ADMM solved problems $20\times$ faster than computing it repeatedly in each iteration [35].

3.2.3 For Numerical Integration With MCMC

The methods based on Markov chain Monte Carlo (MCMC) are widely used in Bayesian posterior inference on $p(\mathbf{W}|\mathcal{X})$ [27], such as M-H and Gibbs sampling [57], [89]. Unlike the gradient-descent optimization with multi-dimensional integrals on $p(\mathcal{X})=\int_{\mathbf{W}} p(\mathcal{X}|\mathbf{W})p(\mathbf{W})d\mathbf{W}$, they introduce numerical approximations by recording samples from the chain and avoid the huge time costs in high-dimensional models. Specifically, they first generate candidate samples by picking from distributions and then accept or reject them based on the corresponding acceptance ratios. To enhance acceptance probabilities and improve efficiency, more methods introduce the gradient of the density of target distributions to generate samples at high-density regions.

Motivation. Given a dataset of n instances, most gradient-based sampling methods [150] introduce the Langevin dynamic and update parameters based on both the gradient and the Gaussian noise as

$$\mathbf{W}_t + \frac{\epsilon_t}{2} \{ \partial_{\mathbf{W}_t} \log p(\mathbf{W}_t) + \sum_{x\in\mathcal{X}} \partial_{\mathbf{W}_t} \log p(x|\mathbf{W}_t)\} + v_t, \qquad (7)$$

where the step size ϵ_t and the variances of noise $v_t \sim N(0, \epsilon_t)$ are balanced. As a result, their trajectories of parameters can converge to the full posterior distribution rather than just the maximum a posteriori mode. However, the calculation of gradients per interaction faces the computation over the whole dataset, and expensive M-H accept/reject tests are required to correct the discretization error. Although Gibbs sampling is free from M-H tests, it is still limited to the exact sampling from conditional posterior distributions. Thus, many stochastic gradient MCMC methods based on mini-batches are proposed to address the above issues.

Mini-Batch Gradient Without M-H Tests. SGLD first introduces the scaled gradients that estimated from mini-batch instances to approximate the gradient of the log-likelihood [202]. Besides, it discards the M-H test and accepts all the generated samples, since its discretization error disappears when $\epsilon_t \rightarrow 0$. To reduce the number of iterations, SGFS prefers samples from approximated Gaussian distributions for large step sizes and switches to samples from the non-Gaussian approximation of posterior distributions for small step sizes [10]. Meanwhile, SGHMC introduces a second-order Langevin dynamics with a friction term to improve the exploration of distant space [51]. In addition, SGRLD extends SGLD for models on probability simplices by using Riemannian geometry of parameter spaces [158], and SGRHMC further takes advantages of both the momentum of SGHMC and the geometry of SGLRD [136].

Discussion Based on Experimental Results. Now we show the effectiveness of the above MCMC methods for Bayesian posterior inference. First, these methods enable efficient inference for large-scale datasets. For example, SGRLD could perform LDA on Wikipedia corpus while Gibbs sampling was not able to run [158]. Second, different mini-batch gradient variants can inherit the properties of their original versions. For example, SGHMC could generate high-quality distant samples and reduce more than 50 percent iterations than SGLD for training Bayesian neural networks [51]. Besides, SGRHMC resulted in lower perplexities of LDA than both SGRLD and SGHMC while remaining similar costs [136].

3.2.4 Summary

To enhance the computational efficiency and remain reliable solutions, the above methods prefer the computations that produce higher reduction in optimization errors. First, both MGD and CGD methods take the importance of involved instances or parameters into account and introduce the careful selections for reducing the time cost. Meanwhile, both can take advantage of Nesterov's extrapolation steps and tune learning rates with accumulated gradients or use a line search for faster convergence. Second, CGD is more suitable for optimizing models with a large number of parameters, especially for linear models where data is stored with inverted indexes. In contrast, if the number of features is much smaller than the number of instances, one should solve problems based on MGD. Third, for handling large models with a massive amount of instances, it is natural to combine two strategies together, i.e., updating a subset of parameters with a few instances in each iteration. For example, a recently proposed method called mini-batch randomized block coordinate descent estimates the partial gradient of selected parameters based on a random mini-batch in each iteration [235]. Finally, mini-batch gradient MCMC algorithms improve qualities of samples and accelerate

TABLE 3
A Brief Lookup Table for LML Methods Based on Computation Parallelism

Categories	Strategies	Representative Methods
Multi-Core Machines	adoption of optimized libraries	acceleration with CPUs [56], [58], [99], [185], with GPUs [5], [25], [103], [156].
	improvement of i/o accesses	balance memory and disks [119], [168], [238], balance CPU and GPU memory [165].
	MapReduce abstraction	MapReduce [167], [179], [29], spark [223], supporting libraries [28], [142], [224].
Multi-Machine Clusters	graph-parallel abstraction	bulk synchronous [138], asynchronous [133], GAS decomposition [48], [84].
	parameter server abstraction	distributed networks [53], [62], gradient boosting [104], general-purpose system [209].
	allreduce abstraction	tree-based [52], butterfly-based [3], [112], ring-based [2], [4], [102], [177], hybrid [87].

Bayesian inference. Besides, these algorithms generally resemble MGD from different views, such as additive noise and momentum. However, since samples from the dynamics of native stochastic variants may not always converge to the desired distribution, some necessary modification terms must be introduced to substitute the correction steps [51], [136]. Besides, the theory on how these additions differentiate a Bayesian algorithm from its optimization remains to be studied [43].

Meanwhile, a significant part of the above LML methods focuses on handling convex problems, while only a few methods can address nonconvex problems. Besides, most methods solve target problems directly. Instead, one may seek the transformed forms of their problems and develop more appropriate algorithms by referring related work in other domains. Details will be discussed in Section 4.1.2.

3.3 Computation Parallelism

Computation parallelism reduces practical time costs from the perspective of computational capabilities. The motivation is, mutually-independent subtasks can be processed simultaneously over multiple computing devices. For this purpose, many parallel processing systems have been developed. On the one hand, these systems increase the number of computations per time unit based on powerful hardware; on the other hand, they introduce advanced software to schedule hardware efficiently. According to the scenarios of hardware, we review the systems from two categories, as shown in Table 3, including the systems with multi-core machines and those with multi-machine clusters. Below we introduce their main characteristics, aiming at utilizing their developments to benefit efficient machine learning methods.

3.3.1 For Multi-Core Machines

The systems here take advantage of multi-core machines for parallelism. Specifically, all cores can access common memories, and each of them execute subtasks independently.

Motivation. To enhance the computational ability of a single machine, recent processing systems increase the number of parallelable instructions by introducing multiple processors into the machine, such as multiple-core CPUs and GPUs. Although their theoretical capabilities could be improved almost linearly with respect to the number of cores, the practical speedup is much lower owing to the overhead. To improve the effectiveness, users have to handle tedious works such as the scheduling of cores and memory.

To address this issue, we review representative works that provide the interfaces with different levels of simplicity

and flexibility. Consequently, one can take their advantages and smoothly implement LML models in a single machine.

Adoption of Highly-Optimized Libraries. When the memory is sufficient, we can directly utilize multi-core CPUs or GPUs to perform the optimization with available highly-optimized libraries, e.g., Mkl for Intel CPUs and Cuda for Nvidia GPUs. On the one hand, CPUs can use relatively cheap RAMs and handle complex operations in parallel. For example, Liblinear and FPSG provide the efficient solutions to L1-regularized linear classification and parallel matrix factorization [56], [58], respectively. For general purposes, Shogun is compatible with support vector machines and multiple kernel learning [185]. Shark additionally supports evolutionary algorithms for multi-objective optimization [99]. Of note, lock-less asynchronous parallel (ASP) can further take advantage of CPU cores and reduce the overhead for parallelizing MGD [163]. On the other hand, GPUs built upon thousands of stream cores are better at handling simple computations in parallel. In particular, deep learning systems are highly benefited from GPU-accelerated tensor computations and generally provide automatic differentiation, such as Theano [25], Caffe [103], MXNet [53], TensorFlow [5] and PyTorch [156]. For example, TensorFlow optimizes execution phase based on the global information of programs and achieves the high utilization of GPUs [5]. PyTorch introduces easy-to-use dynamic dataflow graphs and offers the interface to wrap any module to be parallelized over batch dimension [156].

Improvement of I/O Accesses. When the memory cannot handle a LML model at once, we have to break it into multiple parts, where each part is computed in parallel and different parts are implemented sequentially. In this case, the I/O accesses result in an in-negligible overhead for overall computations, and the effective utilization of memory and extra storage becomes essential. For example, to handle a large graph with n_e edges, GraphChi divides its vertices into many intervals, where each is associated with a shard of ordered edges [119]. When performing updating from an interval to the next, it slides a window over each of the shards for reading and writing, implementing the asynchronous model with $O(2n_e)$ accesses. To avoid expensive pre-sorting of edges, X-Stream instead proposes an edge-centric implementation in synchronous by streaming unordered edge lists [168]. However, it needs to read edges and generate updates in the scatter phase and read updates in the gather phase, leading to $O(3n_e)$ accesses. Recently, Grid-Graph introduces a two-level partitioning, which streams every edge and applies the generated update instantly [238]. It only requires one read pass over edges and several read/write passes over vertices, leading to nearly $O(n_e)$ accesses.
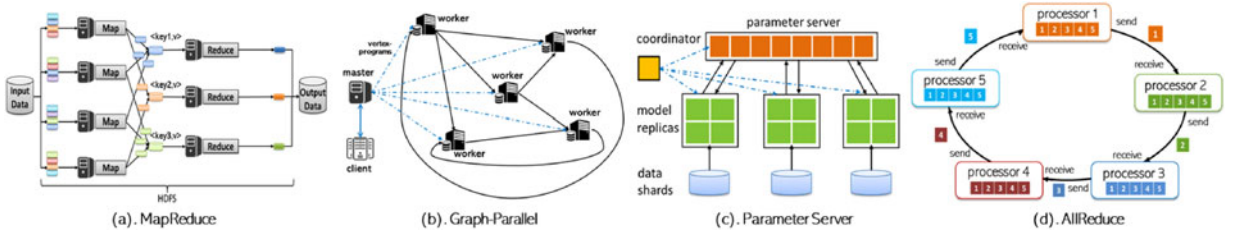
Fig. 4. The toy examples of distributed processing systems. (a) the structure of a MapReduce system [63], (b) the structure of graph-parallel systems [138], (c) a sketch of parameter-server-based cluster [126], and (d) a step of data communication of the ring-based allreduce abstraction [177].

Considering that the memory of GPU is expensive and intermediate feature maps account for the majority of usage, vDNN moves the intermediate data between GPU and CPU, which can train larger networks beyond the limits of the GPU's memory [165].

Discussion Based on Experiments. The systems based on multi-core machines have shown their power to accelerate the implementation of LML. In particular, when the memory is sufficient, FPSG could speed up matrix factorization nearly $7\times$ with a 12-core CPU [58]. Besides, Theano was able to achieve $5\times$ faster for training CNNs on GPUs rather than CPUs and $6\times$ faster for optimizing DBNs [25]. Even with limited memory, by scheduling I/O accesses efficiently, we can still utilize the parallelism and train large models within acceptable time costs. For example, GraphChi used a mac mini and successfully solved large Pagerank problems reported by distributed systems [119]. For the same number of iterations, GridGraph only required fewer than 50 percent time costs of others [238]. Besides, with the support of CPU memory, vDNN could optimize a deep network with the memory requirement of 67 GB based on a 12GB GPU [165].

3.3.2 For Multi-Machine Clusters

With the explosion of data size, processing systems based on distributed clusters have attracted increasing attention. These systems utilize local area networks to integrate a set of machines, where each runs its own tasks.

Motivation. Distributed processing systems introduce parallelism at two levels, including a number of nodes in a cluster and multiple cores within the individual node. One can also construct huge models for complex tasks if model parallelism is applied. Nevertheless, the main challenge of applying a cluster is managing (heterogeneous) machines, such as data communication across the nodes. These tedious works may lead to low resource utilization.

Below we review typical distributed systems to simplify the developments of parallelism [39]. We present the introduction according to their top-level topology configurations.

MapReduce Abstraction. The MapReduce abstraction divides computations among multiple machines [63], where each works with a part of tasks in parallel with its locality-distributed data. As we can see from Fig. 4a, MapReduce systems reformulate complex models into a series of simple Map and Reduce subtasks. Specifically, Map transforms its local data into a set of intermediate key/value pairs, and Reduce merges all intermediate values associated with the same intermediate key. The systems take care of the details of data partitioning, execution scheduling, and communication managing. As a result, the methods that meet this abstraction can take advantage of locality-distributed data and be executed on clusters smoothly.

Hadoop MapReduce is the first open-source implementation, which introduces HDFS files to store the data over disks [179]. Its flexibility allows new data-analysis software to either complement or replace its original elements, such as YARN [195]. Besides, to support iterative machine learning methods, Iterative MapReduce introduces a loop operator as a fundamental extension [167], and Hybrid MapReduce presents a cost-based optimization framework to combine both the task and data parallelism [29]. On the other hand, Spark employs the immutable distributed collection of objects RDD as its architectural foundation [222] and distributes the data over a cluster in the memory to speed up iterative computations [223]. Besides, a stack of libraries are developed to further simplify its programming. For example, MLlib provides a variety of linear algebra and optimization primitives [142] and Spark Streaming eases the streaming learning [224]. Recently, SystemML enables Spark to compile sophisticated machine learning methods into efficient execution plans automatically [28].

Graph-Parallel Abstraction. The graph-parallel abstraction shown in Fig. 4b exploits the structure of sparse graphs to improve communication between different machines [111]. Specifically, it consists of a sparse graph and a vertex-program. The former is partitioned into different subsets of vertexes over multiple machines. The latter is executed in parallel on each vertex and can interact with the neighboring vertexes with the same edge. Compared with MapReduce, these systems constrain the interaction based on sparse adjacency and reduces the cost of communication.

Pregel is a fundamental graph-parallel system running in bulk synchronous parallel (BSP) [138]. It consists of a sequence of supersteps, where messages sent during one superstep are guaranteed to be delivered at the beginning of the next superstep. However, since the slowest machine determines the runtime in each iteration, Pregel can result in idling problems. GraphLab thus introduces ASP by releasing the constraints of supersteps [133], which updates the vertexes using the most recent values. Moreover, PowerGraph introduces a gather, apply, and scatter (GAS) decomposition to factor vertex-programs for power-law graphs [84] and enables the computations in both BSP and ASP. Recently, PowerLyra uses centralized computation for low-degree vertices to avoid frequent communications and follows GAS to distribute the computation for high-degree vertices [48]. Similarly, it supports both BSP and ASP.

Parameter Server Abstraction. The parameter server abstraction distributes data and workloads over worker nodes and maintains distributed shared memory for parameters on server nodes [126]. An example is shown in Fig. 4c. This

abstraction enables an easy-to-use shared interface for I/O access to models, namely, workers can update and retrieve the different parts of parameters as needed.

DistBelief first utilizes computing clusters with the parameter server [62], where the model replicas in workers asynchronously fetch parameters and push gradients with the parameter server. It thus can train deep networks with billions of parameters using thousands of CPU cores. After that, MXNet adopts a two-level structure to reduce the bandwidth requirement [53], where one level of servers manages the data synchronization within a single machine, and the other level of servers manages intermachine synchronization. Moreover, the naive distributed Tensorflow introduces dataflow with the mutable state to mimic the functionality of a parameter server, which provides additional flexibility on optimization algorithms and consistency schemes [5]. Besides, to handle GBDT with high dimensions, DimBoost transforms each local histogram to a low-precision histogram before sending it to the parameter server [104]. Once these local histograms are merged on the servers, a task scheduler assigns active tree nodes among all workers to calculate the best split for each tree node. Meanwhile, distributed MCMC methods are also developed for large-scale Bayesian matrix factorization [9], where the parameter server updates its global copy with new sub-parameter states from workers. In addition, as a general-purpose system, Petuum is also benefited from this abstraction [209] and simplifies the distributed development of various scalable machine learning models.

Allreduce Abstraction. The allreduce abstraction reduces the target arrays in all m machines to a single array and returns the resultant array to all machines. It retains the convergence of gradient descent in BSP and can be divided into tree-based [50], butterfly-based [206], and ring-based [1].

XGBoost introduces tree-allreduce by organizing all the workers as a binomial tree [52]. Its aggregation steps follow a bottom-to-up scheme starting from the leaves and ending at the root. However, these steps cannot overlap in its implementation. LightGBM thus introduces butterfly-allreduce with a recursive halving strategy [3], [112]. Specifically, at the kth step, each worker exchanges $\frac{n}{2k}$ data with a worker that is $\frac{m}{2k}$ distance away. This process iterates until the nearest workers exchange their data.

Recently, ring-allreduce has attracted more attention by employing bandwidth-optimal algorithms to reduce communication overhead [157]. Fig. 4d displays an illustrative example, where m machines communicate with their neighbors $2 \times (m\text{-}1)$ times by sending and receiving parameter buffers. Specifically, in the first (second) $m\text{-}1$ iterations, each machine receives a buffer and adds it to (substitutes it for) its own value at the corresponding location. These new values will be sent in the next iteration. To ease the use of ring-allreduce on GPUs, Nvidia develops Nccl for its devices [102]. After that, Horovod integrates Nccl into deep learning and allows users to reduce the modification of their single-GPU programs for the distributed implementation [177]. Pytorch utilizes Nccl to operate heavily-parallel programs on independent GPUs [4]. Recently, Tensorflow also provides two different experimental implementations for ring-allreduce [2]. In particular, Caffe2 introduces butterfly-allreduce for inter-machine communication while using ring-allreduce for local GPUs within each machine [87].

Discussion Based on Experiments. Effective distributed systems not only allow us to handle larger-scale datasets and models, but also make data communication more efficient. However, different abstractions show their characteristics in specific tasks. For example, for PageRank, Hadoop and Spark took 198 and 97 seconds for each iteration, while PowerGraph only used 3.6 seconds and was significantly faster [84]. Based on the differentiated partition for low-degree and high-degree vertices, PowerLyra further outperformed PowerGraph with a speedup ranging from $1.4\times$ to $2\times$. For matrix factorization, Petuum was faster than Spark and GraphLab [209]. Besides, when the number of workers was not a power of two, DimBoost could outperform XGBoost and LightGBM significantly by merging parameters with data sparsity [104]. Based on ring-allreduce, Horovod makes TensorFlow more scalable on a large number of GPUs. In particular, it could half the time cost of naive distributed Tensorflow when training CNNs with 128 GPUs [177].

3.3.3 Summary

Now we provide rough guidance of parallelism strategies. First, CPUs are optimized for handling the models with complex sequential operations. For example, by introducing locality sensitive hashing for the sparse selection of activate neurons, the training of FCNs on CPUs was $3.5\times$ faster than the best available GPUs [42]. On the contrary, GPUs could enhance the computational capability greatly for the intensive computational models with huge numbers of dense matrix and vector operations [74]. Second, although machine learning methods may be compatible with multiple abstractions [154], it is crucial to choose a proper abstraction for a specific setting. In general, graph-parallel and ring-allreduce are more suitable for graph models and deep learning models, respectively. Third, we can ease the development of higher-level systems by leveraging the existing systems. For example, built upon YARN, Angel develops a parameter server system for general-purpose LML [106], which supports both model and data parallelism. Similarly, Glint applies Spark for convenient data processing in memory and introduces an asynchronous parameter server for large topic models [100]. Moreover, PS2 launches Spark and parameter servers as two separate applications without hacking Spark [233], which makes it compatible with any version of Spark. GraphX also introduces the graph-parallel to accelerate graph-based models on Spark [85].

However, the best choice of practical developments varies for the size of datasets and need further studies. For example, for training SVMs, the efficiency of GPUs only became higher than CPUs when the sizes of datasets increased to some degree [127]. Besides, owing to the overhead, the return of distributed systems becomes lower with the increasing number of processors, and we have to control the return on investment regardless of abstractions. For example, although larger CNNs was supposed to benefit more with multiple machines, a model with 1.7 billion parameters only had $12\times$ speedup using 81 machines [62].

3.4 Hybrid Collaboration

The methods in the above sections scale up LML from the perspective of computational complexities, computational

TABLE 4
A Brief Lookup Table for LML Methods Based on Hybrid Collaboration

Categories	Strategies	Representative Methods
Gradient	compression	quantize component values [14], [105], [176], [203], [226], remove redundancy gradients [130].
	delay	limited delay with stale synchronous [7], [101], [121], [166], [234], [236], infinite delay [240].

efficiency, and computational capabilities, respectively. Thus, they are independent of each other, and any partial enhancement enables the overall improvement.

Motivation. In general, we can enhance machine learning efficiency by jointly using multiple strategies directly [40], [71], [88]. The most typical work is deep learning, which has been undergoing unprecedented development over the past decades [47], [54], [123]. The key reasons that facilitate its rapid development are detailed as follows. First, deep learning methods introduce prior knowledge such as filter decomposition to simplify predictive models with fewer parameters [216]. Second, owing to its mini-batch-based optimization, various advanced MGD algorithms can be employed to train neural networks, leading to faster converges [116]. Third, since GPUs are paired with a highly-optimized implementation of 2D convolutions and well-suited to cross-GPU parallelization [117], it is efficient for researchers to debug their advanced networks [219].

However, the above perspectives can be further improved for synergy effects. In particular, for simplified models, we can modify optimization algorithms to improve parallelism. We divided them into two strategies as listed in Table 4, including gradient compression and gradient delay.

Gradient Compression. To further accelerate the data communication in distributed optimization, many compression methods have been proposed, where only a small number of data is communicated across machines. For example, 1-bit MGD quantizes each component of gradients aggressively to one bit per value and feeds back the quantization error across mini-batches [176]. On the contrary, QSGD directly quantizes the components by randomized rounding to a discrete set of values while preserving the expectation with minimal variance [14]. As a result, it makes a trade-off between the number of bits communicated per iteration and the added variance. To exploit valuable gradients with outliers, TernGrad introduces layer-wise ternarizing and gradient clipping [203]. Meanwhile, DGC only transmits gradients larger than a threshold for sparsification while accumulating the rest of gradients locally [130]. In particular, when gradients are sparse and their distribution is nonuniform, the nonzero elements in a gradient are generally stored as key-value pairs to save space. SketchML thus takes the distribution into account and uses a quantile sketch to summarize gradient values into several buckets for encoding [105]. Besides, it stores the keys with a delta format and transfers them with fewer bytes. Furthermore, ZipML proposes to optimize the compression on gradients, models, and instances and then perform an end-to-end low-precision learning [226]. Of note, the above compression methods are compatible with various parallel systems, such as multiple GPUs [14], [176] and parameter servers [105], [130], [203].

Gradient Delay. ASP-based MGD and CGD algorithms [131], [163] minimize the overhead of locking and alleviate the idling issues. However, they remain huge costs of data communication. The stale synchronous parallel (SSP) algorithms with gradient delays thus get more attention [96]. For example, [121] orders CPU cores where each updates variables in a round-robin fashion. Its delay reduces the cost of reading parameters of the latest models. [7] introduces tree-allreduce in distributed settings, where each parent averages the gradients of the children nodes from the previous round with its own gradient, and then passes the result back up the tree. In addition, CoCoA and Hydra use workers to perform some steps of optimization with their local instances and variables in each iteration [101], [166], respectively. EASGD introduces an elastic force between central and local models for more exploration to avoid many local optima [234]. Moreover, [240] proposes to solve subproblems exactly on each machine without communication between machines before the end, namely infinite delay. Since the delayed gradient is just a zero-order approximation of the correct version, DCASGD leverages the Taylor expansion of gradient functions and the approximation of the Hessian matrix of the loss function to compensate for delay [236]. On the other hand, delayed gradients can also be utilized to improve the convergence, which in turn reduces the communication. For example, ECQSGD and DoubleSqueeze accumulate all the previous quantization errors rather than only the last one for error feedback [191], [205]. Besides, [140] introduces the delayed gradients to chooses adaptive learning rates in parallel settings.

Discussion Based on Experiments. Now we give a brief discussion on the methods of hybrid collaboration. First, gradient compression can reduce not only the time cost per iteration but also the overall cost. For example, compared with full 32-bit gradients, the training of 16-GPU AlexNet with 4-bit QSGD led to more than $4\times$ speedup on communications, and $2.5\times$ speedup on overall costs for the same accuracy [14]. In particular, for linear models, SketchML could accelerate the original optimizer and achieve more than $4\times$ improvements [105]. Of note, models with larger communication-to-computation ratios and networks with smaller bandwidth can benefit more from compression [203]. Second, gradient delay also reduces the amount of data communication and remarkably increases the proportion of time workers spend on the computation. For example, based on local updates, CoCoA was able to converge to a more accurate solution with $25\times$ faster than the best non-locally updating competitor [101]. Besides, the reduction of communication cost could be nearly linear with respect to the delay under a large number of workers [234].

4 DISCUSSIONS

Existing LML methods have established a solid foundation for big data analysis. Below we outline some promising extension directions and discuss important open issues.

4.1 Extension Directions

4.1.1 For Model Simplification

The methods of model simplification reduce computational complexities with adequate prior knowledge. Therefore, we may further explore the distributions and structures of instances to enhance the scalability of predictive models.

Explore Distributions of Data. The low-rank approximation for kernels assumes that mapped instances in RKHS nearly lie on a manifold of a much lower dimension, and their inner products can be estimated on the low-dimension space. Graph-based methods are also scaled up based on the assumption that nearby points are more likely to share labels. Thus, to make use of large-scale unstructured data, the underlying global and local data distributions must be considered [82]. Based on this motivation, HSE and AER build hierarchical indexes on unlabeled instances for coarse-to-fine query selection [15], [78]. As a result, although the cost of the quality estimation of each candidate is not reduced, they lower computational complexities significantly by decreasing the number of candidates.

Exploit Structures Within Instances. Various types of convolutions based on filter decomposition have been proposed to reduce the sizes of deep models. These works demonstrate that to make receptive fields more effectively, a critical direction is to take the features with multi-scale invariance into account and make use of the structural information of instances. Following this motivation, DCNs thus develop deformable convolutions to model larger transformations efficiently [59], [239], which only add a few parameters.

Analyze Objective Tasks. A more in-depth analysis of objective tasks provides another powerful way for model simplification. For example, inspired by the coarse-to-fine categorization of visual scenes in scene-selective cortex [147], hierarchical convolutional networks are developed for image classification [213]. These networks decrease the number of parameters and correspondingly reduce the involved computations per iteration. In addition, distributing parameters and computations according to the frequencies of objective predictions also reduces computational complexities [143].

4.1.2 For Optimization Approximation

To enhance computational efficiency, the methods of optimization approximation prefer the computations that bring significant reduction in optimization errors. However, most of them focus on convex problems. Besides, researchers generally follow stereotyped routines to address their emerging problems without seeking more appropriate solutions. To further speed up the optimization for a broader range of scenarios, the following extensions can be considered.

Optimization for Nonconvex Problems. The optimization may fall into poor local minima when problems are nonconvex [187], leading to unstable performances in real-world tasks. Therefore, it is necessary to enhance the ability of algorithms to escape or bypass the poor minima. To this end, various randomized operations can be considered, such as randomized weight initialization and random noise on gradients [188]. Besides, inspired by curriculum learning [114], we may first utilize instances that are easy to fit and then gradually turn to difficult ones, such as boundary instances.

As a result, it allows algorithms to obtain a good solution at early stages and then tune it for a better minimum.

Solutions Inspired by Mathematical Models. By transforming target problems into classical models in other mathematical areas, such as geometry, we can borrow the experiences of handling these models to accelerate the optimization. For example, a simple iterative scheme used to find the minimum enclosing ball (MEB) can be used to solve large-scale SVM problems [194]. The motivation is, the dual of both MEB and SVM problems can be transformed into the same form. Similarly, [107] introduces cutting-plane algorithms that iteratively refine a feasible set to solve SVM on high-dimension sparse features.

4.1.3 For Computation Parallelism

Computation parallelism enhances computational capacities based on multiple computing devices. Benefit from existing systems, we can take advantage of flexible programming interfaces and low data communication for parallel data processing. However, with the fast upgrading of hardware, software, and predictive models, the following issues recently attract much attention.

Flexible Hardware Abstractions. Highly-efficient parallel computations require fast data communication between different nodes. Although allreduce is able to minimize the communication overhead for most methods, it supposes that the memory and the computational ability of different nodes are at the same level. Otherwise, the utilization of resources will be limited by the weakest one. Considering the real-world scenarios where distributed systems generally consist of heterogeneous devices, more flexible abstractions need to be studied. A possible solution is to utilize domain-specific knowledge such as intra-job predictability for cluster scheduling [159], [208]. Besides, we may further explore hybrid abstractions and take account of traffic optimization to balance computational resources [46], [87].

Modularized Open-Source Software. There are increasing interests in supporting open-source projects. Several motivations are as follows. The developments of parallel systems on general-purpose machine learning can be challenging even for large companies. Besides, open-source software can benefit more communities and democratize data science. To this end, normalized and modularized learning frameworks (similar to computer architectures [196]) are preferred. As a consequence, any localized enhancement can be achieved by contributors, improving the overall efficiency. In addition, to construct a vibrant and lively community, it is necessary to provide easy-to-use interfaces with different programming languages to benefit more researchers [145], as well as the simple postprocessing for productization.

4.2 Open Issues

4.2.1 Tighter Bounds of Complexities

Various complexities during the procedure of data analysis can be used to evaluate required computational resources in real-world tasks. Below we discuss some important ones.

Bounds of Computational Complexities. The computational complexity determines the numbers of computing devices and the spaces of memory. However, many existing bounds of computational complexities only provide general

guarantees for any probability distribution on instances and for any optimization algorithm that minimizes disagreement on training data. To estimate the complexities more precisely, it is necessary to establish tighter bounds for real-world data distributions and specific optimization algorithms, which may be achieved with the assistance of heuristical arguments from statistical physics [6].

Other Related Complexities. The measures of many other complexities also need to be studied. For example, over-parameterized networks can be well trained with a much smaller set of instances than the one estimated based on the current sample complexity [184]. Besides, more instances are required for adversarial training to build robustness [173]. In addition, the complexity of communication and the number of gates in a circuit can be used to estimate the practical limits on what machines can and cannot do [16].

4.2.2 Collection of Valuable Data

Large-scale datasets play a crucial role in LML, as the patterns for making decisions are learned from data without being explicitly programmed. Therefore, our machine learning community is in great need of large-scale instance-level datasets, and efficient methods for generating such data either in a supervised or unsupervised manner. To this end, we highlight the following directions.

Annotation Tools. Annotation tools aim at building large-scale annotated datasets by collecting contributions from a lot of people, such as LabelMe in computer vision [192]. To reduce the costs of obtaining high-quality labels and simplify the annotation procedure, it is essential to ease the modes of annotation and interaction for oracles with scalable active learning methods [109]. Besides, it is difficult to collect clean instances with categorical and strong supervision information. The reasons are as follows. First, there exist a large number of instances whose classes are fuzzy themselves. The one-hot or multi-hot encoding that groups them into specific classes can significantly harm their supervision information. Second, the inconsistent and inaccurate labels are ubiquitous in real-world situations due to subjective data-labeling processes. To address these issues, efficient weakly-supervised learning recently obtains urgent attention [237].

Data Augmentation. Data augmentation applies transformations to training sets to increase their scales. Based on the augmented data, we can improve the performance of predictive models, especially when original datasets are imbalanced or their instances are insufficient. These transformations can be as simple as flipping or rotating an image, or as complex as applying generative adversarial learning. Recently, BigGAN augments image datasets successfully by synthesizing high fidelity natural instances [37].

4.2.3 Moderate Specialization of Hardware

Over the past years, the design of computer architectures focuses on computation over communication. However, with the progress of semiconductor, we observed a new reality that data communication across processors becomes more expensive than the computation. Besides, the flexibility of general-purpose computing devices also makes the computation energy-inefficient in many emerging tasks. Although

some special-purpose accelerators have been employed to be orders of magnitude improvements such as GPUs, FPGA, and TPUs, most of them are customized to a single- or narrow-type of machine learning methods. Therefore, for future hardware, it is necessary to exploit both the performance and energy-efficiency of specialization while broadening compatible methods [95].

4.2.4 Quantum Machine Learning

Quantum computers exploit superposition and entanglement principles of quantum mechanics, obtaining an immense number of calculations in parallel. Compared to digital computers, they can reduce both execution time and energy consumption dramatically, such as IBM-Q. However, since the programming models of these quantum computers are fundamentally different from those of digital computers, more attention should be paid to redesign the corresponding machine learning methods [26], [162].

4.2.5 Privacy Protection

With the development of LML, increasing importance has been attached to privacy protection. Specifically, when personal and sensitive data are analyzed in the cloud or other distributed environments [8], it is necessary to ensure that the analysis will not violate the privacy of individuals. Recently, federate learning provides an alternative by bringing codes to edge devices and updating the global model with secure aggregation [30], [31].

5 CONCLUSION

Large-scale machine learning (LML) has significantly facilitated the data analysis in a mass scale over the past decades. However, despite these advances, the current LML requires further improvements to handle rapidly increasing data. In this paper, we first surveyed existing LML methods from three independent perspectives, namely, model simplification to reduce the computational complexity, optimization approximation to improve computational efficiency, and computation parallelism to enhance the computational capability. After that, we discussed the limitations of these methods and the possible extensions that can make further improvements. Besides, some important open issues in related areas are presented. We hope that this survey can provide a clean sketch on LML, and the discussions will advance the developments for next-generation methods.

REFERENCES

[1] "Baidu-allreduce," 2017. [Online]. Available: https://github.com/baidu-research/baidu-allreduce
[2] "Multi-worker mirrored strategy," 2020. [Online]. Available: https://www.tensorflow.org/guide/distributed_training#multiworkermirroredstrategy

[3] "Parallel learning of lightgbm," 2020. [Online]. Available: https://lightgbm.readthedocs.io/en/latest/Parallel-Learning-Guide.html

[4] "Torch.distributed," 2020. [Online]. Available: https://pytorch.org/docs/stable/distributed

[5] M. Abadi et al., "Tensorflow: A system for large-scale machine learning," in Proc. 12th USENIX Conf. Operating Syst. Des. Implementation, 2016, pp. 265–283.

[6] A. Agarwal and L. Bottou, "A lower bound for the optimization of finite sums," in Proc. 32nd Int. Conf. Mach. Learn., 2015, pp. 78–86.

[7] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in Proc. 24th Int. Conf. Neural Inf. Process. Syst., 2011, pp. 873–881.

[8] N. Agarwal, A. T. Suresh, F. Yu, S. Kumar, and H. B. Mcmahan, "cpSGD: Communication-efficient and differentially-private distributed SGD," in Proc. 32nd Int. Conf. Neural Inf. Process. Syst., 2018, pp. 7575–7586.

[9] S. Ahn, A. Korattikara, N. Liu, S. Rajan, and M. Welling, "Large-scale distributed bayesian matrix factorization using stochastic gradient MCMC," in Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2015, pp. 9–18.

[10] S. Ahn, A. Korattikara, and M. Welling, "Bayesian posterior sampling via stochastic gradient fisher scoring," in Proc. 29th Int. Conf. Mach. Learn., 2012, pp. 1591–1598.

[11] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid, "Good practice in large-scale learning for image classification," IEEE Trans. Pattern Anal. Mach. Intell., vol. 36, no. 3, pp. 507–520, Mar. 2014.

[12] O. Y. Al-Jarrah, P. D. Yoo, S. Muhaidat, G. K. Karagiannidis, and K. Taha, "Efficient machine learning for big data: A review," Big Data Res., vol. 2, no. 3, pp. 87–93, 2015.

[13] G. Alain, A. Lamb, C. Sankar, A. Courville, and Y. Bengio, "Variance reduction in SGD by distributed importance sampling," 2015, arXiv:1511.06481. [Online]. Available: Available: https://arxiv.org/abs/1511.06481

[14] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in Proc. 31st Conf. Neural Inf. Process. Syst., 2017, pp. 1709–1720.

[15] O. M. Aodha, N. D. F. Campbell, J. Kautz, and G. J. Brostow, "Hierarchical subquery evaluation for active learning on a graph," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2014, pp. 564–571.

[16] Y. Arjevani and O. Shamir, "Communication complexity of distributed convex learning and optimization," in Proc. 28th Int. Conf. Neural Inf. Process. Syst., 2015, pp. 1756–1764.

[17] Y. Baveye, E. Dellandréa, C. Chamaret, and L. Chen, "Deep learning vs kernel methods: Performance for emotion prediction in videos," in Proc. Int. Conf. Affect. Comput. Intell. Interact., 2015, pp. 77–83.

[18] I. Bayer, X. He, B. Kanagal, and S. Rendle, "A generic coordinate descent framework for learning from implicit feedback," in Proc. 26th Int. Conf. World Wide Web, 2017, pp. 1341–1350.

[19] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," SIAM J. Imag. Sci., vol. 2, no. 1, pp. 183–202, 2009.

[20] M. Belkin, S. Ma, and S. Mandal, "To understand deep learning we need to understand kernel learning," in Proc. 35th Int. Conf. Mach. Learn., 2018, pp. 541–549.

[21] Y. Ben-Haim and E. Tom-Tov, "A streaming parallel decision tree algorithm," J. Mach. Learn. Res., vol. 11, no. 2, pp. 849–872, 2010.

[22] S. Bengio, J. Weston, and D. Grangier, "Label embedding trees for large multi-class tasks," in Proc. 23rd Int. Conf. Neural Inf. Process. Syst., 2010, pp. 163–171.

[23] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in Neural Networks: Tricks of the Trade. Berlin, Germany: Springer, 2012, pp. 437–478.

[24] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," IEEE Trans. Pattern Anal. Mach. Intell., vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[25] J. Bergstra et al., "Theano: Deep learning on GPUs with python," in Proc. Conf. Neural Inf. Process. Syst., 2011, pp. 1–48.

[26] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," Nature, vol. 549, no. 7671, 2017, Art. no. 195.

[27] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," J. Mach. Learn. Res., vol. 3, pp. 993–1022, 2003.

[28] M. Boehm et al., "Systemml: Declarative machine learning on spark," Proc. VLDB Endowment, vol. 9, no. 13, pp. 1425–1436, 2016.

[29] M. Boehm et al., "Hybrid parallelization strategies for large-scale machine learning in systemml," Proc. VLDB Endowment, vol. 7, no. 7, pp. 553–564, 2014.

[30] K. Bonawitz et al., "Towards federated learning at scale: System design," 2019, arXiv: 1902.01046. [Online]. Available: https://arxiv.org/abs/1902.01046

[31] K. Bonawitz et al., "Practical secure aggregation for privacy-preserving machine learning," in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., 2017, pp. 1175–1191.

[32] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in Proc. 20th Int. Conf. Neural Inf. Process. Syst., 2008, pp. 161–168.

[33] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," SIAM Rev., vol. 60, no. 2, pp. 223–311, 2018.

[34] D. Bouneffouf and I. Birol, "Sampling with minimum sum of squared similarities for nystrom-based large scale spectral clustering," in Proc. 24th Int. Conf. Artif. Intell., 2015, pp. 2313–2319.

[35] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," Foundations Trends Mach. Learn., vol. 3, no. 1, pp. 1–122, 2011.

[36] L. Breiman, "Random forests," Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.

[37] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," 2018, arXiv: 1809.11096. [Online]. Available: https://arxiv.org/abs/1809.11096

[38] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer, "A stochastic quasi-newton method for large-scale optimization," SIAM J. Optim., vol. 26, no. 2, pp. 1008–1031, 2016.

[39] Z. Cai, Z. J. Gao, S. Luo, L. L. Perez, Z. Vagena, and C. Jermaine, "A comparison of platforms for implementing and running very large scale machine learning algorithms," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2014, pp. 1371–1382.

[40] D. Calandriello, I. Koutis, A. Lazaric, and M. Valko, "Improved large-scale graph learning through ridge spectral sparsification," in Proc. 35th Int. Conf. Mach. Learn., 2018, pp. 688–697.

[41] K.-W. Chang, C.-J. Hsieh, and C.-J. Lin, "Coordinate descent method for large-scale l2-loss linear support vector machines," J. Mach. Learn. Res., vol. 9, pp. 1369–1398, 2008.

[42] B. Chen, T. Medini, and A. Shrivastava, "Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems," 2019, arXiv: 1903.03129. [Online]. Available: https://arxiv.org/abs/1903.03129

[43] C. Chen, D. Carlson, Z. Gan, C. Li, and L. Carin, "Bridging the gap between stochastic gradient MCMC and stochastic optimization," in Proc. Int. Conf. Artif. Intell. Statist., 2016, pp. 1051–1060.

[44] H. Chen et al., "Addernet: Do we really need multiplications in deep learning?" in Proc. Comput. Vis. Pattern Recognit., 2020, pp. 1468–1477.

[45] J. Chen, H. R. Fang, and Y. Saad, "Fast approximate kNN graph construction for high dimensional data via recursive lanczos bisection," J. Mach. Learn. Res., vol. 10, no. 5, pp. 1989–2012, 2009.

[46] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in Proc. Conf. ACM Special Interest Group Data Commun., 2018, pp. 191–205.

[47] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," Mobile Netw. Appl., vol. 19, no. 2, pp. 171–209, 2014.

[48] R. Chen, J. Shi, Y. Chen, B. Zang, H. Guan, and H. Chen, "PowerLyra: Differentiated graph computation and partitioning on skewed graphs," ACM Trans. Parallel Comput., vol. 5, no. 3, pp. 1–39, 2019.

[49] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in Proc. Conf. Neural Inf. Process. Syst., 2018, pp. 6571–6583.

[50] T. Chen, I. Cano, and T. Zhou, "Rabit: A reliable allreduce and broadcast interface," Univ. Washington, Tech. Rep., 2015. [Online]. Available: https://x-algo.cn/wp-content/uploads/2016/09/rabit.pdf

[51] T. Chen, E. Fox, and C. Guestrin, "Stochastic gradient hamiltonian monte carlo," in Proc. 31st Int. Conf. Mach. Learn., 2014, pp. 1683–1691.

[52] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2016, pp. 785–794.

[53] T. Chen *et al.*, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015, *arXiv:1512.01274*. [Online]. Available: https://arxiv.org/abs/1512.01274

[54] X.-W. Chen and X. Lin, "Big data deep learning: Challenges and perspectives," *IEEE Access*, vol. 2, pp. 514–525, 2014.

[55] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," 2017, *arXiv: 1710.09282*. [Online]. Available: https://arxiv.org/abs/1710.09282

[56] W.-L. Chiang, M.-C. Lee, and C.-J. Lin, "Parallel dual coordinate descent method for large-scale linear classification in multi-core environments," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1485–1494.

[57] S. Chib and E. Greenberg, "Understanding the metropolis-hastings algorithm," *Amer. Statistician*, vol. 49, no. 4, pp. 327–335, 1995.

[58] W.-S. Chin, Y. Zhuang, Y.-C. Juan, and C.-J. Lin, "A fast parallel stochastic gradient method for matrix factorization in shared memory systems," *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 1, pp. 1–24, 2015.

[59] J. Dai *et al.*, "Deformable convolutional networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 764–773.

[60] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Commun. Pure Appl. Math.*, vol. 57, no. 11, pp. 1413–1457, 2004.

[61] Y. Dauphin, H. de Vries, and Y. Bengio, "Equilibrated adaptive learning rates for non-convex optimization," in *Proc. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1504–1512.

[62] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1223–1231.

[63] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[64] A. Defazio, F. Bach, and S. Lacoste-Julien, "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 1646–1654.

[65] J. Deng, S. Satheesh, A. C. Berg, and F. Li, "Fast and balanced: Efficient label tree learning for large scale object recognition," in *Proc. 25th Annu. Conf. Neural Inf. Process. Syst.*, 2011, pp. 567–575.

[66] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Trans. Signal Inf. Process.*, vol. 3, no. 2, pp. 1–29, 2014.

[67] I. S. Dhillon, P. K. Ravikumar, and A. Tewari, "Nearest neighbor based greedy coordinate descent," in *Proc. 24th Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 2160–2168.

[68] T. Dozat, "Incorporating nesterov momentum into adam," Tech. Rep., 2016. [Online]. Available: http://cs229.stanford.edu/proj2015/054_report.pdf

[69] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[70] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. Hoboken, NJ, USA: Wiley, 2012.

[71] C. Dünner *et al.*, "Snap ML: A hierarchical framework for machine learning," in *Proc. 32nd Conf. Neural Inf. Process. Syst.*, 2018, pp. 250–260.

[72] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 834–849.

[73] A. Farahat, A. Ghodsi, and M. Kamel, "A novel greedy algorithm for nyström approximation," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 269–277.

[74] K. Fatahalian, J. Sugerman, and P. Hanrahan, "Understanding the efficiency of GPU algorithms for matrix-matrix multiplication," in *Proc. ACM SIGGRAPH/EUROGRAPHICS Conf. Graph. Hardware*, 2004, pp. 133–137.

[75] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: A statistical view of boosting (with discussion and a rejoinder by the authors)," *Ann. Statist.*, vol. 28, no. 2, pp. 337–407, 2000.

[76] J. H. Friedman, "Stochastic gradient boosting," *Comput. Statist. Data Anal.*, vol. 38, no. 4, pp. 367–378, 2002.

[77] W. Fu, M. Wang, S. Hao, and T. Mu, "FLAG: Faster learning on anchor graph with label predictor optimization," *IEEE Trans. Big Data*, to be published, doi: 10.1109/TBDATA.2017.2757522.

[78] W. Fu, M. Wang, S. Hao, and X. Wu, "Scalable active learning by approximated error reduction," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2018, pp. 1396–1405.

[79] Y. Fujiwara and G. Irie, "Efficient label propagation," in *Proc. 31st Int. Conf. Mach. Learn.*, 2014, pp. 784–792.

[80] N. Gazagnadou, R. M. Gower, and J. Salmon, "Optimal minibatch and step sizes for saga," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 2142–2150.

[81] T. V. Gestel, J. A. Suykens, G. Lanckriet, A. Lambrechts, B. D. Moor, and J. Vandewalle, "Bayesian framework for least-squares support vector machine classifiers, gaussian processes, and kernel fisher discriminant analysis," *Neural Comput.*, vol. 14, no. 5, pp. 1115–1147, 2002.

[82] N. Gilardi and S. Bengio, "Local machine learning models for spatial data analysis," *J. Geographic Inf. Decis. Anal.*, vol. 4, pp. 11–28, 2000.

[83] A. Gittens and M. W. Mahoney, "Revisiting the nyström method for improved large-scale machine learning," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 3977–4041, 2016.

[84] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Proc. 10th USENIX Conf. Operating Syst. Des. Implementation*, 2012, Art. no. 2.

[85] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework," in *Proc. 11th USENIX Conf. Operating Syst. Des. Implementation*, 2014, pp. 599–613.

[86] S. Gopal, "Adaptive sampling for SGD by exploiting side information," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 364–372.

[87] P. Goyal *et al.*, "Accurate, large minibatch SGD: Training imagenet in 1 hour," 2017, *arXiv: 1706.02677*. [Online]. Available: https://arxiv.org/abs/1706.02677

[88] E. Grave, A. Joulin, M. Cissé, D. Grangier, and H. Jégou, "Efficient softmax approximation for GPUs," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 1302–1310.

[89] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proc. Nat. Acad. Sci. United States America*, vol. 101, no. suppl 1, pp. 5228–5235, 2004.

[90] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Rev.*, vol. 53, no. 2, pp. 217–288, 2011.

[91] S. Hanneke, "The optimal sample complexity of PAC learning," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1319–1333, 2016.

[92] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proc. Comput. Vis. Pattern Recognit.*, 2015, pp. 5353–5360.

[93] X. He, J. Tang, X. Du, R. Hong, T. Ren, and T.-S. Chua, "Fast matrix factorization with non-uniform weights on missing data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 8, pp. 2791–2804, Aug. 2020.

[94] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*. vol. 80, Philadelphia, PA, USA: Siam, 2002.

[95] M. D. Hill *et al.*, "21st century computer architecture," 2016, *arXiv:1609.06756*. [Online]. Available: https://arxiv.org/abs/1609.06756

[96] Q. Ho *et al.*, "More effective distributed ML via a stale synchronous parallel parameter server," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 1223–1231.

[97] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv: 1704.04861*. [Online]. Available: https://arxiv.org/abs/1704.04861

[98] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan, "A dual coordinate descent method for large-scale linear SVM," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 408–415.

[99] C. Igel, V. Heidrich-Meisner, and T. Glasmachers, "Shark," *J. Mach. Learn. Res.*, vol. 9, pp. 993–996, 2008.

[100] R. Jagerman, C. Eickhoff, and M. de Rijke, "Computing webscale topic models using an asynchronous parameter server," in *Proc. 40th Int. ACM SIGIR Conf.*, 2017, pp. 1337–1340.

[101] M. Jaggi *et al.*, "Communication-efficient distributed dual coordinate ascent," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 3068–3076.

[102] S. Jeaugey, "Nccl 2.0," 2017. [Online]. Available: https://ondemand.gputechconf.com/gtc/2017/presentation/s7155-jeaugey-nccl.pdf

[103] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. ACM 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.

[104] J. Jiang, B. Cui, C. Zhang, and F. Fu, "DimBoost: Boosting gradient boosting decision tree to higher dimensions," in *Proc. Int. Conf. Manage. Data*, 2018, pp. 1363–1376.

[105] J. Jiang, F. Fu, T. Yang, and B. Cui, "SketchML: Accelerating distributed machine learning with data sketches," in *Proc. Int. Conf. Manage. Data*, 2018, pp. 1269–1284.

[106] J. Jiang, L. Yu, J. Jiang, Y. Liu, and B. Cui, "Angel: A new large-scale machine learning system," *Nat. Sci. Rev.*, vol. 5, no. 2, pp. 216–236, 2018.

[107] T. Joachims, "Training linear SVMs in linear time," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2006, pp. 217–226.

[108] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 315–323.

[109] A. J. Joshi, F. Porikli, and N. P. Papanikolopoulos, "Scalable active learning for multiclass image classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2259–2273, Nov. 2012.

[110] Y. Kalantidis and Y. Avrithis, "Locally optimized product quantization for approximate nearest neighbor search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 2321–2328.

[111] V. Kalavri, V. Vlassov, and S. Haridi, "High-level programming abstractions for distributed graph processing," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 2, pp. 305–324, Feb. 2018.

[112] G. Ke *et al.*, "Lightgbm: A highly efficient gradient boosting decision tree," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 3146–3154.

[113] N. Ketkar *et al.*, *Deep Learning with Python.* vol. 1, Berlin, Germany: Springer, 2017.

[114] F. Khan, B. Mutlu, and X. Zhu, "How do humans teach: On curriculum learning and teaching dimension," in *Proc. 24th Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 1449–1457.

[115] J. Kim and J. F. Canny, "Interpretable learning for self-driving cars by visualizing causal attention," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2961–2969.

[116] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980

[117] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[118] S. Kumar, M. Mohri, and A. Talwalkar, "Sampling methods for the nyström method," *J. Mach. Learn. Res.*, vol. 13 pp. 981–1006, 2012.

[119] A. Kyrola, G. Blelloch, and C. Guestrin, "Graphchi: Large-scale graph computation on just a PC," in *Proc. 10th USENIX Conf. Operating Syst. Des. Implementation*, 2012, pp. 31–46.

[120] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin, "A survey of open source tools for machine learning with big data in the hadoop ecosystem," *J. Big Data*, vol. 2, no. 1, 2015, Art. no. 24.

[121] J. Langford, A. Smola, and M. Zinkevich, "Slow learners are fast," in *Proc. 22nd Int. Conf. Neural Inf. Process. Syst.*, 2009, pp. 2331–2339.

[122] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 265–272.

[123] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, 2015, Art. no. 436.

[124] Y. T. Lee and A. Sidford, "Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems," in *Proc. IEEE 54th Annu. Symp. Foundations Comput. Sci.*, 2013, pp. 147–156.

[125] H. Li and Z. Lin, "Accelerated proximal gradient methods for nonconvex programming," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 379–387.

[126] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Conf. Operating Syst. Des. Implementation*, 2014, pp. 583–598.

[127] Q. Li, "Fast parallel machine learning algorithms for large datasets using graphic processing unit," Ph. D. thesis, Virginia Commonwealth Univ., Richmond, VA, 2011.

[128] S. S. Liew, M. Khalil-Hani, and R. Bakhteri, "Bounded activation functions for enhanced training stability of deep neural networks on visual pattern recognition problems," *Neurocomputing*, vol. 216, pp. 718–734, 2016.

[129] Q. Lin, Z. Lu, and L. Xiao, "An accelerated proximal coordinate gradient method," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 3059–3067.

[130] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," 2017, *arXiv: 1712.01887*. [Online]. Available: https://arxiv.org/abs/1712.01887

[131] J. Liu, S. Wright, C. Ré, V. Bittorf, and S. Sridhar, "An asynchronous parallel stochastic coordinate descent algorithm," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 469–477.

[132] W. Liu, J. He, and S.-F. Chang, "Large graph construction for scalable semi-supervised learning," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 679–686.

[133] Y. Low, D. Bickson, J. E. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *Proc. VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[134] J. Lu, S. C. Hoi, J. Wang, P. Zhao, and Z.-Y. Liu, "Large scale online kernel learning," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1613–1655, 2016.

[135] S. Ma and M. Belkin, "Diving into the shallows: A computational perspective on large-scale shallow learning," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 3778–3787.

[136] Y.-A. Ma, T. Chen, and E. Fox, "A complete recipe for stochastic gradient MCMC," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2917–2925.

[137] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. Int. Conf. Mach. Learn.*, 2013, Art. no. 3.

[138] G. Malewicz *et al.*, "Pregel: A system for large-scale graph processing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 135–146.

[139] P.-G. Martinsson, V. Rokhlin, and M. Tygert, "A randomized algorithm for the decomposition of matrices," *Appl. Comput. Harmon. Anal.*, vol. 30, no. 1, pp. 47–68, 2011.

[140] B. McMahan and M. Streeter, "Delay-tolerant algorithms for asynchronous distributed online learning," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2915–2923.

[141] R. Memisevic, C. Zach, M. Pollefeys, and G. E. Hinton, "Gated softmax classification," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2010, pp. 1603–1611.

[142] X. Meng *et al.*, "MLlib: Machine learning in apache spark," *J. Mach. Learn. Res.*, vol. 17, no. 34, pp. 1–7, 2016.

[143] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.

[144] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2005, pp. 46–252.

[145] P. Moritz *et al.*, "Ray: A distributed framework for emerging AI applications," in *Proc. 13th USENIX Conf. Operating Syst. Des. Implementation*, 2018, pp. 561–577.

[146] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.

[147] B. Musel *et al.*, "Coarse-to-fine categorization of visual scenes in scene-selective cortex," *J. Cogn. Neurosci.*, vol. 26, no. 10, pp. 2287–2297, 2014.

[148] M. Mutny and A. Krause, "Efficient high dimensional bayesian optimization with additivity and quadrature fourier features," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 9018–9029.

[149] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 807–814.

[150] R. M. Neal *et al.*, "MCMC using hamiltonian dynamics," *Handbook Markov Chain Monte Carlo*, vol. 2, no. 11, 2011, Art. no. 2.

[151] Y. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," *SIAM J. Optim.*, vol. 22, no. 2, pp. 341–362, 2012.

[152] Y. Nesterov, "Gradient methods for minimizing composite functions," *Math. Program.*, vol. 140, no. 1, pp. 125–161, 2013.

[153] J. Nutini, M. Schmidt, I. Laradji, M. Friedlander, and H. Koepke, "Coordinate descent converges faster with the gauss-southwell rule than random selection," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 1632–1641.

[154] B. C. Ooi *et al.*, "SINGA: A distributed deep learning platform," in *Proc. 23rd ACM Int. Conf. Multimedia*, 2015, pp. 685–688.

[155] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations Trends Optim.*, vol. 1, no. 3, pp. 127–239, 2014.

[156] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

[157] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *J. Parallel Distrib. Comput.*, vol. 69, no. 2, pp. 117–124, 2009.

[158] S. Patterson and Y. W. Teh, "Stochastic gradient riemannian langevin dynamics on the probability simplex," in *Proc. 26th Int. Conf. Neural Inf. Processi. Syst.*, 2013, pp. 3102–3110.

[159] Y. Peng et al., "Towards comprehensive traffic forecasting in cloud computing: Design and application," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2210–2222, Aug. 2016.

[160] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Netw.*, vol. 12, no. 1, pp. 145–151, 1999.

[161] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2008, pp. 1177–1184.

[162] P. Rebentrost, M. Mohseni, and S. Lloyd, "Quantum support vector machine for big data classification," *Phys. Rev. Lett.*, vol. 113, no. 13, 2014, Art. no. 130503.

[163] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 693–701.

[164] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," 2019, *arXiv: 1904.09237*. [Online]. Available: https://arxiv.org/abs/1904.09237

[165] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture.*, 2016, pp. 1–13.

[166] P. Richtárik and M. Takáč, "Distributed coordinate descent method for learning with big data," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 2657–2681, 2016.

[167] J. Rosen et al. "Iterative MapReduce for large scale machine learning," 2013, *arXiv:1303.3517*. [Online]. Available: https://arxiv.org/abs/1303.3517

[168] A. Roy, I. Mihailovic, and W. Zwaenepoel, "X-stream: Edge-centric graph processing using streaming partitions," in *Proc. 24th ACM Symp. Operating Syst. Princ.*, 2013, pp. 472–488.

[169] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*. [Online]. Available: https://arxiv.org/abs/1609.04747

[170] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Mach. Intell.*, vol. 1, no. 5, pp. 206–215, 2019.

[171] O. Russakovsky et al., "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

[172] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, no. 3, pp. 660–674, May/Jun. 1991.

[173] L. Schmidt, S. Santurkar, D. Tsipras, K. Talwar, and A. Madry, "Adversarially robust generalization requires more data," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 5014–5026.

[174] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient," *Math. Program.*, vol. 162, no. 1–2, pp. 83–112, 2017.

[175] M. Schmidt, N. L. Roux, and F. R. Bach, "Convergence rates of inexact proximal-gradient methods for convex optimization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 1458–1466.

[176] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs," in *Proc. INTERSPEECH*, 2014, pp. 1058–1062.

[177] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in tensorflow," 2018, *arXiv: 1802.05799*. [Online]. Available: https://arxiv.org/abs/1802.05799

[178] H.-J. M. Shi, S. Tu, Y. Xu, and W. Yin, "A primer on coordinate descent algorithms," 2016, *arXiv:1610.00040*. [Online]. Available: https://arxiv.org/abs/1610.00040

[179] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, 2010, pp. 1–10.

[180] S. Si, C. J. Hsieh, and I. S. Dhillon, "Memory efficient kernel approximation," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 701–709.

[181] L. Sifre and S. Mallat, "Rigid-motion scattering for image classification," Ph.D. Dissertation. Citeseer, 2014. [Online]. Available: https://www.di.ens.fr/data/publications/papers/phd_sifre.pdf

[182] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: https://arxiv.org/abs/1409.1556

[183] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," 2017, *arXiv:1711.00489*. [Online]. Available: https://arxiv.org/abs/1711.00489

[184] M. Soltanolkotabi, A. Javanmard, and J. D. Lee, "Theoretical insights into the optimization landscape of over-parameterized shallow neural networks," *IEEE Trans. Inf. Theory*, vol. 65, no. 2, pp. 742–769, Feb. 2019.

[185] S. Sonnenburg et al., "The shogun machine learning toolbox," *J. Mach. Learn. Res.*, vol. 11, no. 6, pp. 1799–1802, 2010.

[186] B. Sriperumbudur and Z. Szabó, "Optimal rates for random fourier features," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1144–1152.

[187] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A survey of optimization methods from a machine learning perspective," *IEEE Trans. Cyberne.*, vol. 50, no. 8, pp. 3668–3681, Aug. 2020.

[188] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 1139–1147.

[189] C. Szegedy et al. "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[190] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.

[191] H. Tang, X. Lian, T. Zhang, and J. Liu, "Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 6155–6165.

[192] A. Torralba, B. C. Russell, and J. Yuen, "Labelme: Online image annotation and applications," *Proc. IEEE*, vol. 98, no. 8, pp. 1467–1484, Aug. 2010.

[193] C.-W. Tsai, C.-F. Lai, H.-C. Chao, and A. V. Vasilakos, "Big data analytics: A survey," *J. Big Data*, vol. 2, no. 1, 2015, Art. no. 21.

[194] I. W. Tsang, A. Kocsor, and J. T. Kwok, "Simpler core vector machines with enclosing balls," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, pp. 911–918.

[195] V. K. Vavilapalli et al., "Apache hadoop yarn: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput.*, 2013, Art. no. 5.

[196] J. Von Neumann, *The Computer and the Brain*. London, U.k.: Yale Univ. Press, 2012.

[197] J. Wang, J. Wang, G. Zeng, Z. Tu, R. Gan, and S. Li, "Scalable k-NN graph construction for visual descriptors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 1106–1113.

[198] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," 2014, *arXiv:1408.2927*. [Online]. Available: https://arxiv.org/abs/1408.2927

[199] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, "A survey on learning to hash," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 769–790, Apr. 2018.

[200] M. Wang, W. Fu, S. Hao, H. Liu, and X. Wu, "Learning on big graph: Label inference and regularization with anchor hierarchy," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 5, pp. 1101–1114, May 2017.

[201] M. Wang, W. Fu, S. Hao, D. Tao, and X. Wu, "Scalable semi-supervised learning by efficient anchor graph regularization," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 7, pp. 1864–1877, Jul. 2016.

[202] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 681–688.

[203] W. Wen et al., "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1509–1519.

[204] S. J. Wright, "Coordinate descent algorithms," *Math. Program.*, vol. 151, no. 1, pp. 3–34, 2015.

[205] J. Wu, W. Huang, J. Huang, and T. Zhang, "Error compensated quantized SGD and its applications to large-scale distributed optimization," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 5325–5333.

[206] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, "Deep image: Scaling up image recognition," 2015, *arXiv:1501.02876*. [Online]. Available: https://arxiv.org/abs/1501.02876

[207] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.

[208] W. Xiao *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *Proc. 13th USENIX Conf. Operating Syst. Des. Implementation*, 2018, pp. 595–610.

[209] E. P. Xing *et al.*, "Petuum: A new platform for distributed machine learning on big data," *IEEE Trans. Big Data*, vol. 1, no. 2, pp. 49–67, Jun. 2015.

[210] B. Xu *et al.*, "Efficient manifold ranking for image retrieval," in *Proc. 34th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2011, pp. 525–534.

[211] C. Xu, D. Tao, and C. Xu, "A survey on multi-view learning," 2013, *arXiv:1304.5634*. [Online]. Available: https://arxiv.org/abs/1304.5634

[212] W. Xu, "Towards optimal one pass large scale learning with averaged stochastic gradient descent," 2011, *arXiv:1107.2490*. [Online]. Available: https://arxiv.org/abs/11107.2490

[213] Z. Yan *et al.* "HD-CNN: Hierarchical deep convolutional neural networks for large scale visual recognition," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 2740–2748.

[214] T. Yang, Y.-F. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou, "Nyström method vs random fourier features: A theoretical and empirical comparison," in *Proc. Advances Neural Inf. Process. Syst.*, 2012, pp. 476–484.

[215] Y. Yang, M. Pilanci, and M. J. Wainwrights, "Randomized sketches for kernels: Fast and optimal nonparametric regression," *Ann. Statist.*, vol. 45, no. 3, pp. 991–1023, 2017.

[216] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 4801–4811.

[217] D. M. Young, *Iterative Solution of Large Linear Systems*. Amsterdam, The Netherlands: Elsevier, 2014.

[218] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv:1511.07122*, 2015. [Online]. Available: https://arxiv.org/abs/1511.07122

[219] M. Yu *et al.*, "GradiVeQ: Vector quantization for bandwidth-efficient gradient aggregation in distributed CNN training," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 5125–5135.

[220] G.-X. Yuan, C.-H. Ho, and C.-J. Lin, "Recent advances of large-scale linear classification," *Proc. IEEE*, vol. 100, no. 9, pp. 2584–2603, Sep. 2012.

[221] S. Yun and K.-C. Toh, "A coordinate gradient descent method for l1-regularized convex minimization," *Comput. Optim. Appl.*, vol. 48, no. 2, pp. 273–307, 2011.

[222] M. Zaharia *et al.*, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation*, 2012, pp. 15–28.

[223] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput.*, 2010, p. 10.

[224] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: fault-tolerant streaming computation at scale," in *Proc. 24th ACM Symp. Operating Syst. Princ.*, 2013, pp. 423–438.

[225] M. D. Zeiler, "Adadelta: An adaptive learning rate method," 2012, *arXiv:1212.5701*. [Online]. Available: https://arxiv.org/abs/1212.5701

[226] H. Zhang, J. Li, K. Kara, D. Alistarh, J. Liu, and C. Zhang, "ZipML: Training linear models with end-to-end low precision, and a little bit of deep learning," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 4035–4043.

[227] H. Zhang, F. Shen, W. Liu, X. He, H. Luan, and T.-S. Chua, "Discrete collaborative filtering," in *Proc. 39th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2016, pp. 325–334.

[228] K. Zhang, L. Lan, J. T. Kwok, S. Vucetic, and B. Parvin, "Scaling up graph-based semisupervised learning via prototype vector machines," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 3, pp. 444–457, Mar. 2015.

[229] K. Zhang, I. W. Tsang, and J. T. Kwok, "Improved nyström low-rank approximation and error analysis," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 1232–1239.

[230] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "A survey on deep learning for big data," *Inf. Fusion*, vol. 42, pp. 146–157, 2018.

[231] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6848–6856.

[232] Y.-M. Zhang, K. Huang, G. Geng, and C.-L. Liu, "Fast KNN graph construction with locality sensitive hashing," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2013, pp. 660–674.

[233] Z. Zhang, B. Cui, Y. Shao, L. Yu, J. Jiang, and X. Miao, "PS2: Parameter server on spark," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 376–388.

[234] Y. LeCun, S. Zhang, A. Choromanska, "Deep learning with elastic averaging SGD," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 685–693.

[235] T. Zhao, M. Yu, Y. Wang, R. Arora, and H. Liu, "Accelerated mini-batch randomized block coordinate descent method," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 3329–3337.

[236] S. Zheng *et al.* "Asynchronous stochastic gradient descent with delay compensation," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 4120–4129.

[237] Z.-H. Zhou, "A brief introduction to weakly supervised learning," *Nat. Sci. Rev.*, vol. 5, no. 1, pp. 44–53, 2017.

[238] X. Zhu, W. Han, and W. Chen, "Gridgraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2015, pp. 375–386.

[239] X. Zhu, H. Hu, S. Lin, and J. Dai, "Deformable convNets V2: More deformable, better results," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 9308–9316.

[240] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2010, pp. 2595–2603.
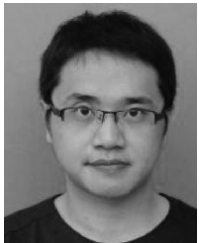
Meng Wang (Senior Member, IEEE) received the BE and PhD degrees in special class for the gifted young from the Department of Electronic Engineering and Information Science, University of Science and Technology of China (USTC), Hefei, China, in 2003 and 2008, respectively. He is currently a professor with the Hefei University of Technology (HFUT), China. He has authored more than 200 book chapters, journal, and conference papers in his research areas. His current research interests include multimedia content analysis, computervision, and pattern recognition. He was a recipient of the ACM SIGMM Rising Star Award in 2014. He is an associate editor of the *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Circuits and Systems for Video Technology*, and *IEEE Transactions on Neural Networks and Learning Systems*.
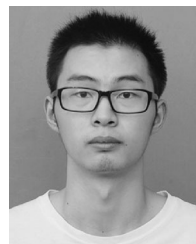
Weijie Fu is currently working toward the PhD degree in the School of Computer Science and Information Engineering, Hefei University of Technology (HFUT). His current research interest focuses on machine learning and data mining.

Xiangnan He received the PhD degree in computer science from the National University of Singapore (NUS), Singapore, in 2016, and did postdoctoral research in NUS until 2018. He is currently a professor with the University of Science and Technology of China (USTC). His research interests include span information retrieval, data mining, and multi-media analytics. He has more than 50 publications appeared in several top conferences such as SIGIR, WWW, and MM, and journals including the *IEEE Transactions on Knowledge and Data Engineering*, the *Tongue Operated Integrated System*, and the *IEEE Transactions on Multimedia*. His work on recommender systems has received the Best Paper Award Honourable Mention in WWW 2018 and ACM SIGIR 2016. Moreover, he has served as the PC member for several top conferences including SIGIR, WWW, MM, KDD etc., and the regular reviewer for journals including the *IEEE Transactions on Knowledge and Data Engineering*, the *Tongue Operated Integrated System*, the *IEEE Transactions on Multimedia*, the *IEEE Transactions on Neural Networks and Learning Systems* etc.

Xindong Wu (Fellow, IEEE) received the bachelor's and master's degrees in computer science from the Hefei University of Technology, China, and the PhD degree in artificial intelligence from the University of Edinburgh, Britain. He is Yangtze River Scholar with the School of Computer Science and Information Engineering, Hefei University of Technology (HFUT), China. His research interests include data mining, knowledge-based systems, and Web information exploration. He is the Steering Committee Chair of the IEEE ICDM, the editor-in-chief of KAIS, by Springer, and a Series Editor of the Springer Book Series on AI&KP. He was the editor-in-chief of the *IEEE Transactions on Knowledge and Data Engineering* between 2005 and 2008. He served as Program Committee chair/co-chair for ICDM '03, KDD-07, and CIKM 2010.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Shijie Hao received the BE, MS, and PhD degrees from the School of Computer Science and Information Engineering from the Hefei University of Technology, Hefei, Anhui. He is an associate professor with the Hefei University of Technology (HFUT), China. His current research interests include machine learning and image processing.