A Survey of Optimization Methods From a Machine Learning Perspective

Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao

***Abstract*—Machine learning develops rapidly, which has made many theoretical breakthroughs and is widely applied in various fields. Optimization, as an important part of machine learning, has attracted much attention of researchers. With the exponential growth of data amount and the increase of model complexity, optimization methods in machine learning face more and more challenges. A lot of work on solving optimization problems or improving optimization methods in machine learning has been proposed successively. The systematic retrospect and summary of the optimization methods from the perspective of machine learning are of great significance, which can offer guidance for both developments of optimization and machine learning research. In this article, we first describe the optimization problems in machine learning. Then, we introduce the principles and progresses of commonly used optimization methods. Finally, we explore and give some challenges and open problems for the optimization in machine learning.**

***Index Terms*—Approximate Bayesian inference, deep neural network (DNN), machine learning, optimization method, reinforcement learning (RL).**

I. Introduction

RECENTLY, machine learning has grown at a remarkable rate, attracting a great number of researchers and practitioners. It has become one of the most popular research directions and plays a significant role in many fields, such as machine translation, speech recognition, image recognition, recommendation systems, etc. Optimization is one of the core components of machine-learning. The essence of most machine learning algorithms is to build an optimization model and learn the parameters in the objective function from the given data. In the era of immense data, the effectiveness and efficiency of the numerical optimization algorithms dramatically influence the popularization and application of the machine-learning models. In order to promote the development of machine learning, a series of effective optimization methods was put forward, which have improved the performance and efficiency of the machine-learning methods.

Manuscript received June 16, 2019; revised October 24, 2019; accepted October 28, 2019. Date of publication November 18, 2019; date of current version July 10, 2020. This work was supported in part by NSFC under Project 61673179, and in part by Shanghai Sailing Program under Grant 17YF1404600. This article was recommended by Associate Editor Y. Xia. *(Corresponding author: Jing Zhao.)*

The authors are with the School of Computer Science and Technology, East China Normal University, Shanghai 200062, China (e-mail: shiliangsun@gmail.com; jzhao@cs.ecnu.edu.cn).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the author.

Digital Object Identifier 10.1109/TCYB.2019.2950779

From the perspective of the gradient information in optimization, popular optimization methods can be divided into three categories: 1) first-order optimization methods, which are represented by the widely used stochastic gradient methods; 2) high-order optimization methods, in which Newton's method is a typical example; and 3) heuristic derivative-free optimization methods, in which the coordinate descent method is a representative.

As the representative of the first-order optimization methods, the stochastic gradient descent (SGD) method [1], [2], as well as its variants, has been widely used in recent years and is evolving at a high speed. However, many users pay little attention to the characteristics or application scope of these methods. They often adopt them as black-box optimizers, which may limit the functionality of the optimization methods. In this article, we comprehensively introduce the fundamental optimization methods. Particularly, we systematically explain their advantages and disadvantages, their application scope, and the characteristics of their parameters. We hope that the targeted introduction will help users choose the first-order optimization methods more conveniently and make parameter adjustment more reasonable in the learning process.

Compared with the first-order optimization methods, high-order methods [3]–[5] converge at a faster speed in which the curvature information makes the search direction more effective. High-order optimizations attract widespread attention but face more challenges. The difficulty in the high-order methods lies in the operation and storage of the inverse matrix of the Hessian matrix. To solve this problem, many variants based on Newton's method have been developed, most of which try to approximate the Hessian matrix through some techniques [6], [7]. In subsequent studies, the stochastic quasi-Newton method and its variants are introduced to extend high-order methods to large-scale data [8]–[10].

The derivative-free optimization methods [11], [12] are mainly used in the case that the derivative of the objective function may not exist or be difficult to calculate. There are two main ideas in the derivative-free optimization methods. One is adopting a heuristic search based on empirical rules, and the other is fitting the objective function with samples. The derivative-free optimization methods can also work in conjunction with the gradient-based methods.

Most machine-learning problems, once formulated, can be solved as the optimization problems. Optimization in the fields of deep neural network (DNN), reinforcement learning (RL), meta learning, variational inference- and Markov chain Monte Carlo (MCMC) encounters different difficulties and challenges. The optimization methods developed in the specific machine-learning fields are different, which can be inspiring to the development of general optimization methods.

DNNs have shown great success in pattern recognition and machine learning. There are two popular NNs, that is, convolutional neural networks (CNNs) [13] and recurrent neural networks (RNNs), which play important roles in various fields of machine learning. CNNs are feedforward neural networks with convolution calculation. CNNs have been successfully used in many fields, such as image processing [14], [15]; video processing [16]; and natural language processing (NLP) [17], [18]. RNNs are a kind of sequential model and very active in NLP [19]–[22]. Besides, RNNs are also popular in the fields of image processing [23], [24] and video processing [25]. In the field of constrained optimization, RNNs can achieve excellent results [26]–[29]. In these works, the parameters of weights in RNNs can be learned by analytical methods, and these methods can find the optimal solution according to the trajectory of the state solution.

The stochastic gradient-based algorithms are widely used in DNNs [30]–[33]. However, various problems are emerging when employing stochastic gradient-based algorithms. For example, the learning rate will be oscillating in the later training stage of some adaptive methods [34], [35], which may lead to the problem of nonconvergence. Thus, further optimization algorithms based on variance reduction were proposed to improve the convergence rate [36], [37]. Moreover, combining the SGD and the characteristics of its variants is a possible direction to improve the optimization. Especially, switching an adaptive algorithm to the SGD method can improve the accuracy and convergence speed of the algorithm [38].

RL is a branch of machine learning, for which an agent interacts with the environment by trial-and-error mechanism and learns an optimal policy by maximizing cumulative rewards [39]. Deep RL combines the RL and deep learning techniques and enables the RL agent to have a good perception of its environment. Recent research has shown that deep learning can be applied to learn a useful representation for the RL problems [40]–[44]. The stochastic optimization algorithms are commonly used in RL and deep RL models.

Meta learning [45], [46] has recently become very popular in the field of machine learning. The goal of meta learning is to design a model that can efficiently adapt to the new environment with as few samples as possible. The application of meta learning in supervised learning can solve few-shot learning problems [47]. In general, the meta learning methods can be summarized into the following three types [48]: 1) metric-based methods [49]–[52]; 2) model-based methods [53], [54]; and 3) optimization-based methods [47], [55], [56]. We will describe the details of optimization-based meta learning methods in the subsequent sections.

Variational inference is a useful approximation method which aims to approximate the posterior distributions in the Bayesian machine learning. It can be considered as an optimization problem. For example, mean-field variational inference uses a coordinate ascent to solve this optimization problem [57]. As the amount of data increase continuously, it is not friendly to use the traditional optimization method to handle the variational inference. Thus, the stochastic variational inference was proposed, which introduced natural gradients and extended the variational inference to large-scale data [58].

Optimization methods have a significative influence on various fields of machine learning. For example, Pajarinen *et al.* [5] proposed the transformer network using Adam optimization [33], which is applied to machine translation tasks. Ledig *et al.* [59] proposed the super-resolution generative adversarial network for image super resolution, which is also optimized by Adam. Wu *et al.* [60] proposed actor–critic using trust-region optimization to solve the deep RL on Atari games as well as the MuJoCo environments.

The stochastic optimization method can also be applied to MCMC sampling to improve efficiency. In this kind of application, stochastic gradient Hamiltonian Monte Carlo (HMC) is a representative method [61] where the stochastic gradient accelerates the step of gradient update when handling large-scale samples. The noise introduced by the stochastic gradient can be characterized by introducing the Gaussian noise and friction terms. In addition, the deviation caused by HMC discretization can be eliminated by the friction term, and thus the Metropolis–Hasting step can be omitted. The hyper parameter settings in the HMC will affect the performance of the model. There are some efficient ways to automatically adjust the hyperparameters and improve the performance of the sampler.

The development of optimization brings a lot of contributions to the progress of machine learning. However, there are still many challenges and open problems for optimization problems in machine learning.

1) How to improve the optimization performance with insufficient data in DNNs is a tricky problem. If there are not enough samples in the training of DNNs, it is prone to cause the problem of high variances and overfitting [62]. In addition, nonconvex optimization has been one of the difficulties in DNNs, which makes the optimization tend to obtain a locally optimal solution rather than the global optimal solution.
2) For sequential models, the samples are often truncated by batches when the sequence is too long, which will cause deviation. How to analyze the deviation of stochastic optimization in this case and correct it is vital.
3) The stochastic variational inference is graceful and practical, and it is probably a good choice to develop methods of applying high-order gradient information to stochastic variational inference.
4) It may be a great idea to introduce the stochastic technique to the conjugate gradient method to obtain an elegant and powerful optimization algorithm. The detailed techniques to make improvements in the stochastic conjugate gradient is an interesting and challenging problem.

The purpose of this article is to summarize and analyze the classical and modern optimization methods from a machine-learning perspective. The remainder of this article is organized as follows. Section II summarizes the machine-learning problems from the perspective of optimization. Section III discusses the classical optimization algorithms and their latest developments in machine learning. Particularly, the recent popular optimization methods including the first- and second-order optimization algorithms are emphatically introduced. Section IV presents the challenges and open problems in the optimization methods. In Section V, we conclude this article. We introduce the developments and applications of optimization methods in some specific machine-learning fields in the supplementary material.

II. Machine Learning Formulated as Optimization

Almost all machine-learning algorithms can be formulated as an optimization problem to find the extremum of an objective function. Building models and constructing reasonable objective functions are the first step in the machine-learning methods. With the determined objective function, appropriate numerical or analytical optimization methods are usually used to solve the optimization problem.

According to the modeling purpose and the problem to be solved, the machine-learning algorithms can be divided into supervised learning, semisupervised learning (SSL), unsupervised learning, and RL. Particularly, supervised learning is further divided into the classification problem (e.g., sentence classification [17], [63]; image classification [64]–[66], etc.) and regression problem; unsupervised learning is divided into clustering and dimension reduction [67]–[69], among others.

A. Optimization Problems in Supervised Learning

For supervised learning, the goal is to find an optimal mapping function $f(x)$ to minimize the loss function of the training samples

$$\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} L\left(y^i, f\left(x^i, \theta\right)\right) \tag{1}$$

where N is the number of training samples, θ is the parameter of the mapping function, x^i is the feature vector of the ith samples, y^i is the corresponding label, and L is the loss function.

There are many kinds of loss functions in supervised learning, such as the square of Euclidean distance, cross-entropy, contrast loss, hinge loss, information gain- and so on. For the regression problems, the simplest way is using the square of Euclidean distance as the loss function, that is, minimizing square errors on training samples. But the generalization performance of this kind of empirical loss is not necessarily good. Another typical form is structured risk minimization, whose representative method is the support vector machine. On the objective function, regularization items are usually added to alleviate overfitting, for example, in terms of L_2-norm

$$\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} L\left(y^i, f\left(x^i, \theta\right)\right) + \lambda\|\theta\|_2^2 \tag{2}$$

where λ is the compromise parameter, which can be determined through cross-validation.

B. Optimization Problems in Semisupervised Learning

SSL is the method between supervised and unsupervised learning, which incorporates labeled data and unlabeled data during the training process. It can deal with different tasks, including classification tasks [70], [71]; regression tasks [72]; clustering tasks [73], [74]; and dimensionality reduction tasks [75], [76]. There are different kinds of SSL methods, including self-training, generative models, semisupervised support vector machines (S3VM) [77], graph-based methods, multilearning methods, and others. We take S3VM as an example to introduce the optimization in SSL.

S3VM is a learning model that can deal with binary classification problems and only part of the training set in this problem is labeled. Let D^l be the labeled data which can be represented as $D^l = \{\{x^1, y^1\}, \{x^2, y^2\}, \ldots, \{x^l, y^l\}\}$, and D^u be the unlabeled data which can be represented as $D^u = \{x^{l+1}, x^{l+2}, \ldots, x^N\}$ with $N = l + u$. In order to use the information of unlabeled data, an additional constraint on the unlabeled data is added to the original objective of SVM with slack variables ζ^i. Specifically, defining ϵ^j as the misclassification error of the unlabeled instance if its true label is positive and z^j as the misclassification error of the unlabeled instance if its true label is negative. The constraint means to make $\sum_{j=l+1}^{N}\min(\epsilon^i, \zeta^i)$ as small as possible. Thus, an S3VM problem can be described as

$$\begin{aligned} \min \quad & \|\omega\| + C\left[\sum_{i=1}^{l}\zeta^i + \sum_{j=l+1}^{N}\min\left(\epsilon^i, z^j\right)\right] \\ \text{subject to} \quad & y^i\left(\mathbf{w}\cdot x^i + b\right) + \zeta^i \geq 1, \zeta \geq 0, i = 1, \ldots, l \\ & \mathbf{w}\cdot x^j + b + \epsilon^j \geq 1, \epsilon \geq 0, j = l+1, \ldots, N \\ & -\left(\mathbf{w}\cdot x^j + b\right) + z^j \geq 1, z^j \geq 0 \end{aligned} \tag{3}$$

where C is a penalty coefficient. The optimization problem in S3VM is a mixed-integer problem which is difficult to deal with [78]. There are various methods summarized in [79] to deal with this problem, such as the branch-and-bound techniques [80] and convex relaxation methods [81].

C. Optimization Problems in Unsupervised Learning

Clustering algorithms [67], [82]–[84] divide a group of samples into multiple clusters, ensuring that the differences between the samples in the same cluster are as small as possible, and samples in different clusters are as different as possible. The optimization problem for the k-means clustering algorithm is formulated as minimizing the following loss function:

$$\min_{S} \sum_{k=1}^{K}\sum_{x\in S_k}\|x - \mu_k\|_2^2 \tag{4}$$

where K is the number of clusters, x is the feature vector of samples, μ_k is the center of cluster k, and S_k is the sample set of cluster k. The implication of this objective function is to make the sum of variances of all clusters as small as possible.

The dimensionality reduction algorithm ensures that the original information from data is retained as much as possible after projecting them into the low-dimensional space.

The principal component analysis (PCA) [85]–[87] is a typical algorithm of dimensionality reduction methods. The objective of PCA is formulated to minimize the reconstruction error as

$$\min \sum_{i=1}^{N} \left\| \overline{x}^i - x^i \right\|_2^2 \quad \text{where} \quad \overline{x}^i = \sum_{j=1}^{D'} z_j^i e_j, D \gg D' \tag{5}$$

where N represents the number of samples, x_i is a D-dimensional vector, and $\overline{x}^i$ is the reconstruction of x^i. $z^i = \{z_1^i, \ldots, z_{D'}^i\}$ is the projection of x^i in D'-dimensional coordinates. e_j is the standard orthogonal basis under D'-dimensional coordinates.

Another common optimization goal in the probabilistic models is to find an optimal probability density function of $p(x)$, which maximizes the logarithmic-likelihood function (MLE) of the training samples

$$\max \sum_{i=1}^{N} \ln p\left(x^i; \theta\right). \tag{6}$$

In the framework of the Bayesian methods, some prior distributions are often assumed on parameter θ, which also has the effect of alleviating overfitting.

D. Optimization Problems in Reinforcement Learning

RL [42], [88], [89], unlike supervised learning and unsupervised learning, aims to find an optimal strategy function, whose output varies with the environment. For a deterministic strategy, the mapping function from state s to action a is the learning target. For an uncertain strategy, the probability of executing each action is the learning target. In each state, the action is determined by $a = \pi(s)$, where $\pi(s)$ is the policy function.

The optimization problem in RL can be formulated as maximizing the cumulative return after executing a series of actions which are determined by the policy function

$$\max_{\pi} V_\pi(s) \quad \text{where} \quad V_\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | S_t = s\right] \tag{7}$$

where $V_\pi(s)$ is the value function of state s under policy π, r is the reward, and $\gamma \in [0, 1]$ is the discount factor.

E. Optimization for Machine Learning

Overall, the main steps of machine learning are to build a model hypothesis, define the objective function, and solve the maximum or minimum of the objective function to determine the parameters of the model. In these three vital steps, the first two steps are the modeling problems of machine learning, and the third step is to solve the desired model by optimization methods.

III. Fundamental Optimization Methods and Progresses

From the perspective of gradient information, the fundamental optimization methods can be divided into first-order optimization methods, high-order optimization methods, and derivative-free optimization methods. These methods have a long history and are constantly evolving. They are progressing in many practical applications and have achieved good performance. Here, we mainly introduce the first-order optimization methods. The high-order and derivative-free optimization methods are presented in the supplementary material. Besides these fundamental methods, preconditioning is a useful technique for optimization methods. Applying reasonable preconditioning can reduce the number of iterations and obtain better spectral characteristics. These technologies have been widely used in practice. For the convenience of researchers, we summarize the existing common optimization toolkits in a table at the end of this section.

A. First-Order Methods

In the field of machine learning, the most commonly used first-order optimization methods are mainly based on gradient descent. In this section, we introduce some of the representative algorithms along with the development of the gradient descent methods. At the same time, the classical alternating direction method of multipliers (ADMMs) and the Frank–Wolfe method in numerical optimization are also introduced.

1) Gradient Descent: The gradient descent method is the earliest and most common optimization method. The idea of the gradient descent method is that variables update iteratively in the (opposite) direction of the gradients of the objective function. The update is performed to gradually converge to the optimal value of the objective function. The learning rate η determines the step size in each iteration, and thus influences the number of iterations to reach the optimal value [90].

The steepest descent algorithm is a widely known algorithm. The idea is to select an appropriate search direction in each iteration so that the value of the objective function minimizes the fastest. Gradient descent and steepest descent are not the same because the direction of the negative gradient does not always descend fastest. Gradient descent is an example of using the Euclidean norm in steepest descent [91].

Next, we give the formal expression of the gradient descent method. For a linear regression model, we assume that $f_\theta(x)$ is the function to be learned, $L(\theta)$ is the loss function, and θ is the parameter to be optimized. The goal is to minimize the loss function with

$$L(\theta) = \frac{1}{2N} \sum_{i=1}^{N} \left(y^i - f_\theta\left(x^i\right)\right)^2 \tag{8}$$

$$f_\theta(x) = \sum_{j=1}^{D} \theta_j x_j \tag{9}$$

where N is the number of training samples, D is the number of input features, x^i is an independent variable with $x^i = (x_1^i, \ldots, x_D^i)$ for $i = 1, \ldots, N$, and y^i is the target output. The gradient descent alternates the following two steps until it converges.

1) Derive $L(\theta)$ for θ_j to obtain the gradient corresponding to each θ_j

$$\frac{\partial L(\theta)}{\partial \theta_j} = -\frac{1}{N} \sum_{i=1}^{N} \left(y^i - f_\theta\left(x^i\right)\right) x_j^i. \tag{10}$$

2) Update each θ_j in the negative gradient direction to minimize the risk function

$$\theta_j' = \theta_j + \eta \cdot \frac{1}{N}\sum_{i=1}^{N}\left(y^i - f_\theta\left(x^i\right)\right)x_j^i. \tag{11}$$

The gradient descent method is simple to implement. The solution is global optimal when the objective function is convex. It often converges at a slower speed if the variable is closer to the optimal solution, and more careful iterations need to be performed.

In the above linear regression example, note that all of the training data are used in each iteration step, so the gradient descent method is also called the batch gradient descent. If the number of samples is N and the dimension of x is D, the computational complexity for each iteration will be $O(ND)$. In order to mitigate the cost of computation, some parallelization methods were proposed [92], [93]. However, the cost is still hard to accept when dealing with large-scale data. Thus, the SGD method emerges.

2) Stochastic Gradient Descent: Since the batch gradient descent has high computational complexity in each iteration for large-scale data and does not allow online update, SGD was proposed [1]. The idea of SGD is using one sample randomly to update the gradient per iteration, instead of directly calculating the exact value of the gradient. The stochastic gradient is an unbiased estimate of the real gradient [1]. The cost of the SGD algorithm is independent of sample numbers and can achieve sublinear convergence speed [37]. SGD reduces the update time to deal with large numbers of samples and removes a certain amount of computational redundancy, which significantly accelerates the calculation. In the strong convex problem, SGD can achieve the optimal convergence speed [36], [94]–[96]. Meanwhile, it overcomes the disadvantage of batch gradient descent that cannot be used for online learning.

The loss function (8) can be written as the following equation:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\left(y^i - f_\theta\left(x^i\right)\right)^2 = \frac{1}{N}\sum_{i=1}^{N}\operatorname{cost}\left(\theta, \left(x^i, y^i\right)\right). \tag{12}$$

If a random sample i is selected in SGD, the loss function will be $L^*(\theta)$

$$L^*(\theta) = \operatorname{cost}\left(\theta, \left(x^i, y^i\right)\right) = \frac{1}{2}\left(y^i - f_\theta\left(x^i\right)\right)^2. \tag{13}$$

The gradient update in SGD uses the random sample i rather than all samples in each iteration

$$\theta' = \theta + \eta\left(y^i - f_\theta\left(x^i\right)\right)x^i. \tag{14}$$

Since SGD uses only one sample per iteration, the computational complexity for each iteration is $O(D)$, where D is the number of features. The update rate for each iteration of SGD is much faster than that of batch gradient descent when the number of samples N is large. SGD increases the overall optimization efficiency at the expense of more iterations, but the increased iteration number is insignificant compared with the high computational complexity caused by large numbers of samples. It is possible to use only thousands of samples overall to obtain the optimal solution even when the sample size is hundreds of thousands. Therefore, compared with batch methods, SGD can effectively reduce the computational complexity and accelerate convergence.

However, one problem in SGD is that the gradient direction oscillates because of additional noise introduced by random selection, and the search process is blind in the solution space. Unlike batch gradient descent which always moves toward the optimal value along the negative direction of the gradient, the variance of gradients in SGD is large and the movement direction in SGD is biased. So a compromise between the two methods, the mini-batch gradient descent method (MSGD), was proposed [1].

The MSGD uses b independent identically distributed samples (b is generally in 50 to 256 [90]) as the sample sets to update the parameters in each iteration. It reduces the variance of the gradients and makes the convergence more stable, which helps to improve the optimization speed. For brevity, we will call MSGD as SGD in the following sections.

As a common feature of stochastic optimization, SGD has a better chance of finding the global optimal solution for complex problems. The deterministic gradient in batch gradient descent may cause the objective function to fall into a local minimum for the multimodal problem. The fluctuation in the SGD helps the objective function jump to another possible minimum. However, the fluctuation in SGD always exists, which may more or less slow down the process of convergence.

There are still many details to be noted about the use of SGD in the concrete optimization process [90], such as the choice of a proper learning rate. A too small learning rate will result in a slower convergence rate, while a too large learning rate will hinder convergence, making loss function fluctuate at the minimum. One way to solve this problem is to set up a predefined list of learning rates or a certain threshold and adjust the learning rate during the learning process [1], [97]. However, these lists or thresholds need to be defined in advance according to the characteristics of the dataset. It is also inappropriate to use the same learning rate for all parameters. If data are sparse and features occur at different frequencies, it is not expected to update the corresponding variables with the same learning rate. A higher learning rate is often expected for less frequently occurring features [30], [33].

Besides the learning rate, how to avoid the objective function being trapped in infinite numbers of the local minimum is a common challenge. Some work has proved that this difficulty does not come from the local minimum values, but comes from the "saddle point" [98]. The slope of a saddle point is positive in one direction and negative in another direction, and gradient values in all directions are zero. It is an important problem for SGD to escape from these points. Some research about escaping from saddle points was developed [99], [100].

3) Nesterov Accelerated Gradient Descent: Although SGD is popular and widely used, its learning process is sometimes prolonged. How to adjust the learning rate, how to speed up the convergence, and how to prevent from being trapped at a local minimum during the search are worthwhile research directions.

Much work is presented to improve SGD. For example, the momentum idea was proposed to be applied in SGD [101]. The concept of momentum is derived from the mechanics of physics, which simulates the inertia of objects. The idea of applying momentum in SGD is to preserve the influence of the previous update direction on the next iteration to a certain degree. The momentum method can speed up the convergence when dealing with high curvature, small but consistent gradients, or noisy gradients [102]. The momentum algorithm introduces the variable v as the speed, which represents the direction and the rate of the parameter's movement in the parameter space. The speed is set as the average exponential decay of the negative gradient.

In the gradient descent method, the speed update is $v = \eta \cdot (-(\partial L(\theta))/\partial(\theta))$ each time. Using the momentum algorithm, the amount of the update v is not just the amount of gradient descent calculated by $\eta \cdot (-(\partial L(\theta))/\partial(\theta))$. It also takes into account the friction factor, which is represented as the previous update v^{old} multiplied by a momentum factor ranging between [0, 1]. Generally, the mass of the object is set to 1. The formulation is expressed as

$$v = \eta \cdot \left(-\frac{\partial L(\theta)}{\partial(\theta)}\right) + v^{\text{old}} \cdot mtm \tag{15}$$

where mtm is the momentum factor. If the current gradient is parallel to the previous speed v^{old}, the previous speed can speed up this search. The proper momentum plays a role in accelerating the convergence when the learning rate is small. If the derivative decays to 0, it will continue to update v to reach equilibrium and will be attenuated by friction. It is beneficial to escape from the local minimum in the training process so that the search process can converge more quickly [101], [103]. If the current gradient is opposite than the previous update v^{old}, the value v^{old} will have a deceleration effect on this search.

The momentum method with a proper momentum factor plays a positive role in reducing the oscillation of convergence when the learning rate is large. How to select the proper size of the momentum factor is also a problem. If the momentum factor is small, it is hard to obtain the effect of improving convergence speed. If the momentum factor is large, the current point may jump out of the optimal value point. Many experiments have empirically verified the most appropriate setting for the momentum factor is 0.9 [90].

The Nesterov accelerated gradient descent (NAG) makes further improvement over the traditional momentum method [103], [104]. In Nesterov momentum, the momentum $v^{\text{old}} \cdot mtm$ is added to θ, denoted as $\widetilde{\theta}$. The gradient of $\widetilde{\theta}$ is used when updating. The detailed update formulas for parameters θ are as follows:

$$\begin{cases} \widetilde{\theta} \ = \theta + v^{\text{old}} \cdot mtm \\ v \ = v^{\text{old}} \cdot mtm + \eta \cdot \left(-\frac{\partial L(\widetilde{\theta})}{\partial(\theta)}\right) \\ \theta' = \theta + v. \end{cases} \tag{16}$$

The improvement of Nesterov momentum over momentum is reflected in updating the gradient of the future position instead of the current position. From the update formula, we can find that Nestorov momentum includes more gradient information compared with the traditional momentum method. Note that Nesterov momentum improves the convergence from $O(1/k)$ (after k steps) to $O(1/k^2)$, when not using stochastic optimization [104].

Another issue worth considering is how to determine the size of the learning rate. It is more likely to occur with the oscillation if the search is closer to the optimal point. Thus, the learning rate should be adjusted. The learning rate decay factor d is commonly used in the SGD's momentum method, which makes the learning rate decrease with the iteration period [105]. The formula of the learning rate decay is defined as

$$\eta_t = \frac{\eta_0}{1 + d \cdot t} \tag{17}$$

where η_t is the learning rate at the t-th iteration, η_0 is the original learning rate, and d is a decimal in $[0, 1]$. As can be seen from the formula, the smaller d is, the slower the decay of the learning rate will be. The learning rate remains unchanged when $d = 0$ and the learning rate decays fastest when $d = 1$.

4) Adaptive Learning Rate Method: The manually regulated learning rate greatly influences the effect of the SGD method. It is a tricky problem for setting an appropriate value of the learning rate [30], [33], [106]. Some adaptive methods were proposed to adjust the learning rate automatically. These methods are free of parameter adjustment, fast to converge, and often achieve not bad results. They are widely used in DNNs to deal with optimization problems.

The most straightforward improvement to SGD is AdaGrad [30]. AdaGrad adjusts the learning rate dynamically based on the historical gradient in some previous iterations. The update formulas are as follows:

$$\begin{cases} g_t = \frac{\partial L(\theta_t)}{\partial \theta} \\ V_t = \sqrt{\sum_{i=1}^{t} (g_i)^2 + \epsilon} \\ \theta_{t+1} = \theta_t - \eta \frac{g_t}{V_t} \end{cases} \tag{18}$$

where g_t is the gradient of parameter θ at iteration t, V_t is the accumulate historical gradient of parameter θ at iteration t, and θ_t is the value of parameter θ at iteration t.

The difference between AdaGrad and the gradient descent is that during the parameter update process, the learning rate is no longer fixed, but is computed using all historical gradients accumulated up to this iteration. One main benefit of AdaGrad is that it eliminates the need to tune the learning rate manually. Most implementations use a default value of 0.01 for η in (18).

Although AdaGrad adaptively adjusts the learning rate, it still has two issues: 1) the algorithm still needs to set the global learning rate η manually and 2) as the training time increases, the accumulated gradient will become larger and larger, making the learning rate tend to zero, resulting in an ineffective parameter update.

AdaGrad was further improved to AdaDelta [31] and RMSProp [32] for solving the problem that the learning rate will eventually go to zero. The idea is to consider not accumulating all historical gradients but focusing only on the gradients in a window over a period, and using the exponential moving average to calculate the second-order cumulative momentum

$$V_t = \sqrt{\beta V_{t-1} + (1-\beta)(g_t)^2} \tag{19}$$

where β is the exponential decay parameter. Both RMSProp and AdaDelta have been developed independently around the same time, stemming from the need to resolve the radically diminishing learning rates of AdaGrad.

Adaptive moment estimation (Adam) [33] is another advanced SGD method, which introduces an adaptive learning rate for each parameter. It combines the adaptive learning rate and momentum methods. In addition to storing an exponentially decaying average of past squared gradients V_t, like AdaDelta and RMSProp, Adam also keeps an exponentially decaying average of past gradients m_t, similar to the momentum method

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{20}$$

$$V_t = \sqrt{\beta_2 V_{t-1} + (1-\beta_2)(g_t)^2} \tag{21}$$

where β_1 and β_2 are the exponential decay rates. The final update formula for the parameter θ is

$$\theta_{t+1} = m_t - \eta \frac{\sqrt{1-\beta_2}}{1-\beta_1} \frac{m_t}{V_t + \epsilon}. \tag{22}$$

The default values of β_1, β_2, and ϵ are suggested to set to 0.9, 0.999, and 10^{-8}, respectively. Adam works well in practice and compares favorably to other adaptive learning rate algorithms.

5) Variance Reduction Methods: Due to a large amount of redundant information in the training samples, the SGD methods are very popular since they were proposed. However, the stochastic gradient method can only converge at a sublinear rate and the variance of gradient is often very large. How to reduce the variance and improve SGD to the linear convergence has always been an important problem.

Stochastic Average Gradient: The stochastic average gradient (SAG) method [36] is a variance reduction method proposed to improve the convergence speed. The SAG algorithm maintains parameter d recording the sum of the N latest gradients $\{g_i\}$ in memory, where g_i is calculated using one sample $i, i \in \{1, \ldots, N\}$. The detailed implementation is to select a sample i_t to update d randomly, and use d to update the parameter θ in iteration t

$$\begin{cases} d &= d - \hat{g}_{i_t} + g_{i_t}(\theta_{t-1}) \\ \hat{g}_{i_t} &= g_{i_t}(\theta_{t-1}) \\ \theta_t &= \theta_{t-1} - \frac{\alpha}{N} d \end{cases} \tag{23}$$

where the updated item d is calculated by replacing the old gradient $\hat{g}_{i_t}$ in d with the new gradient $g_{i_t}(\theta_{t-1})$ in iteration t,and α is a constant representing the learning rate. Thus, each update only needs to calculate the gradient of one sample, not the gradients of all samples. The computational overhead is no different from SGD, but the memory overhead is much larger. This is a typical way of using space for saving time. The SAG has been shown to be a linear convergence algorithm [36], which is much faster than SGD, and has great advantages over other stochastic gradient algorithms.

However, the SAG method is applicable only to the case where the loss function is smooth and the objective function is convex [36], [107], such as convex linear prediction problems. In this case, the SAG achieves a faster convergence rate than the SGD. In addition, under some specific problems, it can even deliver better convergence than the standard batch gradient descent.

Stochastic Variance Reduction Gradient: Since the SAG method is applicable only to smooth and convex functions and needs to store the gradient of each sample, it is inconvenient to be applied in the nonconvex neural networks. The stochastic variance reduction gradient (SVRG) [37] method was proposed to improve the performance of optimization in the complex models.

The algorithm of SVRG maintains the interval average gradient $\tilde{\mu}$ by calculating the gradients of all samples in every w iterations instead of in each iteration

$$\tilde{\mu} = \frac{1}{N} \sum_{i=1}^{N} g_i\left(\tilde{\theta}\right) \tag{24}$$

where $\tilde{\theta}$ is the interval update parameter. The interval parameter $\tilde{\mu}$ contains the average memory of all sample gradients in the past time for each time interval w. SVRG picks uniform $i_t \in \{1, \ldots, N\}$ randomly and executes gradient updates to the current parameters

$$\theta_t = \theta_{t-1} - \eta \cdot \left(g_{i_t}(\theta_{t-1}) - g_{i_t}\left(\tilde{\theta}\right) + \tilde{\mu}\right). \tag{25}$$

The gradient is calculated up to two times in each update. After w iterations, perform $\tilde{\theta} \leftarrow \theta_w$ and start the next w iterations. Through these updates, θ_t and the interval update parameter $\tilde{\theta}$ will converge to the optimal θ^*, and then $\tilde{\mu} \to 0$, and

$$g_{i_t}(\theta_{t-1}) - g_{i_t}\left(\tilde{\theta}\right) + \tilde{\mu} \to g_{i_t}(\theta_{t-1}) - g_{i_t}\left(\theta^*\right) \to 0. \tag{26}$$

SVRG proposes a vital concept called variance reduction. This concept is related to the convergence analysis of SGD, in which it is necessary to assume that there is a constant upper bound for the variance of the gradients. This constant upper bound implies that the SGD cannot achieve linear convergence. However, in SVRG, the upper bound of variance can be continuously reduced due to the special update item $g_{i_t}(\theta_{t-1}) - g_{i_t}(\tilde{\theta}) + \tilde{\mu}$, thus achieving linear convergence [37].

The strategies of SAG and SVRG are related to variance reduction. Compared with SAG, SVRG does not need to maintain all gradients in memory, which means that memory resources are saved, and it can be applied to complex problems efficiently. Experiments have shown that the performance of SVRG is remarkable on a nonconvex neural network [37], [108], [109]. There are also many variants of such linear convergence stochastic optimization algorithms, such as the SAGA algorithm [110].

6) Alternating Direction Method of Multipliers: The augmented Lagrangian multiplier method is a common method to solve optimization problems with linear constraints. Compared with the naive Lagrangian multiplier method, it makes problems easier to solve by adding a penalty term to the objective. Consider the following example:

$$\min\{\theta_1(x) + \theta_2(y) | Ax + By = b, x \in \mathcal{X}, y \in \mathcal{Y}\}. \tag{27}$$

The augmented Lagrange function for problem (27) is

$$\mathcal{L}_\beta(x,y,\lambda) = \theta_1(x) + \theta_2(y) - \lambda^\top(Ax+By-b) + \frac{\beta}{2}\|Ax+By-b\|^2. \tag{28}$$

When solved by the augmented Lagrangian multiplier method, its t-th step iteration starts from the given λ_t, and the optimization turns out to be

$$\begin{cases} (x_{t+1}, y_{t+1}) = \arg\min\left\{\mathcal{L}_\beta(x,y,\lambda_t) | x \in \mathcal{X}, y \in \mathcal{Y}\right\} \\ \lambda_{t+1} \qquad\quad = \lambda_t - \beta(Ax_{t+1} + By_{t+1} - b). \end{cases} \tag{29}$$

Separating the (x, y) subproblem in (29), the augmented Lagrange multiplier method can be relaxed to the following ADMMs [111], [112]. Its t-th step iteration starts with the given (y_t, λ_t), and the details of iterative optimization are as follows:

$$\begin{cases} x_{t+1} = \arg\min\left\{\theta_1(x) - (\lambda_t)^\top Ax + \frac{\beta}{2}\|\mathrm{Con}_x\|^2 | x \in \mathcal{X}\right\} \\ y_{t+1} = \arg\min\left\{\theta_2(y) - (\lambda_t)^\top By + \frac{\beta}{2}\left\|\mathrm{Con}_y\right\|^2 | y \in \mathcal{Y}\right\} \\ \lambda_{t+1} = \lambda_t - \beta(Ax_{t+1} + By_{t+1} - b) \end{cases} \tag{30}$$

where $\mathrm{Con}_x = Ax + By_t - b$ and $\mathrm{Con}_y = Ax_{t+1} + By - b$.

The penalty parameter β has a certain impact on the convergence rate of the ADMM. The larger β is, the greater the penalties for the constraint term. In general, a monotonically increasing sequence of $\{\beta_t\}$ can be adopted instead of the fixed β [113]. Specifically, an auto-adjustment criterion that automatically adjusts $\{\beta_t\}$ based on the current value of $\{x_t\}$ during the iteration was proposed and applied for solving some convex optimization problems [114], [115].

The ADMM method uses the separable operators in the convex optimization problem to divide a large problem into multiple small problems that can be solved in a distributed manner. In theory, the framework of ADMM can solve most of the large-scale optimization problems. However, there are still some problems in practical applications. For example, if we use a stop criterion to determine whether convergence occurs, the original residuals and dual residuals are both related to β, and β with a large value will lead to difficulty in meeting the convergence conditions [116].

7) Frank–Wolfe Method: In 1956, Frank and Wolfe proposed an algorithm to solve the linear constraint problems [117]. The basic idea is to approximate the objective function with a linear function, then solve the linear programming to find the feasible descending direction and, finally, make a 1-D search along the direction in the feasible domain. This method is also called the approximate linearization method.

Here, we give a simple example of the Frank–Wolfe method. Consider the optimization problem

$$\begin{cases} \min & f(x) \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{cases} \tag{31}$$

where A is an $m \times n$ full-row rank matrix, and the feasible region is $S = \{x | Ax = b, x \geq 0\}$. Expand $f(x)$ linearly at x_0, $f(x) \approx f(x_0) + \nabla f(x_0)^\top(x - x_0)$, and substitute it into (31). Then, we have

$$\begin{cases} \min & f(x_t) + \nabla f(x_t)^\top(x - x_t) \\ \text{s.t.} & x \in S \end{cases} \tag{32}$$

which is equivalent to

$$\begin{cases} \min & \nabla f(x_t)^\top x \\ \text{s.t.} & x \in S. \end{cases} \tag{33}$$

Suppose there exist an optimal solution y_t, and then there must be

$$\begin{cases} \nabla f(x_t)^\top y_t < \nabla f(x_t)^\top x_t \\ \nabla f(x_t)^\top(y_t - x_t) < 0. \end{cases} \tag{34}$$

So $y_t - x_t$ is the decreasing direction of $f(x)$ at x_t. A fetch step of λ_t updates the search point in a feasible direction. The detailed operation is shown in Algorithm 1.

Algorithm 1 Frank–Wolfe Method [117], [118]

```
Input: $x_0$, $\varepsilon \geq 0$, $t := 0$
Output: $x^*$
  $y_t \leftarrow \min \nabla f(x_t)^\top x$
  while $|\nabla f(x_t)^\top(y_t - x_t)| > \varepsilon$ do
      $\lambda_t = \arg\min_{0 \leq \lambda \leq 1} f(x_t + \lambda(y_t - x_t))$
      $x_{t+1} \approx x_t + \lambda_t(y_t - x_t)$
      $t := t + 1$
      $y_t \leftarrow \min \nabla f(x_t)^\top x$
  end while
  $x^* \approx x_t$
```

The algorithm satisfies the following convergence theorem [117]:

1) x_t is the Kuhn–Tucker point of (31) when $\nabla f(x_t)^\top(y_t - x_t) = 0$;
2) since y_t is an optimal solution for problem (33), the vector d_t satisfies $d_t = y_t - x_t$ and is the feasible descending direction of f at point x_t when $\nabla f(x_t)^\top(y_t - x_t) \neq 0$.

The Frank–Wolfe algorithm is a first-order iterative method to solve the convex optimization problems with constrained conditions. It consists of determining the feasible descent direction and calculating the search step size. The algorithm is characterized by fast convergence in early iterations and slower in later phases. When the iterative point is close to the optimal solution, the search direction and the gradient direction of the objective function tend to be orthogonal. Such a direction is not the best downward direction so that the Frank–Wolfe algorithm can be improved and extended in terms of the selection of the descending directions [119]–[121].

8) Summary: We summarize the above-mentioned first-order optimization methods in terms of properties, advantages, and disadvantages in Table I in the supplementary material.

B. Preconditioning in Optimization

Preconditioning is a very important technique in optimization methods. Reasonable preconditioning can reduce the iteration number of optimization algorithms. For many important iterative methods, the convergence depends largely on the spectral properties of the coefficient

TABLE I
AVAILABLE TOOLKITS FOR OPTIMIZATION

Toolkit	Language	Description
CVX [124]	Matlab	CVX is a matlab-based modeling system for convex optimization but cannot handle large-scale problems. http://cvxr.com/cvx/download/
CVXPY [125]	Python	CVXPY is a python package developed by Stanford University Convex Optimization Group for solving convex optimization problems. http://www.cvxpy.org/
CVXOPT [126]	Python	CVXOPT can be used for handling convex optimization. It is developed by Martin Andersen, Joachim Dahl, and Lieven Vandenberghe. http://cvxopt.org/
APM [127]	Python	APM python is suitable for large-scale optimization and can solve the problems of linear programming, quadratic programming, integer programming, nonlinear optimization and so on. http://apmonitor.com/wiki/index.php/Main/PythonApp
SPAMS [128]	C++	SPAMS is an optimization toolbox for solving various sparse estimation problems, which is developed and maintained by Julien Mairal. Available interfaces include matlab, R, python and C++. http://spams-devel.gforge.inria.fr/
minConf	Matlab	minConf can be used for optimizing differentiable multivariate functions which subject to simple constraints on parameters. It is a set of matlab functions, in which there are many methods to choose from. https://www.cs.ubc.ca/ schmidtm/Software/minConf.html
tf.train.optimizer [129]	Python; C++; CUDA	The basic optimization class, which is usually not called directly and its subclasses are often used. It includes classic optimization algorithms such as gradient descent and AdaGrad. https://www.tensorflow.org/api_guides/python/train

matrix [122]. It can be simply considered that the pretreatment is to transform a difficult linear system $A\theta = b$ into an equivalent system with the same solution but better spectral characteristics. For example, if M is a nonsingular approximation of the coefficient matrix A, the transformed system

$$M^{-1}A\theta = M^{-1}b \tag{35}$$

will have the same solution as the system $A\theta = b$. But (35) may be easier to solve and the spectral properties of the coefficient matrix $M^{-1}A$ may be more favorable.

In most linear systems, for example, $A\theta = b$, the matrix A is often complex and makes it hard to solve the system. Therefore, some transformation is needed to simplify this system. M is called the preconditioner. If the matrix after using preconditioner is obviously structured, or sparse, it will be beneficial to the calculation [123].

The conjugate gradient algorithm is the most commonly used optimization method with preconditioning technology, which speeds up the convergence. The algorithm is shown in Algorithm 2.

Algorithm 2 Preconditioned Conjugate Gradient Method [93]

```
Input: $A$, $\theta_0$, $M$, $b$
Output: The solution $\theta^*$
  $f_0 = f(\theta_0)$
  $g_0 = \nabla f(\theta_0) = A\theta_0 - b$
  $y_0$ is the solution of $My = g_0$
  $d_0 = -g_0$
  $t = t$
  while $g_t \neq 0$ do
      $\eta_t = \frac{g_t^\top y_t}{d_t^\top A d_t}$
      $\theta_{t+1} = \theta_t + \eta_t d_t$
      $g_{t+1} = g_t + \eta_t A d_t$
      $y_{t+1}$ =solution of $My = g_t$
      $\beta_{t+1} = \frac{g_{t+1}^\top y_{t+1}}{g_t^\top d_t}$
      $d_{t+1} = -y_{t+1} + \beta_{t+1} d_t$
      $t = t + 1$
  end while
```

C. Public Toolkits for Optimization

The fundamental optimization methods are applied in machine-learning problems extensively. There are many integrated powerful toolkits. We summarize the existing common optimization toolkits and present them in Table I.

IV. CHALLENGES AND OPEN PROBLEMS

With the rise of practical demand and the increase of the complexity of the machine-learning models, the optimization methods in machine learning still face challenges. In this part, we discuss open problems and challenges for some optimization methods in machine learning, which may offer suggestions or ideas for future research and promote the wider application of optimization methods in machine learning.

A. Challenges in Deep Neural Networks

There are still many challenges while optimizing DNNs. Here, we mainly discuss two challenges with respect to data and model. One is insufficient data in training, and the other is a nonconvex objective in DNNs.

1) Insufficient Data in Training Deep Neural Networks: In general, deep learning is based on big datasets and complex models. It requires a large number of training samples to achieve good training effects. But in some particular fields, finding a sufficient amount of training data is difficult. If we do not have enough data to estimate the parameters in the neural networks, it may lead to high variance and overfitting.

There are some techniques in neural networks that can be used to reduce the variance. Adding L_2 regularization to the objective is a natural method to reduce the model complexity. Recently, a common method is dropout [62]. In the training process, each neuron is allowed to stop working with a probability of p, which can prevent the synergy between certain neurons. M subnets can be sampled like bagging by multiple inputs and returns [130]. Each expected result at the output layer is calculated as

$$o = \mathbb{E}_M\big[f(x;\theta,M)\big] = \sum_{i=1}^{M} p(M_i) f(x;\theta,M_i) \tag{36}$$

where $p(M_i)$ is the probability of the ith subnet. Dropout can prevent overfitting and improve the generalization ability of the network, but its disadvantage is increasing the training time as each training changes from the full network to a subnetwork [131].

Not only overfitting but also some training details will affect the performance of the model due to the complexity of the DNNs. The improper selection of the learning rate and the number of iterations in the SGD will make the model unable to converge, which makes the accuracy of model fluctuate greatly. Besides, taking an inappropriate black box of neural-network construction may result in training not being able to continue, so designing an appropriate neural-network model is particularly important. These impacts are even greater when data are insufficient.

The technology of transfer learning [132] can be applied to build networks in the scenario of insufficient data. Its idea is that the models trained from other data sources can be reused in similar target fields after certain modifications and improvements, which dramatically alleviates the problems caused by insufficient datasets. Moreover, the advantages brought by transfer learning are not limited to reducing the need for sufficient training data but also can avoid overfitting effectively and achieve better performance in general. However, if target data are not as relevant to the original training data, the transferred model does not bring good performance.

Meta learning methods can be used for systematically learning parameter initialization, which ensures that training begins with a suitable initial model. However, it is necessary to ensure the correlation between multiple tasks for meta-training and tasks for meta-testing. Under the premise of models with similar data sources for training, transfer learning and meta learning can overcome the difficulties caused by insufficient training data in new data sources, but these methods usually introduce a large number of parameters or complex parameter adjustment mechanisms, which need to be further improved for specific problems. Therefore, using insufficient data for training DNNs is still a challenge.

2) Nonconvex Optimization in the Deep Neural Network: Convex optimization has good properties and a comprehensive set of tools is open to solve the optimization problem. However, many machine-learning problems are formulated as the nonconvex optimization problems. For example, almost all of the optimization problems in DNNs are nonconvex. Nonconvex optimization is one of the difficulties in the optimization problem. Unlike convex optimization, there may be innumerable optimum solutions in its feasible domain in nonconvex problems. The complexity of the algorithm for searching the global optimal value is NP-hard [108].

In recent years, nonconvex optimization has gradually attracted the attention of researchers. The methods for solving nonconvex optimization problems can be roughly divided into two types. One is to transform the nonconvex optimization into a convex optimization problem, and then use the convex optimization method. The other is to use some special optimization method for solving nonconvex functions directly. There is some work on summarizing the optimization methods for solving nonconvex functions from the perspective of machine learning [133].

1) *Relaxation Method:* Relax the problem to make it become a convex optimization problem. There are many relaxation techniques, for example, the branch-and-bound method called αBB convex relaxation [134], [135], which uses a convex relaxation at each step to compute the lower bound in the region. The convex relaxation method has been used in many fields. In the field of computer vision, a convex relaxation method was proposed to calculate minimal partitions [136]. For unsupervised and SSL, the convex relaxation method was used for solving semidefinite programming [137].
2) *Nonconvex Optimization Methods:* These methods include projection gradient descent [138], [139]; alternating minimization [140]–[142]; expectation maximization algorithm [143], [144]; and stochastic optimization and its variants [37].

B. Difficulties in Sequential Models With Large-Scale Data

When dealing with the large-scale time series, the usual solutions are using stochastic optimization, processing data in mini-batches, or utilizing distributed computing to improve computational efficiency [145]. For a sequential model, segmenting the sequences can affect the dependencies between the data on the adjacent time indices. If sequence length is not an integral multiple of the mini-batch size, the general operation is to add some items sampled from the previous data into the last subsequence. This operation will introduce the wrong dependency in the training model. Therefore, the analysis of the difference between the approximated solution obtained and the exact solution is a direction worth exploring.

Particularly, in RNNs, the problem of gradient vanishing and gradient explosion is also prone to occur. So far, it is generally solved by specific interaction modes of LSTM and GRU [146] or gradient clipping. Better appropriate solutions for dealing with problems in RNNs are still worth investigating.

C. High-Order Methods for Stochastic Variational Inference

The high-order optimization method utilizes curvature information and thus converges fast. Although computing and storing the Hessian matrices are difficult, with the development of research, the calculation of the Hessian matrix has made great progress [8], [9], [147], and the second-order optimization method has become more and more attractive. Recently, stochastic methods have also been introduced into the second-order method, which extends the second-order method to large-scale data [8], [10].

We have introduced some work on stochastic variational inference. It introduces the stochastic method into variational inference, which is an interesting and meaningful combination. This makes variational inference be able to handle large-scale data. A natural idea is whether we can incorporate second-order optimization methods (or higher-order) into stochastic variational inference, which is interesting and challenging.

D. Stochastic Optimization in Conjugate Gradient

Stochastic methods exhibit powerful capabilities when dealing with large-scale data, especially for first-order optimization [148]. Then, the relevant experts and scholars also introduced this stochastic idea to the second-order optimization methods [149]–[151] and achieved good results.

The conjugate gradient method is an elegant and attractive algorithm, which has the advantages of both the first-order and second-order optimization methods. The standard form of a conjugate gradient is not suitable for a stochastic approximation. Using the fast Hessian-gradient product, the stochastic method is also introduced to conjugate gradient, in which some numerical results show the validity of the algorithm [148]. Another version of the stochastic conjugate gradient method employs the variance reduction technique and converges quickly with just a few iterations and requires less storage space during the running process [152]. The stochastic version of conjugate gradient is a potential optimization method and is still worth studying.

V. Conclusion

This article introduced and summarized the frequently used optimization methods from the perspective of machine learning, and studied their applications in various fields of machine learning. First, we described the theoretical basis of optimization methods from the first-order, high-order, and derivative-free aspects, as well as the research progress in recent years. Then, we described the applications of the optimization methods in different machine-learning scenarios and the approaches to improve their performance in the supplementary material. Finally, we discussed some challenges and open problems in the machine-learning optimization methods.

References

[1] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Stat.*, vol. 22, no. 3, pp. 400–407, 1951.

[2] P. Jain, S. Kakade, R. Kidambi, P. Netrapalli, and A. Sidford, "Parallelizing stochastic gradient descent for least squares regression: Mini-batching, averaging, and model misspecification," *J. Mach. Learn. Res.*, vol. 18, no. 223, pp. 1–42, 2018.

[3] D. F. Shanno, "Conditioning of quasi-Newton methods for function minimization," *Math. Comput.*, vol. 24, no. 111, pp. 647–656, 1970.

[4] J. Hu, B. Jiang, L. Lin, Z. Wen, and Y.-X. Yuan, "Structured quasi-Newton methods for optimization with orthogonality constraints," *SIAM J. Sci. Comput.*, vol. 41, pp. 2239–2269, 2019.

[5] J. Pajarinen, H. L. Thai, R. Akrour, J. Peters, and G. Neumann, "Compatible natural gradient policy search," *Mach. Learn.*, vol. 108, nos. 8–9, pp. 1443–1466, 2019.

[6] J. E. Dennis, Jr., and J. J. Moré, "Quasi-Newton methods, motivation and theory," *SIAM Rev.*, vol. 19, no. 1, pp. 46–89, 1977.

[7] J. Martens, "Deep learning via Hessian-free optimization," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 735–742.

[8] F. Roosta-Khorasani and M. W. Mahoney, "Sub-sampled Newton methods II: Local convergence rates," *arXiv preprint arXiv:1601.04738*, pp. 1–56, Feb. 2016.

[9] P. Xu, J. Yang, F. Roosta-Khorasani, C. Ré, and M. W. Mahoney, "Sub-sampled Newton methods with non-uniform sampling," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3000–3008.

[10] R. Bollapragada, R. H. Byrd, and J. Nocedal, "Exact and inexact subsampled Newton methods for optimization," *IMA J. Numer. Anal.*, vol. 1, no. 2, pp. 1–34, 2018.

[11] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: A review of algorithms and comparison of software implementations," *J. Glob. Optim.*, vol. 56, no. 3, pp. 1247–1293, 2013.

[12] A. S. Berahas, R. H. Byrd, and J. Nocedal, "Derivative-free optimization of noisy functions via quasi-Newton methods," *SIAM J. Optim.*, vol. 29, no. 2, pp. 965–993, 2019.

[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

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

[15] P. Sermanet and D. Eigen, "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *Proc. Int. Conf. Learn. Represent.*, 2014, pp. 1–16.

[16] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Columbus, OH, USA, 2014, pp. 1725–1732.

[17] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. Conf. Empir. Methods Nat. Lang. Process.*, 2014, pp. 1746–1751.

[18] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, Jan. 2012.

[19] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Proc. Assoc. Adv. Artif. Intell.*, 2015, pp. 2267–2273.

[20] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empir. Methods Nat. Lang. Process.*, 2014, pp. 1724–1734.

[21] P. Liu, X. Qiu, and X. Huang, "Recurrent neural network for text classification with multi-task learning," in *Proc. Int. Joint Conf. Artif. Intell.*, 2016, pp. 2873–2879.

[22] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. Int. Conf. Acoust. Speech Signal Process.*, Vancouver, BC, Canada, 2013, pp. 6645–6649.

[23] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "DRAW: A recurrent neural network for image generation," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1462–1471.

[24] A. V. D. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1747–1756.

[25] A. Ullah, J. Ahmad, K. Muhammad, M. Sajjad, and S. W. Baik, "Action recognition in video sequences using deep bi-directional LSTM with CNN features," *IEEE Access*, vol. 6, pp. 1155–1166, 2017.

[26] Y. Xia and J. Wang, "A bi-projection neural network for solving constrained quadratic optimization problems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 2, pp. 214–224, Feb. 2015.

[27] S. Zhang, Y. Xia, and J. Wang, "A complex-valued projection neural network for constrained optimization of real functions in complex variables," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 12, pp. 3227–3238, Dec. 2015.

[28] Y. Xia and J. Wang, "Robust regression estimation based on low-dimensional recurrent neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 12, pp. 5935–5946, Dec. 2018.

[29] Y. Xia, J. Wang, and W. Guo, "Two projection neural networks with reduced model complexity for nonlinear programming," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published.

[30] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Jul. 2011.

[31] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, pp. 1–6, Dec. 2012.

[32] T. Tieleman and G. Hinton, "Divide the gradient by a running average of its recent magnitude," *COURSERA Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, 2014, pp. 1–15.

[34] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and beyond," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–23.

[35] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, pp. 1–16, Jan. 2015.

[36] N. L. Roux, M. Schmidt, and F. R. Bach, "A stochastic gradient method with an exponential convergence rate for finite training sets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 2663–2671.

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

[38] N. S. Keskar and R. Socher, "Improving generalization performance by switching from Adam to SGD," *arXiv preprint arXiv:1712.07628*, pp. 1–10, Dec. 2017.

[39] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[40] J. Mattner, S. Lange, and M. Riedmiller, "Learn to swing up and balance a real pole based on raw visual input data," in *Proc. Int. Conf. Neural Inf. Process.*, 2012, pp. 126–133.

[41] V. Mnih *et al.*, "Playing Atari with deep reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst. Workshop*, 2013, pp. 1–9.

[42] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[43] Y. Bengio, "Learning deep architectures for AI," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.

[44] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: An overview," in *Proc. SAI Intell. Syst. Conf.*, 2016, pp. 426–440.

[45] J. Schmidhuber, "Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-… hook," Ph.D. dissertation, Institut für Informatik, Technische Universität München, Munich, Germany, 1987.

[46] T. Schaul and J. Schmidhuber, "Metalearning," *Scholarpedia*, vol. 5, no. 6, pp. 46–50, 2010.

[47] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.

[48] O. Vinyals. (2017). *Model vs Optimization Meta Learning*. [Online]. Available: http://metalearning-symposium.ml/files/vinyals.pdf

[49] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a"siamese" time delay neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, Denver, CO, USA, 1994, pp. 737–744.

[50] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *Proc. Int. Conf. Mach. Learn. WorkShop*, 2015, pp. 1–30.

[51] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3630–3638.

[52] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4077–4087.

[53] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proc. Int. Conf. Mach. Learn.*, New York, NY, USA, 2016, pp. 1842–1850.

[54] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–15.

[55] M. Andrychowicz *et al.*, "Learning to learn by gradient descent by gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3981–3989.

[56] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–11.

[57] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[58] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *J. Mach. Learn. Res.*, vol. 14, pp. 1303–1347, May 2013.

[59] C. Ledig *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proc. Comput. Vis. Pattern Recognit.*, 2017, pp. 4681–4690.

[60] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5279–5288.

[61] T. Chen, E. Fox, and C. Guestrin, "Stochastic gradient Hamiltonian Monte Carlo," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1683–1691.

[62] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jun. 2014.

[63] W. Yin and H. Schütze, "Multichannel variable-size convolution for sentence classification," in *Proc. Conf. Comput. Lang. Learn.*, 2015, pp. 204–214.

[64] J. Yang, K. Yu, Y. Gong, and T. S. Huang, "Linear spatial pyramid matching using sparse coding for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Miami, FL, USA, 2009, pp. 1794–1801.

[65] Y. Bazi and F. Melgani, "Gaussian process approach to remote sensing image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 48, no. 1, pp. 186–197, Jan. 2010.

[66] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 3642–3649.

[67] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A k-means clustering algorithm," *J. Roy. Stat. Soc. C (Appl. Stat.)*, vol. 28, no. 1, pp. 100–108, 1979.

[68] S. Guha, R. Rastogi, and K. Shim, "ROCK: A robust clustering algorithm for categorical attributes," *Inf. Syst.*, vol. 25, no. 5, pp. 345–366, 2000.

[69] C. Ding, X. He, H. Zha, and H. D. Simon, "Adaptive dimension reduction for clustering high dimensional data," in *Proc. IEEE Int. Conf. Data Min.*, 2002, pp. 147–154.

[70] M. Guillaumin, J. Verbeek, and C. Schmid, "Multimodal semi-supervised learning for image classification," in *Proc. Comput. Vis. Pattern Recognit.*, San Francisco, CA, USA, 2010, pp. 902–909.

[71] O. Chapelle and A. Zien, "Semi-supervised classification by low density separation," in *Proc. Int. Conf. Artif. Intell. Stat.*, 2005, pp. 57–64.

[72] Z.-H. Zhou and M. Li, "Semi-supervised regression with co-training," in *Proc. Int. Joint Conf. Artif. Intell.*, 2005, pp. 908–913.

[73] A. Demiriz and K. P. Bennett, "Semi-supervised clustering using genetic algorithms," *Artif. Neural Netw. Eng.*, vol. 1, pp. 809–814, Sep. 1999.

[74] B. Kulis, S. Basu, I. Dhillon, and R. Mooney, "Semi-supervised graph clustering: A kernel approach," *Mach. Learn.*, vol. 74, no. 1, pp. 1–22, 2009.

[75] D. Zhang and Z.-H. Zhou, "Semi-supervised dimensionality reduction," in *Proc. SIAM Int. Conf. Data Min.*, 2007, pp. 629–634.

[76] P. Chen, L. Jiao, F. Liu, J. Zhao, Z. Zhao, and S. Liu, "Semi-supervised double sparse graphs based discriminant analysis for dimensionality reduction," *Pattern Recognit.*, vol. 61, pp. 361–378, Jan. 2017.

[77] K. P. Bennett and A. Demiriz, "Semi-supervised support vector machines," in *Proc. Adv. Neural Inf. Process. Syst.*, 1999, pp. 368–374.
[78] E. Cheung, *Optimization Methods for Semi-Supervised Learning*. Waterloo, ON, Canada: Univ. Waterloo, 2018.
[79] O. Chapelle, V. Sindhwani, and S. S. Keerthi, "Optimization techniques for semi-supervised support vector machines," *J. Mach. Learn. Res.*, vol. 9, pp. 203–233, Jun. 2008.
[80] O. Chapelle, V. Sindhwani, and S. S. Keerthi, "Branch and bound for semi-supervised support vector machines," in *Proc. Adv. Neural Inf. Process. Syst.*, 2007, pp. 217–224.
[81] Y.-F. Li, I. W. Tsang, J. T. Kwok, and Z.-H. Zhou, "Convex and scalable weakly labeled SVMs," *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 2151–2188, 2013.
[82] F. Murtagh, "A survey of recent advances in hierarchical clustering algorithms," *Comput. J.*, vol. 26, no. 4, pp. 354–359, 1983.
[83] V. Estivill-Castro and J. Yang, "A fast and robust general purpose clustering algorithm," in *Proc. 4th Eur. Workshop Principles Knowl. Disc. Databases Data Min.*, 2000, pp. 208–218.
[84] G. H. Ball and D. J. Hall, "A clustering technique for summarizing multivariate data," *Behav. Sci.*, vol. 12, no. 2, pp. 153–155, 1967.
[85] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometr. Intell. Lab. Syst.*, vol. 2, nos. 1–3, pp. 37–52, 1987.
[86] I. T. Jolliffe, "Principal component analysis," in *International Encyclopedia of Statistical Science*. Heidelberg, Germany: Springer, 2011, pp. 1094–1096.
[87] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *J. Roy. Stat. Soc. B (Stat. Methodol.)*, vol. 61, no. 3, pp. 611–622, 1999.
[88] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[89] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, Mar. 1996.
[90] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, pp. 1–14, Jun. 2016.
[91] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[92] J. Alspector, R. Meir, B. Yuhas, A. Jayakumar, and D. Lippe, "A parallel gradient descent method for learning in analog VLSI neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 1993, pp. 836–844.
[93] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY, USA: Springer, 2006.
[94] A. S. Nemirovsky and D. B. Yudin, *Problem Complexity and Method Efficiency in Optimization*. Chichester, U.K.: Wiley, 1983.
[95] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM J. Optim.*, vol. 19, no. 4, pp. 1574–1609, 2009.
[96] A. Agarwal, M. J. Wainwright, P. L. Bartlett, and P. K. Ravikumar, "Information-theoretic lower bounds on the oracle complexity of convex optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 1–9.
[97] C. Darken, J. Chang, and J. Moody, "Learning rate schedules for faster stochastic gradient search," in *Proc. Neural Netw. Signal Process.*, 1992, pp. 3–12.
[98] I. Sutskever, "Training recurrent neural networks," Ph.D. dissertation, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2013.
[99] Z. Allen-Zhu, "Natasha 2: Faster non-convex optimization than SGD," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 2675–2686.
[100] R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping from saddle points—Online stochastic gradient for tensor decomposition," in *Proc. Conf. Learn. Theory*, 2015, pp. 797–842.
[101] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Comput. Math. Math. Phys.*, vol. 4, no. 5, pp. 1–17, 1964.
[102] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[103] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1139–1147.
[104] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence $O(\frac{1}{k^2})$," *Doklady Akademii Nauk SSSR*, vol. 269, no. 3, pp. 543–547, 1983.
[105] L. C. Baird, III, and A. W. Moore, "Gradient descent for general reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 1999, pp. 968–974.
[106] C. Darken and J. E. Moody, "Note on learning rate schedules for stochastic optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 1991, pp. 832–838.
[107] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient," *Math. Program.*, vol. 162, nos. 1–2, pp. 83–112, 2017.
[108] Z. Allen-Zhu and E. Hazan, "Variance reduction for faster non-convex optimization," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 699–707.
[109] S. J. Reddi, A. Hefny, S. Sra, B. Póczós, and A. Smola, "Stochastic variance reduction for nonconvex optimization," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 314–323.
[110] A. Defazio, F. Bach, and S. Lacoste-Julien, "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 1646–1654.
[111] M. J. Powell, "A method for nonlinear constraints in minimization problems," in *Optimization*. New York, NY, USA: Academic, 1969, pp. 283–298.
[112] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.
[113] A. Nagurney and P. Ramanujam, "Transportation network policy modeling with goal targets and generalized penalty functions," *Transport. Sci.*, vol. 30, pp. 3–13, 1996.
[114] B. He, H. Yang, and S. Wang, "Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities," *J. Optim. Theory Appl.*, vol. 106, no. 1, pp. 337–356, 2000.
[115] D. Hallac, C. Wong, S. Diamond, A. Sharang, S. Boyd, and J. Leskovec, "SnapVX: A network-based convex optimization solver," *J. Mach. Learn. Res.*, vol. 18, nos. 1–5, pp. 1–5, 2017.
[116] B. Wahlberg, S. Boyd, M. Annergren, and Y. Wang, "An ADMM algorithm for a class of total variation regularized estimation problems," *IFAC Proc. Vol.*, vol. 45, no. 16, pp. 83–88, 2012.
[117] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *Naval Res. Logistics Quart.*, vol. 3, nos. 1–2, pp. 95–110, 1956.
[118] M. Jaggi, "Revisiting Frank–Wolfe: Projection-free sparse convex optimization," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 427–435.
[119] M. Fukushima, "A modified Frank–Wolfe algorithm for solving the traffic assignment problem," *Transport. Res. B Methodol.*, vol. 18, no. 2, pp. 169–177, 1984.
[120] M. Patriksson, *The Traffic Assignment Problem: Models and Methods*. Mineola, NY, USA: Dover, 2015.
[121] K. L. Clarkson, "Coresets, sparse greedy approximation, and the Frank–Wolfe algorithm," *ACM Trans. Algorithms*, vol. 6, no. 4, pp. 63–96, 2010.
[122] T. Huckle, "Approximate sparsity patterns for the inverse of a matrix and preconditioning," *Appl. Numer. Math.*, vol. 30, nos. 2–3, pp. 291–303, 1999.
[123] M. Benzi, "Preconditioning techniques for large linear systems: A survey," *J. Comput. Phys.*, vol. 182, no. 2, pp. 418–477, 2002.
[124] M. Grant and S. Boyd. (2014). *CVX: MATLAB Software for Disciplined Convex Programming, Version 2.1*. [Online]. Available: http://cvxr.com/cvx
[125] S. Diamond and S. Boyd, "CVXPY: A python-embedded modeling language for convex optimization," *J. Mach. Learn. Res.*, vol. 17, pp. 2909–2913, Apr. 2016.
[126] M. Andersen, J. Dahl, and L. Vandenberghe. (2013). *CVXOPT: A Python Package for Convex Optimization, Version 1.1.6*. [Online]. Available: https://cvxopt.org/
[127] J. D. Hedengren, R. A. Shishavan, K. M. Powell, and T. F. Edgar, "Nonlinear modeling, estimation and predictive control in APMonitor," *Comput. Chem. Eng.*, vol. 70, pp. 133–148, Nov. 2014.
[128] J. Mairal, F. Bach, J. Ponce, G. Sapiro, R. Jenatton, and G. Obozinski. (2014). *SPAMS: A Sparse Modeling Software, Version 2.3*. [Online]. Available: http://spams-devel.gforge.inria.fr/downloads.html
[129] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. USENIX Symp. Oper. Syst. Design Implement.*, 2016, pp. 265–283.
[130] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.
[131] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, pp. 1–8, Feb. 2015.
[132] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.
[133] P. Jain and P. Kar, "Non-convex optimization for machine learning," *Found. Trends Mach. Learn.*, vol. 10, pp. 142–336, Dec. 2017.

[134] C. S. Adjiman and S. Dallwig, "A global optimization method, αBB, for general twice-differentiable constrained NLPs—I. Theoretical advances," *Comput. Chem. Eng.*, vol. 22, no. 9, pp. 1137–1158, 1998.
[135] C. S. Adjiman, C. A. Schweiger, and C. A. Floudas, "Mixed-integer nonlinear optimization in process synthesis," in *Handbook of Combinatorial Optimization*. Boston, MA, USA: Springer, 1998, pp. 1–76.
[136] T. Pock, A. Chambolle, D. Cremers, and H. Bischof, "A convex relaxation approach for computing minimal partitions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 810–817.
[137] L. Xu and D. Schuurmans, "Unsupervised and semi-supervised multi-class support vector machines," in *Proc. Assoc. Adv. Artif. Intell.*, 2005, pp. 904–910.
[138] Y. Chen and M. J. Wainwright, "Fast low-rank estimation by projected gradient descent: General statistical and algorithmic guarantees," *arXiv preprint arXiv:1509.03025*, pp. 1–63, Sep. 2015.
[139] D. Park, A. Kyrillidis, S. Bhojanapalli, C. Caramanis, and S. Sanghavi, "Provable non-convex projected gradient descent for a class of constrained matrix optimization problems," *arXiv preprint arXiv:1606.01316*, pp. 1–29, Jun. 2016.
[140] P. Jain, P. Netrapalli, and S. Sanghavi, "Low-rank matrix completion using alternating minimization," in *Proc. ACM Annu. Symp. Theory Comput.*, 2013, pp. 665–674.
[141] M. Hardt, "Understanding alternating minimization for matrix completion," in *Proc. IEEE Annu. Symp. Found. Comput. Sci.*, 2014, pp. 651–660.
[142] M. Hardt and M. Wootters, "Fast matrix completion without the condition number," in *Proc. Conf. Learn. Theory*, 2014, pp. 638–678.
[143] S. Balakrishnan, M. J. Wainwright, and B. Yu, "Statistical guarantees for the EM algorithm: From population to sample-based analysis," *Ann. Stat.*, vol. 45, no. 1, pp. 77–120, 2017.
[144] Z. Wang, Q. Gu, Y. Ning, and H. Liu, "High dimensional expectation maximization algorithm: Statistical optimization and asymptotic normality," *arXiv preprint arXiv:1412.8729*, pp. 1–84, Dec. 2014.
[145] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–16.
[146] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *Proc. Adv. Neural Inf. Process. Syst. Workshop*, 2014, pp. 1–9.
[147] J. Martens, *Second-Order Optimization For Neural Networks*, Univ. Toronto, Toronto, ON, Canada, 2016.
[148] N. N. Schraudolph and T. Graepel, "Conjugate directions for stochastic gradient descent," in *Proc. Int. Conf. Artif. Neural Netw.*, 2002, pp. 1351–1356.
[149] N. N. Schraudolph, J. Yu, and S. Günter, "A stochastic quasi-Newton method for online convex optimization," in *Proc. Art. Intell. Stat.*, 2007, pp. 436–443.
[150] 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.
[151] A. Bordes, L. Bottou, and P. Gallinari, "SGD-QN: Careful quasi-Newton stochastic gradient descent," *J. Mach. Learn. Res.*, vol. 10, pp. 1737–1754, Jan. 2009.
[152] X.-B. Jin, X.-Y. Zhang, K. Huang, and G.-G. Geng, "Stochastic conjugate gradient algorithm with variance reduction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 5, pp. 1360–1369, May 2019.

Shiliang Sun received the Ph.D. degree in pattern recognition and intelligent systems from the Department of Automation, State Key Laboratory of Intelligent Technology and Systems, Tsinghua University, Beijing, China, in 2007.

He is a Professor with the School of Computer Science and Technology and the Head of the Pattern Recognition and Machine Learning Research Group, East China Normal University, Shanghai, China. His research results have expounded over 100 publications at peer-reviewed journals and conferences, such as the *Journal of Machine Learning Research*, the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, the IEEE TRANSACTIONS ON CYBERNETICS, the IEEE TRANSACTIONS ON MULTIMEDIA, the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, ICML, IJCAI, and ECML. His current research interests include kernel methods, multiview learning, learning theory, approximate inference, sequential modeling, and their applications.

Prof. Sun is on the editorial board of multiple international journals, including *Neurocomputing* and the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS.

Zehui Cao is currently pursuing the master's degree with the Pattern Recognition and Machine Learning Research Group, School of Computer Science and Technology, East China Normal University, Shanghai, China.

Her current research interests include machine learning and pattern recognition.

Han Zhu is currently pursuing the master's degree with the Pattern Recognition and Machine Learning Research Group, School of Computer Science and Technology, East China Normal University, Shanghai, China.

Her current research interests include machine learning and pattern recognition.

Jing Zhao received the Ph.D. degree in pattern recognition and machine learning from the Department of Computer Science and Technology, East China Normal University, Shanghai, China, in 2016.

She is an Assistant Professor with the Pattern Recognition and Machine Learning Research Group, School of Computer Science and Technology, East China Normal University. Her research results have expounded in over ten publications at peer-reviewed journals and conferences, such as the *Journal of Machine Learning Research*, *Pattern Recognition*, the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, and IJCAI. Her current research interests include Bayesian methods, kernel methods, sequential modeling, and their applications.