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 paper, we first describe the optimization problems in machine learning. Then, we introduce the principles and progresses of commonly used optimization methods. Next, we summarize the applications and developments of optimization methods in some popular machine learning fields. Finally, we explore and give some challenges and open problems for the optimization in machine learning.**

***Index Terms*—Machine learning, optimization method, deep neural network, reinforcement learning, approximate Bayesian inference.**

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 system, 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 were put forward, which have improved the performance and efficiency of machine learning methods.

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

As the representative of first-order optimization methods, the stochastic gradient descent 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 paper, 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 to choose the first-order optimization methods more conveniently and make parameter adjustment more reasonable in the learning process.

Compared with first-order optimization methods, high-order methods [3], [4], [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 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], [9], [10].

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 derivative-free optimization methods. One is adopting a heuristic search based on empirical rules, and the other is fitting the objective function with samples. Derivative-free optimization methods can also work in conjunction with gradient-based methods.

Most machine learning problems, once formulated, can be solved as optimization problems. Optimization in the fields of deep neural network, reinforcement learning, meta learning, variational inference and Markov chain Monte Carlo 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.

Deep neural networks (DNNs) have shown great success in pattern recognition and machine learning. There are two

This work was supported by NSFC Project 61370175 and Shanghai Sailing Program 17YF1404600.

Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao are with School of Computer Science and Technology, East China Normal University, 3663 North Zhongshan Road, Shanghai 200062, P. R. China. E-mail: slsun@cs.ecnu.edu.cn, shiliangsun@gmail.com (Shiliang Sun); jzhao@cs.ecnu.edu.cn, jzhao2011@gmail.com (Jing Zhao)

very popular NNs, i.e., 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], [20], [21], [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], [27], [28], [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. Stochastic gradient-based algorithms are widely used in deep neural networks [30], [31], [32], [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 non-converging. Thus, further optimization algorithms based on variance reduction were proposed to improve the convergence rate [36], [37]. Moreover, combining the stochastic gradient descent and the characteristics of its variants is a possible direction to improve the optimization. Especially, switching an adaptive algorithm to the stochastic gradient descent method can improve the accuracy and convergence speed of the algorithm [38].

Reinforcement learning (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 reinforcement learning 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 reinforcement learning problems [40], [41], [42], [43], [44]. 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 the few-shot learning problems [47]. In general, meta learning methods can be summarized into the following three types [48]: metric-based methods [49], [50], [51], [52], model-based methods [53], [54] and optimization-based methods [55], [56], [47]. 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 Bayesian machine learning. It can be considered as an optimization problem. For example, mean-field variational inference uses coordinate ascent to solve this optimization problem [57]. As the amount of data increases 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, [5] proposed the transformer network using Adam optimization [33], which is applied to machine translation tasks. [59] proposed super-resolution generative adversarial network for image super resolution, which is also optimized by Adam. [60] proposed Actor-Critic using trust region optimization to solve the deep reinforcement learning on Atari games as well as the MuJoCo environments.

The stochastic optimization method can also be applied to Markov chain Monte Carlo (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 Gaussian noise and friction terms. Additionally, 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 optimization performance with insufficient data in deep neural networks is a tricky problem. If there are not enough samples in the training of deep neural networks, it is prone to cause the problem of high variances and overfitting [62]. In addition, non-convex optimization has been one of the difficulties in deep neural networks, which makes the optimization tend to get 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 paper is to summarize and analyze classical and modern optimization methods from a machine learning perspective. The remainder of this paper 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 describes the developments and applications of optimization methods in some specific machine learning fields. Section V presents the challenges and open problems in the optimization methods. Finally, we conclude the whole paper.

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 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, machine learning algorithms can be divided into supervised learning, semi-supervised learning, unsupervised learning, and reinforcement learning. Particularly, supervised learning is further divided into the classification problem (e.g., sentence classification [17], [63], image classification [64], [65], [66], etc.) and regression problem; unsupervised learning is divided into clustering and dimension reduction [67], [68], [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,

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

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 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, e.g., in terms of L_2 norm,

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(y^i, f(x^i, \theta)) + \lambda \parallel \theta \parallel_2^2 . \tag{2}$$

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

B. Optimization Problems in Semi-supervised Learning

Semi-supervised 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 semi-supervised learning methods including self-training, generative models, semi-supervised support vector machines (S3VM) [77], graph-based methods, multi-learning method and others. We take S3VM as an example to introduce the optimization in semi-supervised learning.

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 labeled data which can be represented as $D^l = \{\{x^1, y^1\}, \{x^2, y^2\}, ..., \{x^l, y^l\}\}$, and D^u be unlabeled data which can be represented as $D^u = \{x^{l+1}, x^{l+2}, ..., x^N\}$ with $N = l + u$. In order to use the information of unlabeled data, additional constraint on the unlabeled data is added to the original objective of SVM with slack variables ζ^i. Specifically, define 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

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

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], [83], [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. Principal component analysis (PCA) [85], [86], [87] is a typical

algorithm of dimensionality reduction methods. The objective of PCA is formulated to minimize the reconstruction error as

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

where N represents the number of samples, x_i is a D-dimensional vector, $\overline{x}^i$ is the reconstruction of x^i. $z^i = \{z_1^i, ..., z_{D'}^i\}$ 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 probabilistic models is to find an optimal probability density function of $p(x)$, which maximizes the logarithmic likelihood function (MLE) of the training samples,

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

In the framework of 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

Reinforcement learning [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 reinforcement learning can be formulated as maximizing the cumulative return after executing a series of actions which are determined by the policy function,

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

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, 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. 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 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 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

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

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

where N is the number of training samples, D is the number of input features, x^i is an independent variable with $x^i = (x_1^i, ..., x_D^i)$ for $i = 1, ..., N$ 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 get the gradient corresponding to each θ_j:

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

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

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

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 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 computation 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 stochastic gradient descent 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, stochastic gradient descent (SGD) was proposed [1]. The idea of stochastic gradient descent 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 stochastic gradient descent algorithm is independent of sample numbers and can achieve sublinear convergence speed [37]. SGD reduces the update time for dealing 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 [94], [95], [96], [36]. 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:

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

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

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

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

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

Since SGD uses only one sample per iteration, the computation 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 computation complexity caused by large numbers of samples. It is possible to use only thousands of samples overall to get 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 towards 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 converging.

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 [97], [98]. 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" [99]. 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 were developed [100], [101].

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 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 [102]. 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 [103]. 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 $v = \eta \cdot (-\frac{\partial L(\theta)}{\partial(\theta)})$ each time. Using the momentum algorithm, the amount of the update v is not just the amount of gradient descent calculated by $\eta \cdot (-\frac{\partial L(\theta)}{\partial(\theta)})$. 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

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

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 for escaping from the local minimum in the training process so that the search process can converge more quickly [102], [104]. If the current gradient is opposite to 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].

Nesterov Accelerated Gradient Descent (NAG) makes further improvement over the traditional momentum method [104], [105]. In Nesterov momentum, the momentum $v^{old} \cdot mtm$ is added to θ, denoted as $\widetilde{\theta}$. The gradient of $\widetilde{\theta}$ is used when updating. The detailed update formulae for parameters θ are as follows:

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

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 $O(\frac{1}{k})$ (after k steps) to $O(\frac{1}{k^2})$, when not using stochastic optimization [105].

Another issue worth considering is how to determine the size of the learning rate. It is more likely to occur 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 [106]. 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 tth 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 the 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], [107]. Some adaptive methods were proposed to adjust the learning rate automatically. These methods are free of parameter adjustment, fast to converge, and often achieving not bad results. They are widely used in deep neural networks 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 formulae 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 gradient descent is that during the parameter update process, the learning rate is no longer fixed, but is computed using all the 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. 2) As the training time increases, the accumulated gradient will become larger and larger, making the learning rate tend to zero, resulting in 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 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 $i, i \in \{1, ..., N\}$. 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:

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

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, 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 only applicable to the case where the loss function is smooth and the objective function is convex [36], [108], 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 only applicable to smooth and convex functions and needs to store the gradient of each sample, it is inconvenient to be applied in non-convex 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:

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

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 $i_t \in \{1, ..., N\}$ randomly, and executes gradient updates to the current parameters:

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

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 update, θ_t and the interval update parameter $\tilde{\theta}$ will converge to the optimal θ^*, and then $\tilde{\mu} \to 0$, and

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

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 non-convex neural network [37], [109], [110]. There are also many variants of such linear convergence stochastic optimization algorithms, such as the SAGA algorithm [111].

6) Alternating Direction Method of Multipliers: 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,

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

The augmented Lagrange function for problem (27) is

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

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

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

Separating the (x,y) sub-problem in (29), the augmented Lagrange multiplier method can be relaxed to the following alternating direction method of multipliers (ADMM) [112], [113]. Its tth step iteration starts with the given (y_t,λ_t), and the details of iterative optimization are as follows:

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

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 [114]. 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 [115], [116].

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 [117].

7) Frank-Wolfe Method: In 1956, Frank and Wolfe proposed an algorithm for solving linear constraint problems [118]. 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 one-dimensional search along the direction in the feasible domain. This method is also called the approximate linearization method.

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

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

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 equation (31). Then we have

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

which is equivalent to

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

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

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

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 [118], [119]

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 [118]:

(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 for solving 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 [120], [121], [122].

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

B. High-Order Methods

The second-order methods can be used for addressing the problem where an objective function is highly non-linear and ill-conditioned. They work effectively by introducing curvature information.

This section begins with introducing the conjugate gradient method, which is a method that only needs first-order derivative information for well-defined quadratic programming, but overcomes the shortcoming of the steepest descent method, and avoids the disadvantages of Newton's method of storing and calculating the inverse Hessian matrix. But note that when applying it to general optimization problems, the second-order gradient is needed to get an approximation to quadratic programming. Then, the classical quasi-Newton method using second-order information is described. Although the convergence of the algorithm can be guaranteed, the computational process is costly and thus rarely used for solving large machine learning problems. In recent years, with the continuous improvement of high-order optimization methods, more and more high-order methods have been proposed to handle large-scale data by using stochastic techniques [124], [125], [126]. From this perspective, we discuss several high-order methods including the stochastic quasi-Newton method (integrating the second-order information and the stochastic method) and their variants. These algorithms allow us to use high-order methods to process large-scale data.

1) Conjugate Gradient Method: The conjugate gradient (CG) approach is a very interesting optimization method, which is one of the most effective methods for solving large-scale linear systems of equations. It can also be used for solving nonlinear optimization problems [93]. As we know, the first-order methods are simple but have a slow convergence speed, and the second-order methods need a lot of resources. Conjugate gradient optimization is an intermediate algorithm, which can only utilize the first-order information for some problems but ensures the convergence speed like high-order methods.

Early in the 1960s, a conjugate gradient method for solving a linear system was proposed, which is an alternative to Gaussian elimination [127]. Then in 1964, the conjugate gradient method was extended to handle nonlinear optimization for general functions [93]. For years, many different algorithms have been presented based on this method, some of which have been widely used in practice. The main features of these algorithms are that they have faster convergence speed than steepest descent. Next, we describe the conjugate gradient method.

Consider a linear system,

$$A\theta = b, \tag{35}$$

where A is an $n \times n$ symmetric, positive-definite matrix. The matrix A and vector b are known, and we need to solve the value of θ. The problem (35) can also be considered as an optimization problem that minimizes the quadratic positive definite function,

$$\min_{\theta} \quad F(\theta) = \frac{1}{2}\theta^\top A\theta - b\theta + c. \tag{36}$$

The above two equations have an identical unique solution. It enables us to regard the conjugate gradient as a method for solving optimization problems.

The gradient of $F(\theta)$ can be obtained by simple calculation, and it equals the residual of the linear system [93]: $r(\theta) = \nabla F(\theta) = A\theta - b$.

Definition 1: Conjugate: Given an $n \times n$ symmetric positive-definite matrix A, two non-zero vector d_i, d_j are conjugate with respect to A if

$$d_i^\top A d_j = 0. \tag{37}$$

A set of non-zero vector $\{d_1, d_2, d_3, ..., d_n\}$ is said to be conjugate with respect to A if any two unequal vectors are conjugate with respect to A [93].

Next, we introduce the detailed derivation of the conjugate gradient method. θ_0 is a starting point, $\{d_t\}_{t=1}^{n-1}$ is a set of conjugate directions. In general, one can generate the update sequence $\{\theta_1, \theta_2, ..., \theta_n\}$ by a iteration formula:

$$\theta_{t+1} = \theta_t + \eta_t d_t. \tag{38}$$

The step size η_t can be obtained by a linear search, which means choosing η_t to minimize the object function $f(\cdot)$ along $\theta_t + \eta_t d_t$. After some calculations (more details in [93], [128]), the update formula of η_t is

$$\eta_t = \frac{r_t^\top r_t}{d_t^\top A d_t}. \tag{39}$$

The search direction d_t is obtained by a linear combination of the negative residual and the previous search direction,

$$d_t = -r_t + \beta_t d_{t-1}, \tag{40}$$

where r_t can be updated by $r_t = r_{t-1} + \eta_{t-1} A d_{t-1}$. The scalar β_t is the update parameter, which can be determined by satisfying the requirement that d_t and d_{t-1} are conjugate with respect to A, i.e., $d_t^\top A d_{t-1} = 0$. Multiplying both sides of the equation (40) by $d_{t-1}^\top A$, one can obtain β_t by

$$\beta_t = \frac{d_{t-1}^\top A r_t}{d_{t-1}^\top A d_{t-1}}. \tag{41}$$

After several derivations of the above formula according to [93], the simplified version of β_t is

$$\beta_t = \frac{r_t^\top r_t}{r_{t-1}^\top r_{t-1}}. \tag{42}$$

The CG method, has a graceful property that generating a new vector d_t only using the previous vector d_{t-1}, which does not need to know all the previous vectors $d_0, d_1, d_2 \ldots d_{t-2}$. The linear conjugate gradient algorithm is shown in Algorithm 2.

TABLE I: Summary of First-Order Optimization Methods

Method	Properties	Advantages	Disadvantages
GD	Solve the optimal value along the direction of the gradient descent. The method converges at a linear rate.	The solution is global optimal when the objective function is convex.	In each parameter update, gradients of total samples need to be calculated, so the calculation cost is high.
SGD [1]	The update parameters are calculated using a randomly sampled mini-batch. The method converges at a sublinear rate.	The calculation time for each update does not depend on the total number of training samples, and a lot of calculation cost is saved.	It is difficult to choose an appropriate learning rate, and using the same learning rate for all parameters is not appropriate. The solution may be trapped at the saddle point in some cases.
NAG [105]	Accelerate the current gradient descent by accumulating the previous gradient as momentum and perform the gradient update process with momentum.	When the gradient direction changes, the momentum can slow the update speed and reduce the oscillation; when the gradient direction remains, the momentum can accelerate the parameter update. Momentum helps to jump out of locally optimal solution.	It is difficult to choose a suitable learning rate.
AdaGrad [30]	The learning rate is adaptively adjusted according to the sum of the squares of all historical gradients.	In the early stage of training, the cumulative gradient is smaller, the learning rate is larger, and learning speed is faster. The method is suitable for dealing with sparse gradient problems. The learning rate of each parameter adjusts adaptively.	As the training time increases, the accumulated gradient will become larger and larger, making the learning rate tend to zero, resulting in ineffective parameter updates. A manual learning rate is still needed. It is not suitable for dealing with non-convex problems.
AdaDelta/ RMSProp [31], [32]	Change the way of total gradient accumulation to exponential moving average.	Improve the ineffective learning problem in the late stage of AdaGrad. It is suitable for optimizing non-stationary and non-convex problems.	In the late training stage, the update process may be repeated around the local minimum.
Adam [33]	Combine the adaptive methods and the momentum method. Use the first-order moment estimation and the second-order moment estimation of the gradient to dynamically adjust the learning rate of each parameter. Add the bias correction.	The gradient descent process is relatively stable. It is suitable for most non-convex optimization problems with large data sets and high dimensional space.	The method may not converge in some cases.
SAG [36]	The old gradient of each sample and the summation of gradients over all samples are maintained in memory. For each update, one sample is randomly selected and the gradient sum is recalculated and used as the update direction.	The method is a linear convergence algorithm, which is much faster than SGD.	The method is only applicable to smooth and convex functions and needs to store the gradient of each sample. It is inconvenient to be applied in non-convex neural networks.
SVRG [37]	Instead of saving the gradient of each sample, the average gradient is saved at regular intervals. The gradient sum is updated at each iteration by calculating the gradients with respect to the old parameters and the current parameters for the randomly selected samples.	The method does not need to maintain all gradients in memory, which saves memory resources. It is a linear convergence algorithm.	To apply it to larger/deeper neural nets whose training cost is a critical issue, further investigation is still needed.
ADMM [123]	The method solves optimization problems with linear constraints by adding a penalty term to the objective and separating variables into sub-problems which can be solved iteratively.	The 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. The framework is practical in most large-scale optimization problems.	The original residuals and dual residuals are both related to the penalty parameter whose value is difficult to determine.
Frank-Wolfe [118]	The method approximates the objective function with a linear function, solves the linear programming to find the feasible descending direction, and makes a one-dimensional search along the direction in the feasible domain.	The method can solve optimization problems with linear constraints, whose convergence speed is fast in early iterations.	The method converges slowly in later phases. When the iterative point is close to the optimal solution, the search direction and the gradient of the objective function tend to be orthogonal. Such a direction is not the best downward direction.

Algorithm 2 Conjugate Gradient Method [128]

Input: A, b, θ_0
Output: The solution θ^*
$r_0 = A\theta_0 - b$
$d_0 = -r_0$, $t = 0$
while Unsatisfied convergence condition **do**
$\eta_t = \frac{r_t^\top r_t}{d_t^\top A d_t}$
$\theta_{t+1} = \theta_t + \eta_t d_t$
$r_{t+1} = r_t + \eta_t A d_t$
$\beta_{t+1} = \frac{r_{t+1}^\top r_{t+1}}{r_t^\top r_t}$
$d_{t+1} = -r_{t+1} + \beta_{t+1} d_t$
$t = t + 1$
end while

2) Quasi-Newton Methods: Gradient descent employs first-order information, but its convergence rate is slow. Thus, the natural idea is to use second-order information, e.g., Newton's method [129]. The basic idea of Newton's method is to use both the first-order derivative (gradient) and second-order derivative (Hessian matrix) to approximate the objective function with a quadratic function, and then solve the minimum optimization of the quadratic function. This process is repeated until the updated variable converges.

The one-dimensional Newton's iteration formula is shown as

$$\theta_{t+1} = \theta_t - \frac{f'(\theta_t)}{f''(\theta_t)}, \tag{43}$$

where f is the object function. More general, the high-dimensional Newton's iteration formula is

$$\theta_{t+1} = \theta_t - \nabla^2 f(\theta_t)^{-1} \nabla f(\theta_t)\ , \quad t \geq 0, \tag{44}$$

where $\nabla^2 f$ is a Hessian matrix of f. More precisely, if the learning rate (step size factor) is introduced, the iteration formula is shown as

$$\begin{aligned} d_t &= -\nabla^2 f(\theta_t)^{-1} \nabla f(\theta_t), \\ \theta_{t+1} &= \theta_t + \eta_t d_t, \end{aligned} \tag{45}$$

where d_t is the Newton's direction, η_t is the step size. This method can be called damping Newton's method [130]. Geometrically speaking, Newton's method is to fit the local surface of the current position with a quadratic surface, while the gradient descent method is to fit the current local surface with a plane [131].

Quasi-Newton Method Newton's method is an iterative algorithm that requires the computation of the inverse Hessian matrix of the objective function at each step, which makes the storage and computation very expensive. To overcome the expensive storage and computation, an approximate algorithm was considered which is called the quasi-Newton method. The essential idea of the quasi-Newton method is to use a positive definite matrix to approximate the inverse of the Hessian matrix, thus simplifying the complexity of the operation. The quasi-Newton method is one of the most effective methods for solving non-linear optimization problems. Moreover, the second-order gradient is not directly needed in the quasi-Newton method, so it is sometimes more efficient than Newton's method. In the following section, we will introduce several quasi-Newton methods, in which the Hessian matrix and its inverse matrix are approximated by different methods.

Quasi-Newton Condition We first introduce the quasi-Newton condition. Assuming that the objective function f can be approximated by a quadratic function, we can extend $f(\theta)$ to Taylor series at $\theta = \theta_{t+1}$, i.e.,

$$\begin{aligned} f(\theta) \approx\ & f(\theta_{t+1}) + \nabla f(\theta_{t+1})^\top (\theta - \theta_{t+1}) \\ & + \frac{1}{2}(\theta - \theta_{t+1})^\top \nabla^2 f(\theta_{t+1})(\theta - \theta_{t+1}). \end{aligned} \tag{46}$$

Then we can compute the gradient on both sides of the above equation, and obtain

$$\nabla f(\theta) \approx \nabla f(\theta_{t+1}) + \nabla^2 f(\theta_{t+1})(\theta - \theta_{t+1}). \tag{47}$$

Set $\theta = \theta_t$ in (47), we have

$$\nabla f(\theta_t) \approx \nabla f(\theta_{t+1}) + \nabla^2 f(\theta_{t+1})(\theta_t - \theta_{t+1}). \tag{48}$$

Use B to represent the approximate matrix of the Hessian matrix. Set $s_t = \theta_{t+1} - \theta_t$, and $u_t = \nabla f(\theta_{t+1}) - \nabla f(\theta_t)$. The matrix B_{t+1} is satisfied that

$$u_t = B_{t+1} s_t. \tag{49}$$

This equation is called the quasi-Newton condition, or secant equation.

The search direction of quasi-Newton method is

$$d_t = -B_t^{-1} g_t, \tag{50}$$

where g_t is the gradient of f, and the update of quasi-Newton is

$$\theta_{t+1} = \theta_t + \eta_t d_t. \tag{51}$$

The step size η_t is chosen to satisfy the Wolfe conditions, which is a set of inequalities for inexact line searches $\min_{\eta_t} f(\theta_t + \eta_t d_t)$ [132]. Unlike Newton's method, quasi-Newton method uses B_t to approximate the true Hessian matrix. In the following paragraphs, we will introduce some particular quasi-Newton methods, in which H_t is used to express the inverse of B_t, i.e., $H_t = B_t^{-1}$.

DFP In the 1950s, a physical scientist, William C. Davidon [133], proposed a new approach to solve nonlinear problems. Then Fletcher and Powel [134] explained and improved this method, which sparked a lot of research in the late 1960s and early 1970s [6]. DFP is the first quasi-Newton method named after the initials of their three names. The DFP correction formula is one of the most creative inventions in the field of non-linear optimization, shown as below:

$$B_{t+1}^{(DFP)} = (I - \frac{u_t s_t^\top}{u_t^\top s_t}) B_t (I - \frac{s_t u_t^\top}{u_t^\top s_t}) + \frac{u_t u_t^\top}{u_t^\top s_t}. \tag{52}$$

The update formula of H_{t+1} is

$$H_{t+1}^{DFP} = H_t - \frac{H_t u_t u_t^\top H_t}{u_t^\top H_t u_t} + \frac{s_t s_t^\top}{u_t^\top s_t}. \tag{53}$$

BFGS Broyden, Fletcher, Goldfarb and Shanno proposed the BFGS method [135], [136], [137], [3], in which B_{t+1} is updated according to

$$B_{t+1}^{(BFGS)} = B_t - \frac{B_t s_t s_t^\top B_t}{s_t^\top B_t s_t} + \frac{u_t u_t^\top}{u_t^\top s_t}. \tag{54}$$

The corresponding update of H_{t+1} is

$$H_{t+1}^{(BFGS)} = (I - \frac{s_t u_t^\top}{s_t^\top u_t}) H_t (I - \frac{u_t s_t^\top}{s_t^\top u_t}) + \frac{u_t s_t^\top}{s_t^\top u_t}. \tag{55}$$

The quasi-Newton algorithm still cannot solve large-scale data optimization problem because the method generates a sequence of matrices to approximate the Hessian matrix. Storing these matrices needs to consume computer resources, especially for high-dimensional problems. It is also impossible to retain these matrices in the high-speed storage of computers, restricting its use to even small and midsize problems [138].

L-BFGS Limited memory quasi-Newton methods, named L-BFGS [138], [139] is an improvement based on the quasi-Newton method, which is feasible in dealing with the high-dimensional situation. The method stores just a few n-dimensional vectors, instead of retaining and computing fully dense $n \times n$ approximations of the Hessian [140]. The basic idea of L-BFGS is to store the vector sequence in the calculation of approximation H_{t+1}, instead of storing complete matrix H_t. L-BFGS makes further consolidation for the update formula of H_{t+1},

$$\begin{aligned} H_{t+1} &= (I - \frac{s_t u_t^\top}{u_t^\top s_t}) H_t (I - \frac{u_t s_t^\top}{u_t^\top s_t}) + \frac{s_t s_t^\top}{u_t^\top s_t} \\ &= V_t^\top H_t V_t + \rho s_t s_t^\top, \end{aligned} \tag{56}$$

where

$$V_t = I - \rho u_t s_t^\top, \qquad \rho_t = \frac{1}{s_t^\top u_t}. \tag{57}$$

The above equation means that the inverse Hessian approximation H_{t+1} can be obtained using the sequence pair $\{s_l, u_l\}_{l=t-p+1}^{t}$. H_{t+1} can be computed if we know pairs $\{s_l, y_l\}_{l=t-p+1}^{t}$. In other words, instead of storing and calculating the complete matrix H_{t+1}, L-BFGS only computes the latest p pairs of $\{s_l, y_l\}$. According to the equation, a recursive procedure can be reached. When the latest p steps are retained, the calculation of H_{t+1} can be expressed as [139]

$$\begin{aligned} & H_{t+1} = (V_t^\top V_{t-1}^\top \cdots V_{t-p+1}^\top) H_t^0 (V_{t-p+1} V_{t-p+2} \cdots V_t) \\ + \;& \rho_{t-p+1} (V_t^\top V_{t-1}^\top \cdots V_{t-p+2}) s_{t-p+1} s_{t-p+1}^\top (V_{t-p+2} \cdots V_t) \\ + \;& \rho_{t-p+2} (V_t^\top V_{t-1}^\top \cdot V_{t-p+3}^\top) s_{t-p+2} s_{t-p+2}^\top (V_{t-p+3} \cdots V_t) \\ + \;& \cdots \\ + \;& \rho_t s_t s_t^\top. \end{aligned} \tag{58}$$

The update direction $d_t = H_t g_t$ can be calculated, where g_t is the gradient of the objective function f. The detailed algorithm is shown in Algorithms 3 and 4.

For more information about BFGS and L-BFGS algorithms, one can refer to [93], [138]. Recently, the batch L-BFGS on machine learning was proposed [141], which uses the overlapping mini-batches for consecutive samples for quasi-Newton update. It means that the calculation of u_t becomes $u_t = \nabla_{S_{t+1}} f(\theta_{t+1}) - \nabla_{S_t} f(\theta_t)$, where S_t is a small subset of samples, meanwhile S_{t+1} and S_t are not independent, perhaps containing a relatively large overlap. Some numerical results in [141] have shown that the modification in L-BFGS is effective in practice.

Algorithm 3 Two-Loop Recursion for $H_t g_t$ [93]

Input: ∇f_t, u_t, s_t
Output: $H_{t+1} g_{t+1}$
  $g_t = \nabla f_t$
  $H_t^0 = \frac{s_t^\top u_t}{\|u_t\|^2} \mathrm{I}$
  for $l = t-1$ to $t-p$ **do**
    $\eta_l = \rho_l s_l^\top g_{l+1}$
    $g_l = g_{l+1} - \eta_l u_l$
  end for
  $r_{t-p-1} = H_t^0 g_{t-p}$
  for $l = t-p$ to $t-1$ **do**
    $\beta_l = \rho_l u_l^\top \rho_{l-1}$
    $\rho_l = \rho_{l-1} + s_l(\eta_l - \beta_l)$
  end for
  $H_{t+1} g_{t+1} = \rho$

Algorithm 4 Limited-BFGS [139]

Input: $\theta_0 \in R^n$, $\epsilon > 0$
Output: the solution θ^*
  $t = 0$
  $g_0 = \nabla f_0$
  $u_0 = \mathbf{1}$
  $s_0 = \mathbf{1}$
  while $\| g_t \| < \epsilon$ **do**
    Choose H_t^0, for example $H_t^0 = \frac{s_t^\top u_t}{\|u_t\|^2} I$
    $g_t = \nabla f_t$
    $d_t = -H_t g_t$ from Algorithm L-BFGS two-loop recursion for $H_t g_t$
    Search a step size η_t through Wolfe rule
    $\theta_{t+1} = \theta_t + \eta_t d_t$
    if $k > p$ **then**
      Discard the vector pair $\{s_{t-p}, y_{t-p}\}$ from storage
    end if
    Compute and save
    $s_t = \theta_{t+1} - \theta_t, u_t = g_{t+1} - g_t$
    $t = t + 1$
  end while

3) Stochastic Quasi-Newton Method: In many large-scale machine learning models, it is necessary to use a stochastic approximation algorithm with each step of update based on a relatively small training subset [125]. Stochastic algorithms often obtain the best generalization performances in large-scale learning systems [142]. The quasi-Newton method only uses the first-order gradient information to approximate the Hessian matrix. It is a natural idea to combine the quasi-Newton method with the stochastic method, so that it can perform on large-scale problems. Online-BFGS and online-LBFGS are two variants of BFGS [124].

Consider the minimization of a convex stochastic function,

$$\min_{\theta\in\mathbb{R}} \quad F(\theta) = \mathbb{E}[f(\theta, \xi)], \tag{59}$$

where ξ is a random seed. We assume that ξ represents a sample (or a set of samples) consisting of an input-output pair (x, y). In machine learning x typically represents an input and y is the target output. f usually has the following form:

$$f(\theta; \xi) = f(\theta; x_i, y_i) = l(h(w; x_i); y_i), \tag{60}$$

where h is a prediction model parameterized by θ, and l is a loss function. We define $f_i(\theta) = f(\theta; x_i, y_i)$, and use the empirical loss to define the objective,

$$F(\theta) = \frac{1}{N}\sum_{i=1}^{N} f_i(\theta). \tag{61}$$

Typically, if a large amount of training data is used to train the machine learning models, a better choice is using mini-batch stochastic gradient,

$$\nabla F_{S_t}(\theta_t) = \frac{1}{c}\sum_{i\in S_t} \nabla f_i(\theta_t), \tag{62}$$

where subset $S_t \subset \{1, 2, 3 \cdots N\}$ is randomly selected. c is the cardinality of S_t and $c \ll N$. Let $S_t^H \subset \{1, 2, 3, \cdots, N\}$ be a randomly chosen subset of the training samples and the stochastic Hessian estimate can be

$$\nabla^2 F_{S_t}(\theta_t) = \frac{1}{c_h}\sum_{i\in S_t^H} \nabla^2 f_i(\theta_t), \tag{63}$$

where c_h is the cardinality of S_t^H. With given stochastic gradient, a direct approach to develop stochastic quasi-Newton method is to transform deterministic gradients into stochastic gradients throughout the iterations, such as online-BFGS and online-LBFGS [124], which are two stochastic adaptations of the BFGS algorithms. Specifically, following the BFGS described in the previous section, s_t, u_t are modified as

$$s_t := \theta_{t+1} - \theta_t \quad \text{and} \quad u_t := \nabla F_{S_t}(\theta_{t+1}) - \nabla F_{S_t}(\theta_t). \tag{64}$$

One disadvantage of this method is that each iteration requires two gradient estimates. Besides this, a more worrying fact is that updating the inverse Hessian approximations in each step may not be reasonable [143]. Then the stochastic quasi-Newton (SQN) method was proposed, which is to use sub-sampled Hessian-vector products to update H_t by the LBFGS according to [125]. Meanwhile, the authors proposed an effective approach that decouples the stochastic gradient and curvature estimate calculations to obtain a stable Hessian approximation. In particular, since

$$\nabla F(\theta_{t+1}) - \nabla F(\theta_t) \approx \nabla^2 F(\theta_t)(\theta_{t+1} - \theta_t), \tag{65}$$

u_t can be rewritten as

$$u_t := \nabla^2 F_{S_t^H}(\theta_t) s_t. \tag{66}$$

Based on these techniques, an SQN Framework was proposed, and the detailed procedure is shown in Algorithm 5.

Algorithm 5 SQN Framework [143]

Input: θ_0, V, m, η_t
Output: The solution θ^*
for t=1, 2, 3, 4,....., **do**
  $s_t' = H_t g_t$ using the two-loop recursion.
  $s_t = -\eta_t s_t'$
  $\theta_{t+1} = \theta_t + s_t'$
  if update pairs **then**
    Compute s_t and u_t
    Add a new displacement pair $\{s_t, u_t\}$ to V
    if $|V| > m$ **then**
      Remove the eldest pair from V
    end if
  end if
end for

In the above algorithm, $V = \{s_t, u_t\}$ is a collection of m displacement pairs, and g_t is the current stochastic gradient $\nabla F_{S_t}(\theta_t)$. Meanwhile, the matrix-vector product $H_t g_t$ can be computed by a two-loop recursion as described in the previous section. Recently, more and more work has achieved very good results in stochastic quasi-Newton. Specifically, a regularized stochastic BFGS method was proposed, which makes a corresponding analysis of the convergence of this optimization method [144]. Further, an online L-BFGS was presented in [145]. A linearly convergent method was proposed [126], which combines the L-BFGS method in [125] with the variance reduction technique. Besides these, a variance reduced block L-BFGS method was proposed, which works by employing the actions of a sub-sampled Hessian on a set of random vectors [146].

To sum up, we have discussed the techniques of using stochastic methods in second-order optimization. The stochastic quasi-Newton method is a combination of the stochastic method and the quasi-Newton method, which makes the quasi-Newton method extend to large datasets. We have introduced the related work of the stochastic quasi-Newton method in recent years, which reflects the potential of the stochastic quasi-Newton method in machine learning applications.

4) Hessian-Free Optimization Method: The main idea of Hessian-free (HF) method is similar to Newton's method, which employs second-order gradient information. The difference is that the HF method is not necessary to directly calculate the Hessian matrix H. It estimates the product Hv by some techniques, and thus is called "Hessian free".

Consider a local quadratic approximation $Q_\theta(d_t)$ of the object F around parameter θ,

$$F(\theta_t + d_t) \approx Q_\theta(d_t) = F(\theta_t) + \nabla F(\theta_t)^\top d_t + \frac{1}{2} d_t^\top B_t d_t, \tag{67}$$

where d_t is the search direction. The HF method applies the conjugate gradient method to compute an approximate solution d_t of the linear system,

$$B_t d_t = -\nabla F(\theta_t), \tag{68}$$

where $B_t = H(\theta_t)$ is the Hessian matrix, but in practice B_t is often defined as $B_t = H(\theta_t) + \lambda I,\ \lambda \geq 0$ [7]. The new update is then given by

$$\theta_{t+1} = \theta_t + \eta_t d_t, \tag{69}$$

where η_t is the step size that ensures sufficient decrease in the objective function, usually obtained by a linear search. According to [7], the basic framework of HF optimization is shown in Algorithm 6.

Algorithm 6 Hessian-Free Optimization Method [7]

Input: θ_0, $\nabla f(\theta_0)$, λ
Output: The solution θ^*
  $t = 0$
  repeat
    $g_t = \nabla f(\theta_t)$
    Compute λ by some methods
    $B_t(v) \equiv H(\theta_t)v + \lambda v$
    Compute the step size η_t
    $d_t = CG(B_t, -g_t)$
    $\theta_{t+1} = \theta_t + \eta_t d_t$
    $t = t + 1$
  until satisfy convergence condition

The advantage of using the conjugate gradient method is that it can calculate the Hessian-vector product without directly calculating the Hessian matrix. Because in the CG-algorithm, the Hessian matrix is paired with a vector, then we can compute the Hessian-vector product to avoid the calculation of the Hessian inverse matrix. There are many ways to calculate Hessian-vector products, one of which is calculated by a finite difference as [7]

$$Hv = \lim_{\varepsilon \to +0} \frac{\nabla f(\theta + \varepsilon v) - \nabla f(\theta)}{\varepsilon}. \tag{70}$$

Sub-sampled Hessian-Free Method HF is a well-known method, and has been studied for decades in the optimization literatures, but has shortcomings when applied to deep neural networks with large-scale data [7]. Therefore, a sub-sampled technique is employed in HF, resulting in an efficient HF method [7], [147]. The cost in each iteration can be reduced by using only a small sample set S to calculate Hv. The objective function has the following form:

$$\min \ \ F(\theta) = \frac{1}{N} \sum_{i=1}^{N} f_i(\theta). \tag{71}$$

In the tth iteration, the stochastic gradient estimation can be written as

$$\nabla F_{S_t}(\theta_t) = \frac{1}{|S_t|} \sum_{i \in S_t} \nabla f_i(\theta_t), \tag{72}$$

and the stochastic Hessian estimate is expressed as

$$\nabla^2 F_{S_t^H}(\theta_t) = \frac{1}{|S_t^H|} \sum_{i \in S_t^H} \nabla^2 f_i(\theta_t). \tag{73}$$

As described above, we can obtain the approximate solution of direction d_t by employing the CG method to solve the linear system,

$$\nabla^2 F_{S_t^H}(\theta_t) d_t = -\nabla F_{S_t}(\theta_t), \tag{74}$$

in which the stochastic gradient and stochastic Hessian matrix are used. The basic framework of sub-sampled HF algorithm is given in [147].

A natural question is how to determine the size of S_t^H. On one hand, S_t^H can be chosen small enough so that the total cost of CG iteration is not much greater than a gradient evaluation. On the other hand, S_t^H should be large enough to get useful curvature information from Hessian-vector product. How to balance the size of S_t^H is a challenge being studied [147].

5) Natural Gradient: The natural gradient method can be potentially applied to any objective function which measures the performance of some statistical models [148]. It enjoys richer theoretical properties when applied to objective functions based on the KL divergence between the model's distribution and the target distribution, or certain approximation surrogates of these [149].

The traditional gradient descent algorithm is based on the Euclidean space. However, in many cases, the parameter space is not Euclidean, and it may have a Riemannian metric structure. In this case, the steepest direction of the objective function cannot be given by the ordinary gradient and should be given by the natural gradient [148].

We consider such a model distribution $p(y|x, \theta)$, and $\pi(x, y)$ is an empirical distribution. We need to fit the parameters $\theta \in R^N$. Assume that x is an observation vector, and y is its associated label. It has the objective function,

$$F(\theta) = \mathbb{E}_{(x,y)\sim\pi}[-\log p(y|x, \theta)], \tag{75}$$

and we need to solve the optimization problem,

$$\theta^* = \text{argmin}_\theta F(\theta). \tag{76}$$

According to [148], the natural gradient can be transformed from a traditional gradient multiplied by a Fisher information matrix, i.e.,

$$\nabla_N F = G^{-1} \nabla F, \tag{77}$$

where F is the object function, $\bigtriangledown F$ is the traditional gradient, $\bigtriangledown_N F$ is the natural gradient, and G is the Fisher information matrix, with the following form:

$$G = \mathbb{E}_{x\sim\pi}\left[\mathbb{E}_{y\sim p(y|x,\theta)}\left[(\frac{\partial p(y|x;\theta)}{\partial \theta})(\frac{\partial p(y|x;\theta)}{\partial \theta})^\top\right]\right]. \tag{78}$$

The update formula with the natural gradient is

$$\theta_t = \theta_t - \eta_t \nabla_N F. \tag{79}$$

We cannot ignore that the application of the natural gradient is very limited because of too much computation. It is expensive to estimate the Fisher information matrix and calculate its inverse matrix. To overcome this limitation, the truncated Newton's method was developed [7], in which the inverse is calculated by an iterative procedure, thus avoiding the direct calculation of the inverse of the Fisher information matrix. In addition, the factorized natural gradient (FNG) [150]

and Kronecker-factored approximate curvature (K-FAC) [151] methods were proposed to use the derivatives of probabilistic models to calculate the approximate natural gradient update.

6) Trust Region Method: The update process of most methods introduced above can be described as $\theta_t + \eta_t d_t$. The displacement of the point in the direction of d_t can be written as s_t. The typical trust region method (TRM) can be used for unconstrained nonlinear optimization problems [140], [152], [153], in which the displacement s_t is directly determined without the search direction d_t.

For the problem $\min f_\theta(x)$, the TRM [140] uses the second-order Taylor expansion to approximate the objective function $f_\theta(x)$, denoted as $q_t(s)$. Each search is done within the range of trust region with radius $\triangle_t$. This problem can be described as

$$\begin{cases} \min & q_t(s) = f_\theta(x_t) + g_t^\top s + \frac{1}{2} s^\top B_t s, \\ s.t. & ||s_t|| \leq \triangle_t, \end{cases} \tag{80}$$

where g_t is the approximate gradient of the objective function $f(x)$ at the current iteration point x_t, $g_t \approx \nabla f(x_t)$, B_t is a symmetric matrix, which is the approximation of Hessian matrix $\nabla^2 f_\theta(x_t)$, and $\triangle_t > 0$ is the radius of the trust region. If the L_2 norm is used in the constraint function, it becomes the Levenberg-Marquardt algorithm [154].

If s_t is the solution of the trust region subproblem (80), the displacement s_t of each update is limited by the trust region radius $\triangle_t$. The core part of the TRM is the update of $\triangle_t$. In each update process, the similarity of the quadratic model $q(s_t)$ and the objective function $f_\theta(x)$ is measured, and $\triangle_t$ is updated dynamically. The actual amount of descent in the tth iteration is [140]

$$\triangle f_t = f_t - f(x_t + s_t). \tag{81}$$

The predicted drop in the tth iteration is

$$\triangle q_t = f_t - q(s_t). \tag{82}$$

The ratio r_t is defined to measure the approximate degree of both,

$$r_t = \frac{\triangle f_t}{\triangle q_t}. \tag{83}$$

It indicates that the model is more realistic than expected when r_t is close to 1, and then we should consider expanding $\triangle_t$. At the same time, it indicates that the model predicts a large drop and the actual drop is small when r_t is close to 0, and then we should reduce $\triangle_t$. Moreover, if r_t is between 0 and 1, we can leave $\triangle_t$ unchanged. The thresholds 0 and 1 are generally set as the left and right boundaries of r_t [140].

7) Summary: We summarize the mentioned high-order optimization methods in terms of properties, advantages and disadvantages in Table III.

C. Derivative-Free Optimization

For some optimization problems in practical applications, the derivative of the objective function may not exist or is not easy to calculate. The solution of finding the optimal point, in this case, is called derivative-free optimization, which is a discipline of mathematical optimization [155], [156], [157]. It can find the optimal solution without the gradient information.

There are mainly two types of ideas for derivative-free optimization. One is to use heuristic algorithms. It is characterized by empirical rules and chooses methods that have already worked well, rather than derives solutions systematically. There are many types of heuristic optimization methods, including classical simulated annealing arithmetic, genetic algorithms, ant colony algorithms, and particle swarm optimization [158], [159], [160]. These heuristic methods usually yield approximate global optimal values, and theoretical support is weak. We do not focus on such techniques in this section. The other is to fit an appropriate function according to the samples of the objective function. This type of method usually attaches some constraints to the search space to derive the samples. Coordinate descent method is a typical derivative-free algorithm [161], and it can be extended and applied to optimization algorithms for machine learning problems easily. In this section, we mainly introduce the coordinate descent method.

The coordinate descent method is a derivative-free optimization algorithm for multi-variable functions. Its idea is that a one-dimensional search can be performed sequentially along each axis direction to obtain updated values for each dimension. This method is suitable for some problems in which the loss function is non-differentiable.

The vanilla approach is to select a set of bases $e_1, e_2, ..., e_D$ in the linear space as the search directions and minimizes the value of the objective function in each direction. For the target function $L(\Theta)$, when Θ^t is already obtained, the jth dimension of Θ^{t+1} is solved by [155]

$$\theta_j^{t+1} = \arg\min_{\theta_j \in R} L(\theta_1^{t+1}, ..., \theta_{j-1}^{t+1}, \theta_j, \theta_{j+1}^t, ..., \theta_D^t). \tag{84}$$

Thus, $L(\Theta^{t+1}) \leq L(\Theta^t) \leq ... \leq L(\Theta^0)$ is guaranteed. The convergence of this method is similar to the gradient descent method. The order of update can be an arbitrary arrangement from e_1 to e_D in each iteration. The descent direction can be generalized from the coordinate axis to the coordinate block [162].

The main difference between the coordinate descent and the gradient descent is that each update direction in the gradient descent method is determined by the gradient of the current position, which may not be parallel to any coordinate axis. In the coordinate descent method, the optimization direction is fixed from beginning to end. It does not need to calculate the gradient of the objective function. In each iteration, the update is only executed along the direction of one axis, and thus the calculation of the coordinate descent method is simple even for some complicated problems. For indivisible functions, the algorithm may not be able to find the optimal solution in a small number of iteration steps. An appropriate coordinate system can be used to accelerate the convergence. For example, the adaptive coordinate descent method takes principal component analysis to obtain a new coordinate system with as little correlation as possible between the coordinates [163]. The coordinate descent method still has limitations when performing on the non-smooth objective function, which may fall into a non-stationary point.

TABLE II: Summary of High-Order Optimization Methods

Method	Properties	Advantages	Disadvantages
Conjugate Gradient [127]	It is an optimization method between the first-order and second-order gradient methods. It constructs a set of conjugated directions using the gradient of known points, and searches along the conjugated direction to find the minimum points of the objective function.	CG method only calculates the first order gradient but has faster convergence than the steepest descent method.	Compared with the first-order gradient method, the calculation of the conjugate gradient is more complex.
Newton's Method [129]	Newton's method calculates the inverse matrix of the Hessian matrix to obtain faster convergence than the first-order gradient descent method.	Newton's method uses second-order gradient information which has faster convergence than the first-order gradient method. Newton's method has quadratic convergence under certain conditions.	It needs long computing time and large storage space to calculate and store the inverse matrix of the Hessian matrix at each iteration.
Quasi-Newton Method [93]	Quasi-Newton method uses an approximate matrix to approximate the the Hessian matrix or its inverse matrix. Popular quasi-Newton methods include DFP, BFGS and LBFGS.	Quasi-Newton method does not need to calculate the inverse matrix of the Hessian matrix, which reduces the computing time. In general cases, quasi-Newton method can achieve superlinear convergence.	Quasi-Newton method needs a large storage space, which is not suitable for handling the optimization of large-scale problems.
Sochastic Quasi-Newton Method [143].	Stochastic quasi-Newton method employs techniques of stochastic optimization. Representative methods are online-LBFGS [124] and SQN [125].	Stochastic quasi-Newton method can deal with large-scale machine learning problems.	Compared with the stochastic gradient method, the calculation of stochastic quasi-Newton method is more complex.
Hessian Free Method [7]	HF method performs a sub-optimization using the conjugate gradient, which avoids the expensive computation of inverse Hessian matrix.	HF method can employ the second-order gradient information but does not need to directly calculate Hessian matrices. Thus, it is suitable for high dimensional optimization.	The cost of computation for the matrix-vector product in HF method increases linearly with the increase of training data. It does not work well for large-scale problems.
Sub-sampled Hessian Free Method [147]	Sup-sampled Hessian free method uses stochastic gradient and sub-sampled Hessian-vector during the process of updating.	The sub-sampled HF method can deal with large-scale machine learning optimization problems.	Compared with the stochastic gradient method, the calculation is more complex and needs more computing time in each iteration.
Natural Gradient [148]	The basic idea of the natural gradient is to construct the gradient descent algorithm in the predictive function space rather than the parametric space.	The natural gradient uses the Riemann structure of the parametric space to adjust the update direction, which is more suitable for finding the extremum of the objective function.	In the natural gradient method, the calculation of the Fisher information matrix is complex.

D. 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 matrix [164]. 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,

$$M^{-1}A\theta = M^{-1}b, \tag{85}$$

will have the same solution as the system $A\theta = b$. But (85) may be easier to solve and the spectral properties of the coefficient matrix $M^{-1}A$ may be more favorable.

In most linear systems, e.g., $A\theta = b$, 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 [165].

The conjugate gradient algorithm mentioned previously is the most commonly used optimization method with preconditioning technology, which speeds up the convergence. The algorithm is shown in Algorithm 7.

E. Public Toolkits for Optimization

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 III.

IV. Developments and Applications for Selected Machine Learning Fields

Optimization is one of the cores of machine learning. Many optimization methods are further developed in the face of different machine learning problems and specific application environments. The machine learning fields selected in this

TABLE III: Available Toolkits for Optimization

Toolkit	Language	Description
CVX [166]	Matlab	CVX is a matlab-based modeling system for convex optimization but cannot handle large-scale problems. http://cvxr.com/cvx/download/
CVXPY [167]	Python	CVXPY is a python package developed by Stanford University Convex Optimization Group for solving convex optimization problems. http://www.cvxpy.org/
CVXOPT [168]	Python	CVXOPT can be used for handling convex optimization. It is developed by Martin Andersen, Joachim Dahl, and Lieven Vandenberghe. http://cvxopt.org/
APM [169]	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 [123]	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 multi-variate 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 [170]	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

Algorithm 7 Preconditioned Conjugate Gradient Method [93]

Input: A, θ_0, M, b
Output: The solution θ^*

$f_0 = f(\theta_0)$
$g_0 = \nabla f(\theta_0) = A\theta_0 - b$
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

section mainly include deep neural networks, reinforcement learning, variational inference and Markov chain Monte Carlo.

A. Optimization in Deep Neural Networks

The deep neural network (DNN) is a hot topic in the machine learning community in recent years. There are many optimization methods for DNNs. In this section, we introduce them from two aspects, i.e., first-order optimization methods and high-order optimization methods.

1) First-Order Gradient Method in Deep Neural Networks: The stochastic gradient optimization method and its adaptive variants have been widely used in DNNs and have achieved good performance. SGD introduces the learning rate decay factor and AdaGrad accumulates all previous gradients so that their learning rates are continuously decreased and converge to zero. However, the learning rates of these two methods make the update slow in the later stage of optimization. AdaDelta, RMSProp, Adam and other methods use the exponential averaging to provide effective updates and simplify the calculation. These methods use exponential moving average to alleviate the problems caused by the rapid decay of the learning rate but limit the current

learning rate to only relying on a few gradients [34]. Reddi et al. used a simple convex optimization example to demonstrate that the RMSProp and Adam algorithms could not converge [34]. Almost all the algorithms that rely on a fixed-size window of the past gradients will suffer from this problem, including AdaDelta and Nesterov-accelerated adaptive moment estimation (Nadam) [171].

It is better to rely on the long-term memory of past gradients rather than the exponential moving average of gradients to ensure convergence. A new version of Adam [34], called AmsGrad, uses a simple correction method to ensure the convergence of the model while preserving the original computational performance and advantages. Compared with the Adam method, the AmsGrad makes the following changes to the first-order moment estimation and the second-order moment estimation:

$$\begin{cases} m_t = \beta_{1t} m_{t-1} + (1-\beta_{1t}) g_t, \\ V_t = \sqrt{\beta_2 V_{t-1} + (1-\beta_2) g_t^2}, \\ \hat{V}_t = \max(\hat{V}_{t-1}, V_t), \end{cases} \tag{86}$$

where β_{1t} is a non-constant which decreases with time, and β_2 is a constant learning rate. The correction is operated in the second-order moment V_t, making $\hat{V}_t$ monotonous. $\hat{V}_t$ is substantially used in the iteration of the target function. The AmsGrad method takes the long-term memory of past gradients based on the Adam method, guarantees the convergence in the later stage, and works well in applications.

Further, adjusting parameters β_1, β_2 at the same time helps to converge to a certain extent. For example, β_1 can decay modestly as $\beta_{1t} = \frac{\beta_1}{t}$, $\beta_{1t} \leq \beta_1$, for all $t \in [T]$. β_2 can be set as $\beta_{2t} = 1 - \frac{1}{t}$, for all $t \in [T]$, as in AdamNC algorithm [34].

Another idea of combining SGD and Adam was proposed for solving the non-convergence problem of adaptive gradient algorithm [38]. Adaptive algorithms, such as Adam, converge fast and are suitable for processing sparse data. SGD with momentum can converge to more accurate results. The combination of SGD and Adam develops the advantages of both methods. Specifically, it first trains with Adam to quickly drop and then switches to SGD for precise optimization based on the previous parameters at an appropriate switch point. The strategy is named as switching from Adam to SGD (SWATS) [38]. There are two core problems in SWATS. One is when to switch from Adam to SGD, the other is how to adjust the learning rate after switching the optimization algorithm. The SWATS approach is described in detail below.

The movement d^{Adam} of the parameter at iteration t of the Adam is

$$d_t^{Adam} = \frac{\eta^{Adam}}{V_t} m_t, \tag{87}$$

where η^{Adam} is the learning rate of Adam [38]. The movement d^{SGD} of the parameter at iteration t of the SGD is

$$d_t^{SGD} = \eta^{SGD} g_t, \tag{88}$$

where η^{SGD} is the learning rate of SGD and g_t is the gradient of the current position [38].

The movement of SGD can be decomposed into the learning rates along Adam's direction and its orthogonal direction. If SGD is going to finish the trajectory but Adam has not finished due to the momentum after selecting the optimization direction, walking along Adam's direction is a good choice for SWATS. At the same time, SWATS also adjusts its optimized trajectory by moving in the orthogonal direction. Let

$$Proj_{Adam}\ d_t^{SGD} = d_t^{Adam}, \tag{89}$$

and derive solution

$$\eta_t^{SGD} = \frac{(d_t^{Adam})^{\mathrm{T}} d_t^{Adam}}{(d_t^{Adam})^{\mathrm{T}} g_t}, \tag{90}$$

where $Proj_{Adam}$ means the projection in the direction of Adam. To reduce noise, a moving average can be used to correct the estimate of the learning rate,

$$\lambda_t^{SGD} = \beta_2 \lambda_{t-1}^{SGD} + (1-\beta_2)\eta_t^{SGD}, \tag{91}$$

$$\tilde{\lambda}_t^{SGD} = \frac{\lambda_t^{SGD}}{1-\beta_2}, \tag{92}$$

where λ_t^{SGD} is the first moment of learning rate η^{SGD}, and $\tilde{\lambda}_t^{SGD}$ is the learning rate of SGD after converting [38]. For switch point, a simple guideline $|\tilde{\lambda}_t^{SGD} - \lambda_t^{SGD}| < \epsilon$ is often used [38]. Although there is no rigorous mathematical proof for selecting this conversion criterion, it performs well across a variety of applications. For the mathematical proof of switch point, further research can be conducted. Although the SWATS is based on Adam, this switching method is also applicable to other adaptive methods, such as AdaGrad and RMSProp. The procedure is insensitive to hyper-parameters and can obtain an optimal solution comparable to SGD, but with a faster training speed in the case of deep networks.

Recently some researchers are trying to explain and improve the adaptive methods [172], [173]. Their strategies can also be combined with the above switching techniques to enhance the performance of the algorithm.

General fully connected neural networks cannot process sequential data such as text and audio. Recurrent neural network (RNN) is a kind of neural networks that is more suitable for processing sequential data. It was generally considered that the use of first-order methods to optimize RNN was not effective, because the SGD and its variant methods were difficult to learn long-term dependencies in sequence problems [99], [104], [174].

In recent years, a well-designed method of random parameter initialization scheme using only SGD with momentum without curvature information has achieved good results in training RNNs [99]. In [104], [175], some techniques for improving the optimization in training RNNs are summarized such as the momentum methods and NAG. The first-order optimization methods have got development for training RNNs, but they still face the problem of slow convergence in deep RNNs. The high-order optimization methods employing curvature information can accelerate the convergence near the optimal value and is considered to be more effective in optimizing DNNs.

2) High-Order Gradient Method in Deep Neural Networks: We have described the first-order optimization method applied in DNNs. As most DNNs use large-scale data, different versions of stochastic gradient methods were developed and have got excellent performance and properties. For making full use of gradient information, the second-order method is gradually applied to DNNs. In this section, we mainly introduce the Hessian-free method in DNN.

Hessian-free (HF) method has been studied for a long time in the field of optimization, but it is not directly suitable for dealing with neural networks [7]. As the objective function in DNN is not convex, the exact Hessian matrix may not be positive definite. Therefore, some modifications need to be made so that the HF method can be applied to neural networks [176].

The Generalized Gauss-Newton Matrix One solution is to use the generalized Gauss-Newton (GGN) matrix, which can be seen as an approximation of a Hessian matrix [177]. The GGN matrix is a provably positive semidefinite matrix, which avoids the trouble of negative curvature. There are at least two ways to derive the GGN matrix [176]. Both of them require that $f(\theta)$ can be expressed as a composition of two functions written as $f(\theta) = Q(F(\theta))$ where $f(\theta)$ is the object function and Q is convex. The GGN matrix G takes the following form,

$$G = J^\top Q'' J, \tag{93}$$

where J is the Jacobian of F.

Damping Methods Another modification to the HF method is to use different damping methods. For example, Tikhonov damping, one of the most famous damping methods, is implemented by introducing a quadratic penalty term into the quadratic model. A quadratic penalty term $\frac{\lambda}{2}d^\top d$ is added to the quadratic model,

$$Q(\theta) := Q(\theta) + \frac{\lambda}{2}d^\top d = f(\theta_t) + \nabla f(\theta_t)^\top d + \frac{1}{2}d^\top B d, \tag{94}$$

where $B = H + \lambda I$, and $\lambda > 0$ determines the "strength" of the damping which is a scalar parameter. Thus, Bv is formulated as $Bv = (H + \lambda I)v = Hv + \lambda v$. However, the basic Tikhonov damping method is not good in training RNNs [178]. Due to the complex structure of RNNs, the local quadratic approximation in certain directions in the parameter space, even at very small distances, maybe highly imprecise. The Tikhonov damping method can only compensate for this by increasing punishment in all directions because the method lacks a selective mechanism [176]. Therefore, the structural damping was proposed, which makes the performance substantially better and more robust.

The HF method with structural damping can effectively train RNNs [176]. Now we briefly introduce the HF method with structural damping. Let $e(x, \theta)$ mean the vector-value function of θ which can be interpreted as intermediate quantities during the calculation of $f(x, \theta)$, where $f(x, \theta)$ is the object function. For instance, $e(x, \theta)$ might contain the activation function of some layers of hidden units in neural networks (like RNNs). A structural damping can be defined as

$$R(\theta) = \frac{1}{|S|}\sum_{(x,y)\in S} D(e(x, \theta), e(x, \theta_t)), \tag{95}$$

where D is a distance function or a loss function. It can prevent a large change in $e(x, \theta)$ by penalizing the distance between $e(x, \theta)$ and $e(x, \theta_t)$. Then, the damped local objective can be written as

$$Q_\theta(d)' = Q_\theta(d) + \mu R(d + \theta_t) + \frac{\lambda}{2}d^\top d, \tag{96}$$

where μ and λ are two parameters to be dynamically adjusted. d is the direction at the tth iteration. More details of the structural damping can refer to [176].

Besides, there are many second-order optimization methods employed in RNNs. For example, quasi-Newton based optimization and L-BFGS were proposed to train RNNs [179], [180].

In order to make the damping method based on punishment work better, the damping parameters can be adjusted continuously. A Levenberg-Marquardt style heuristic method was used to adjust λ directly [7]. The Levenberg-Marquardt heuristic is described as follows:

1) If $\gamma < \frac{1}{4}\lambda$ then $\lambda \leftarrow \frac{3}{2}\lambda$,
2) If $\gamma > \frac{3}{4}\lambda$ then $\lambda \leftarrow \frac{2}{3}\lambda$,

where γ is a "reduction rate" with the following form,

$$\gamma = \frac{f(\theta_{t-1} + d_t) - f(\theta_{t-1})}{M_{t-1}(d_t)}. \tag{97}$$

Sub-sampling As sub-sampling Hessian can be used to handle large-scale data, several variations of the sub-sampling methods were proposed [8], [9], [10], which used either stochastic gradients or exact gradients. These approaches use $B_t = \nabla^2_{S_t} f(\theta_t)$ as a Hessian approximation, where S_t is a subset of samples. We need to compute the Hessian-vector product in some optimization methods. If we adopt the sub-sampling method, it also means that we can save a lot of computation in each iteration, such as the method proposed in [7].

Preconditioning Preconditioning can be used to simplify the optimization problems. For example, preconditioning can accelerate the CG method. It is found that diagonal matrices are particularly effective and one can use the following preconditioner [7]:

$$M = \left[\mathrm{diag}(\sum_{i=1}^{N} \nabla f_i(\theta) \odot \nabla f_i(\theta)) + \lambda I\right]^\alpha, \tag{98}$$

where $\odot$ denotes the element-wise product and the exponent α is chosen to be less than 1.

B. Optimization in Reinforcement Learning

Reinforcement learning (RL) is an important research field of machine learning and is also one of the most popular topics. Agents using deep reinforcement learning have achieved great success in learning complex behavior skills and solved challenging control tasks in high-dimensional primitive

perceptual state space [181], [182], [183]. It interacts with the environment through the trial-and-error mechanism and learns optimal strategies by maximizing cumulative rewards [39].

We describe several concepts of reinforcement learning as follows:

1) Agent: making different actions according to the state of the external environment, and adjusting the strategy according to the reward of the external environment.
2) Environment: all things outside the agent that will be affected by the action of the agent. It can change the state and provide the reward to the agent.
3) State s: a description of the environment.
4) Action a: a description of the behavior of the agent.
5) Reward $r_t(s_{t-1}, a_{t-1}, s_t)$: the timely return value at time t.
6) Policy $\pi(a|s)$: a function that the agent decides the action a according to the current state s.
7) State transition probability $p(s'|s,a)$: the probability distribution that the environment will transfer to state s' at the next moment, after the agent selecting an action a based on the current state s.
8) $p(s', r|s, a)$: the probability that the agent transforms to state s' and obtains the reward r, where the agent is in state s and selecting the action a.

Many reinforcement learning problems can be described by Markov decision process (MDP) $< S, A, P, \gamma, r >$ [39], in which S is state space, A is action space, P is state transition probability function, r is reward function and γ is the discount factor $0 < \gamma < 1$. At each time, the agent accepts a state and selects the action from an action set according to the policy. The agent receives feedback from the environment and then moves to the next state. The goal of reinforcement learning is to find a strategy that allows us to get the maximum γ-discounted cumulative reward. The discounted return is calculated by

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{99}$$

People do not necessarily know the MDP behind the problem. From this point, reinforcement learning is divided into two categories. One is model-based reinforcement learning which knows the MDP of the whole model (including the transition probability P and reward function r), and the other is the model-free method in which the MDP is unknown. Systematic exploration is required in the latter methods.

The most commonly used value function is the state value function,

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s], \tag{100}$$

which is the expected return of executing policy π from state s. The state-action value function is also essential which is the expected return for selecting action a under state s and policy π,

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]. \tag{101}$$

The value function of the current state s can be calculated by the value function of the next state s'. The Bellman equations of $V_\pi(s)$ and $Q_\pi(s,a)$ describe the relation by

$$\begin{aligned} V_\pi(s) =& \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r(s, a, s') \\ &+ \gamma V_\pi(s')], \end{aligned} \tag{102}$$

$$\begin{aligned} Q_\pi(s,a) =& \sum_{s',r} p(s', r|s, a)[r(s, a, s') \\ &+ \gamma \sum_{a'} \pi(a'|s') Q_\pi(s', a')]. \end{aligned} \tag{103}$$

There are many reinforcement learning methods based on value function. They are called value-based methods, which play a significant role in RL. For example, Q-learning [184] and SARSA [185] are two popular methods which use temporal difference algorithms. The policy-based approach is to optimize the policy $\pi_\theta(a|s)$ directly and update the parameters θ by gradient descent [186].

The actor-critic algorithm is a reinforcement learning method combining policy gradient and temporal differential learning, which learns both a policy and a state value function. It estimates the parameters of two structures simultaneously.

1) The actor is a policy function, which is to learn a policy $\pi_\theta(a|s)$ to obtain the highest possible return.
2) The critic refers to the learned value function $V_\phi(s)$, which estimates the value function of the current policy, that is to evaluate the quality of the actor.

In the actor-critic method, the critic solves a problem of prediction, while the actor pays attention to the control [187]. There is more information of actor-critic method in [88], [187]

The summary of the value-based method, the policy-based method, and the actor-critic method are as follows:

1) The value-based method: It needs to calculate value function, and usually gets a definite policy.
2) The policy-based method: It optimizes the policy π without selecting an action according to value function.
3) The actor-critic method: It combines the above two methods, and learns both the policy π and the state value function.

Deep reinforcement learning (DRL) combines reinforcement learning and deep learning, which defines problems and optimizes goals in the framework of RL, and solves problems such as state representation and strategy representation using deep learning techniques.

DRL has achieved great success in many challenging control tasks and uses DNNs to represent the control policy. For neural network training, a simple stochastic gradient algorithm or other first-order algorithms are usually chosen, but these algorithms are not efficient in exploring the weight space, which makes DRL methods often take several days to train [60]. So, a distributed method was proposed to solve this problem, in which parallel actor-learners have a stabilizing effect during training [182]. It executes multiple agents to interact with the environment simultaneously, which reduces the training time. But this method ignores the sampling efficiency. A scalable and sample-efficient natural gradient

algorithm was proposed, which uses a Kronecker-factored approximation method to compute the natural policy gradient update, and employ the update to the actor and the critic (ACKTR) [60].

C. Optimization in Meta Learning

Meta learning [45], [46] is a popular research direction in the field of machine learning. It solves the problem of learning to learn. In the past cognition, the research of machine learning is to obtain a large amount of data in a specific task firstly and then use the data to train the model. In machine learning, adequate training data is the guarantee of achieving good performance. However, human beings can well process new tasks with only a few training samples, which are much more efficient than traditional machine learning methods. The key point could be that the human brain has learned "how to learn" and can make full use of past knowledge and experience to guide the learning of new tasks. Therefore, how to make machines have the ability to learn efficiently like human beings has become a frontier issue in machine learning.

The goal of meta learning is to design a model that can training well in the new tasks using as few samples as possible without overfitting. The process of adapting to the new tasks is essentially a learning process in the meta-testing, but only with limited samples from new tasks. The application of meta learning methods in supervised learning can solve the few-shot learning problems [47].

As few-shot learning problems receive more and more attention, meta learning is also developing rapidly. In general, meta learning methods can be summarized into the following three types [48]: metric-based methods [49], [50], [51], [52], model-based methods [53], [54] and optimization-based methods [55], [56], [47]. In this subsection, we focus on the optimization-based meta learning methods. In meta learning, there are usually some tasks with sufficient training samples and a new task with only a few training samples. The main idea can be described as follows: in the meta-train step, sample a task τ from the total task set $\mathcal{T}$, which contains $(D_{\tau}^{train}, D_{\tau}^{test})$. For task τ, train and update the optimizer parameter θ with the training samples D_{τ}^{train}, update the meta-optimizer parameter ϕ with the test samples D_{τ}^{test}. The process of sampling tasks and updating parameters are repeated multiple times. In the meta-test step, the trained meta-optimizer is used for learning a new task.

Since the purpose of meta learning is to achieve fast learning, a key point is to make the gradient descent more accurately in the optimization. In some meta learning methods, the optimization process itself can be regarded as a learning problem to learn the prediction gradient rather than a determined gradient descent algorithm [188]. Neural networks with original gradient as input and prediction gradient as output is often used as a meta-optimizer [55]. The neural work is trained using the training and test samples from other tasks and used in the new task. The parameter update in the process of training is as follows:

$$\theta_{t+1} = \theta_t + N(g(\theta_t), \phi), \tag{104}$$

where θ_t is the model parameter at the iteration t, and N is the meta-optimizer with parameter ϕ that learns how to predict the gradient. After training, the meta-optimizer N and its parameter ϕ are updated according to the loss value in the test samples. The experiments have confirmed that learning neural optimizers is advantageous compared to the most advanced adaptive stochastic gradient optimization methods used in deep learning [55]. Due to the similarity between the gradient update in backpropagation and the cell state update in the long short-term memory (LSTM), LSTM is often used as the meta-optimizer [55], [56].

A model-agnostic meta learning algorithm (MAML) is another method for meta learning which was proposed to learn the parameters of any model subjected to gradient descent methods. It is applicable to different learning problems, including classification, regression and reinforcement learning [47]. The basic idea of the model-agnostic algorithm is to begin multiple tasks at the same time, and then get the synthetic gradient direction of different tasks, so as to learn a common base model. The main process can be described as follows: in the meta-train step, multiple tasks batch τ_i, which contains $(D_i^{train}, D_i^{test})$, are extracted from the total task set $\mathcal{T}$. For all τ_i, train and update the parameter $\theta_i^{'}$ with the train samples D_i^{train}:

$$\theta_i^{'} = \theta - \alpha \frac{\partial J_{\tau_i}(\theta)}{\partial(\theta)}, \tag{105}$$

where α is the learning rate of training process and $J_{\tau_i}(\theta)$ is the loss function in task i with training samples D_i^{train}. After the training step, use the synthetic gradient direction of these parameters $\theta_i^{'}$ on the test samples D_i^{test} of the respective task to update parameter θ:

$$\theta = \theta - \beta \frac{\partial \sum_{\tau_i \sim p(\mathcal{T})} J_{\tau_i}(\theta_i^{'})}{\partial(\theta)}, \tag{106}$$

where β is the meta learning rate of the test process and $J_{\tau_i}(\theta)$ is the loss function in task i with test samples D_i^{test}. The meta-train step is repeated multiple times to optimize a good initial parameter θ. In the meta-test step, the trained parameter θ is used as the initial parameter such that the model has a maximal performance on the new task. MAML does not introduce additional parameters for meta learning, nor does it require a specific learner architecture. The development of the method is of great significance to the optimization-based meta learning methods. Recently, an expanded task-agnostic meta learning algorithm is proposed to enhance the generalization of meta-learner towards a variety of tasks, which achieves outstanding performance on few-shot classification and reinforcement learning tasks [189].

D. Optimization in Variational Inference

In the machine learning community, there are many attractive probabilistic models but with complex structures and intractable posteriors, and thus some approximate methods are used, such as variational inference and Markov chain Monte Carlo (MCMC) sampling. Variational inference, a common technique in machine learning, is widely used to

approximate the posterior density of the Bayesian model, which transforms intricate inference problems into high-dimensional optimization problems [190], [191]. Compared with MCMC, the variational inference is faster and more suitable for dealing with large-scale data. Variational inference has been applied to large-scale machine learning tasks, such as large-scale document analysis, computer vision and computational neuroscience [192].

Variational inference often defines a flexible family of distributions indexed by free parameters on latent variables [190], and then finds the variational parameters by solving an optimization problem.

Now let us review the principle of variational inference [58]. Variational inference approximates the true posterior by attempting to minimize the Kullback-Leibler (KL) divergence between a potential factorized distribution and the true posterior.

Let $Z = \{z_i\}$ represent the set of all latent variables and parameters in the model and $X = \{x_i\}$ be a set of all observed data. The joint likelihood of X and Z is $p(Z, X) = p(Z)p(X|Z)$. In Bayesian models, the posterior distribution $p(Z|X)$ should be computed to make further inference.

What we need to do is to approximate $p(Z|X)$ with the distribution $q(Z)$ that belongs to a constrained family of distributions. The goal is to make the two distributions as similar as possible. Variational inference chooses KL divergence to measure the difference between the two distributions, that is to minimize the KL divergence of $q(Z)$ and $p(Z|X)$. Here is the formula for the KL divergence between q and p:

$$\begin{aligned}
&\mathrm{KL}[q(Z)||p(Z|X)] = \mathbb{E}_q\left[\log \frac{q(Z)}{p(Z|X)}\right] \\
&= \mathbb{E}_q[\log q(Z)] - \mathbb{E}_q[\log p(Z|X)] \\
&= \mathbb{E}_q[\log q(Z)] - \mathbb{E}_q[\log p(Z, X)] + \log p(X) \\
&= -\mathrm{ELBO}(q) + const, \qquad (107)
\end{aligned}$$

where $\log p(X)$ is replaced by a constant because we are only interested in q. With the above formula, we can know KL divergence is difficult to optimize because it requires knowing the distribution that we are trying to approximate. An alternative method is to maximize the evidence lower bound (ELBO), a lower bound on the logarithm of the marginal probability of the observations. We can obtain ELBO's formula as

$$\mathrm{ELBO}(q) = \mathbb{E}\left[\log p(Z, X)\right] - \mathbb{E}\left[\log q(Z)\right]. \qquad (108)$$

Variational inference can be treated as an optimization problem with the goal of minimizing the evidence lower bound. A direct method is to solve this optimization problem using the coordinate ascent, which is called coordinate ascent variational inference (CAVI). CAVI iteratively optimizes each factor of the mean-field variational density, while holding the others fixed [192].

Specifically, variational distribution q has the structure of the mean-field, i.e., $q(Z) = \prod_{i=1}^{M} q_i(z_i)$. With this assumption, we can bring the distribution q into the ELBO, by some derivation according to [57], and obtain the following formula:

$$q_i^* \propto \exp\{\mathbb{E}_{-i}[\log p(z_i, Z_{-i}, X)]\}. \qquad (109)$$

Then the CAVI algorithm can be given below in Algorithm 8

Algorithm 8 Coordinate Ascent Variational Inference [192]

Input: $p(X, Z)$, X
Output: $q(Z) = \prod_{i=1}^{M} q_i(z_i)$
  Initialize Variational factors $q_i(z_i)$
  repeat
    for i=1,2,3....,M **do**
      $q_i^* \propto \exp\{\mathbb{E}_{-i}[\log p(z_i, Z_{-i}, X)]\}$
    end for
    Compute ELBO(q):

$$\mathrm{ELBO}(q) = \mathbb{E}[\log p(Z, X)] - \mathbb{E}\log q(Z)$$

  until ELBO converges

In traditional coordinate ascension algorithms, the efficiency of processing large data is very low, because each iteration needs to compute all the data, which is very time-consuming. Modern machine learning models often need to analyze and process large-scale data, which is difficult and costly. Stochastic optimization enables machine learning to be extended on massive data [193]. This reminds us of an attractive technique to handle large data sets: stochastic optimization [97], [192], [194]. By introducing stochastic optimization into variational inference, the stochastic variational inference (SVI) was proposed [58], in which the exponential family is taken as a typical example.

Gaussian process (GP) is an important machine learning method based on statistical learning and Bayesian theory. It is suitable for complex regression problems such as high dimensions, small samples, and nonlinearities. GP has the advantages of strong generalization ability, flexible non-parametric inference, and strong interpretability. However, the complexity and storage requirements of accurate solution for GP are high, which hinders the development of GP under large-scale data. The stochastic variational inference method introduced in this section can popularize variational inference on large-scale datasets, but it can only be applied to probabilistic models with factorized structures. For GPs whose observations are correlated with each other, the stochastic variational inference can be adapted by introducing the global inducing variables as variational variables [195], [196]. Specifically, the observations are assumed to be conditionally independent given the inducing variables and the variational distribution for the inducing variables is assumed to have an explicit form. Thus, the resulting GP model can be factorized in a necessary manner, enabling the stochastic variational inference. This method can also be easily extended to models with non-Gaussian likelihood or latent variable models based on GPs.

E. Optimization in Markov Chain Monte Carlo

Markov chain Monte Carlo (MCMC) is a class of sampling algorithms to simulate complex distributions that are difficult to sample directly. It is a practical tool for Bayesian posterior inference. The traditional and common MCMC algorithms

include Gibbs sampling, slice sampling, Hamiltonian Monte Carlo (HMC) [197], [198], Reimann manifold variants [199], and so on. These sampling methods are limited by the computational cost and are difficult to extend to large-scale data.This section takes HMC as an example to introduce the optimization in MCMC. The bottleneck of the HMC is that the gradient calculation is costly on large data sets.

We first introduce the derivation of HMC. Consider the random variable θ, which can be sampled from the posterior distribution,

$$p(\theta|D) \propto \exp(-U(\theta)), \tag{110}$$

where D is the set of observations, and U is the potential energy function with the following formula:

$$U(\theta) = -\log p(\theta|D) = -\sum_{x\in D} \log p(x|\theta) - \log p(\theta). \tag{111}$$

In HMC [197], an independent auxiliary momentum variable r is introduced from Hamiltonian dynamic. The Hamiltonian function and the joint distribution of θ and r are described by

$$H(\theta, r) = U(\theta) + \frac{1}{2} r^T M^{-1} r = U(\theta) + K(r), \tag{112}$$

$$p(\theta, r) \propto \exp(-U(\theta) - \frac{1}{2} r^T M^{-1} r), \tag{113}$$

where M denotes the mass matrix, and $K(r)$ is the kinetic energy function. The process of HMC sampling is derived by simulating the Hamiltonian dynamic system,

$$\begin{cases} d\theta = M^{-1} r dt, \\ dr = -\nabla U(\theta) dt. \end{cases} \tag{114}$$

Hamiltonian dynamic describes the continuous motion of a particle. Hamiltonian equations are numerically approximated by the discretized leapfrog integrator for practical simulating [197]. The update equations are as follows [197]:

$$\begin{cases} r_i(t + \frac{\epsilon}{2}) = r_i(t) - \frac{\epsilon}{2} dr(t), \\ \theta_i(t + \epsilon) = \theta_i(t) + \epsilon d\theta(t + \frac{\epsilon}{2}), \\ r_i(t + \epsilon) = r_i(t + \frac{\epsilon}{2}) - \frac{\epsilon}{2} dr(t + \epsilon). \end{cases} \tag{115}$$

In the case of large datasets, the gradient of $U(\theta)$ needs to be calculated on the entire data set in each leapfrog iteration. In order to improve the efficiency, the stochastic gradient method was used to calculate $\nabla U(\theta)$ with a mini-batch $\tilde{D}$ sampled uniformly from D, which reduces the cost of calculation [61]. However, the gradient calculated in a mini-batch instead of the full dataset will cause noise. According to the central limit theorem, this noisy gradient can be approximated as

$$\nabla \tilde{U}(\theta) \approx \nabla U(\theta) + \mathcal{N}(0, V(\theta)), \tag{116}$$

where gradient noise obeys normal distribution whose covariance is $V(\theta)$. If we replace $\nabla U(\theta)$ by $\nabla \tilde{U}(\theta)$ directly, the Hamiltonian dynamics will be changed as

$$\begin{cases} d\theta = M^{-1} r dt, \\ dr = -\nabla U(\theta) dt + \mathcal{N}(0, 2B(\theta) dt), \end{cases} \tag{117}$$

where $B(\theta) = \frac{1}{2}\epsilon V(\theta)$ is the diffusion matrix [61].

Since the discretization of the dynamical system introduces noise, the Metropolis-Hastings (MH) correction step should be done after the leapfrog step. These MH steps require expensive calculations overall data in each iteration. Beyond that, there is an incorrect stationary distribution [200] in the stochastic gradient variant of HMC. Thus, Hamiltonian dynamic was further modified, which minimizes the effect of the additional noise, achieves the invariant distribution and eliminates MH steps [61]. Specifically, a friction term is added to the dynamical process of momentum update:

$$\begin{cases} d\theta = M^{-1} r dt, \\ dr = -\nabla U(\theta) dt - B M^{-1} r dt + \mathcal{N}(0, 2B(\theta) dt). \end{cases} \tag{118}$$

The introduced friction term is helpful for decreasing total energy $H(\theta, r)$ and weakening the effects of noise in the momentum update phase. The dynamical system is also the type of second-order Langevin dynamics with friction in physics, which can explore efficiently and counteract the effect of the noisy gradients [61] and thus no MH correction is required. This second-order Langevin dynamic MCMC method, called SGHMC, is used to deal with sampling problems on large data sets [61], [201].

Moreover, HMC is highly sensitive to hyper-parameters, such as the path length (step number) L and the step size ϵ. If the hyper-parameters are not set properly, the efficiency of the HMC will drop dramatically. There are some methods to optimize these two hyper-parameters instead of manually setting them.

1) Path Length L: The value of path length L has a great influence on the performance of HMC. If L is too small, the distance between the resulting sample points will be very close; if L is too large, the resulting sample points will loop back, resulting in wasted computation. In general, manually setting L cannot maximize the sampling efficiency of the HMC.

Matthew et al. [202] proposed an extension of the HMC method called the No-U-Turn sampler (NUTS), which uses a recursive algorithm to generate a set of possible independent samples efficiently, and stops the simulation by discriminating the backtracking automatically. There is no need to set the step parameter L manually. In models with multiple discrete variables, the ability of NUTS to select the track length automatically allows it to generate more valid samples and perform more efficiently than the original HMC.

2) Adaptive Step Size ϵ: The performance of HMC is highly sensitive to the step size ϵ in leapfrog integrator. If ϵ is too small, the update will slow, and the calculation cost will be high; if ϵ is too large, the rejection rate will be high, resulting in useless updates.

To set ϵ reasonably and adaptively, a vanishing adaptation of the dual averaging algorithm can be used in HMC [203], [204]. Specifically, a statistic $H_t = \delta - \alpha_t$ is adopted in dual averaging method, where δ is the desired average acceptance probability, and α_t is the current Metropolis-Hasting acceptance probability for iteration t. The statistic H_t's expectation $h(\epsilon)$ is defined as

$$h(\epsilon) \equiv \mathbb{E}_t[H_t|\epsilon_t] \equiv \lim_{T\to\infty} \frac{1}{T} \mathbb{E}[H_t|\epsilon_t], \tag{119}$$

where ϵ_t is the step size for iteration t in the leapfrog integrator. To satisfy $h(\epsilon) \equiv \mathbb{E}_t[H_t|\epsilon_t] = 0$, we can derive the update formula of ϵ, i.e., $\epsilon_{t+1} = \epsilon_t - \eta_t H_t$. Tuning ϵ by vanishing adaptation algorithm guarantees that the average acceptance probability of Metropolis verges to a fixed value.

The hyper-parameters in the HMC include not only the step size ϵ and the length of iteration steps L, but also the mass M, etc. Optimizing these hyper-parameters can help improve sampling performance [199], [205], [206]. It is convenient and efficient to tune the hyper-parameters automatically without cumbersome adjustments based on data and variables in MCMC. These adaptive tuning methods can be applied to other MCMC algorithms to improve the performance of the samplers.

In addition to second-order SGHMC, stochastic gradient Langevin dynamics (SGLD) [207] is a first-order Langevin dynamic technique combined with stochastic optimization. Efficient variants of both SGLD and SGHMC are still active [201], [208].

V. Challenges and Open Problems

With the rise of practical demand and the increase of the complexity of 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, respectively. One is insufficient data in training, and the other is a non-convex objective in DNNs.

1) Insufficient Data in Training Deep Neural Networks: In general, deep learning is based on big data sets 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 puts and returns [209]. Each expected result at the output layer is calculated as

$$o = \mathbb{E}_M[f(x;\theta,M)] = \sum_{i=1}^{M} p(M_i) f(x;\theta,M_i), \qquad (120)$$

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 sub-network [210].

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 [211] 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 is 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) Non-convex Optimization in Deep Neural Network: Convex optimization has good properties and a comprehensive set of tools are open to solve the optimization problem. However, many machine learning problems are formulated as non-convex optimization problems. For example, almost all the optimization problems in DNNs are non-convex. Non-convex optimization is one of the difficulties in the optimization problem. Unlike convex optimization, there may be innumerable optimum solutions in its feasible domain in non-convex problems. The complexity of the algorithm for searching the global optimal value is NP-hard [109].

In recent years, non-convex optimization has gradually attracted the attention of researches. The methods for solving non-convex optimization problems can be roughly divided into two types. One is to transform the non-convex optimization into a convex optimization problem, and then use the convex optimization method. The other is to use some special optimization method for solving non-convex functions directly. There is some work on summarizing the optimization methods for solving non-convex functions from the perspective of machine learning [212].

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 [213], [214], 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 [215]. For unsupervised and semi-supervised learning, the convex relaxation method was used for solving semidefinite programming [216].

2) Non-convex optimization methods: These methods include projection gradient descent [217], [218], alternating minimization [219], [220], [221], expectation maximization algorithm [222], [223] and stochastic optimization and its variants [37].

B. Difficulties in Sequential Models with Large-Scale Data

When dealing with large-scale time series, the usual solutions are using stochastic optimization, processing data in mini-batches, or utilizing distributed computing to improve computational efficiency [224]. 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 [225] 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], [226], 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 [227]. Then the relevant experts and scholars also introduced this stochastic idea to the second-order optimization methods [124], [125], [228] and achieved good results.

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. Through 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 [227]. Another version of 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 [229]. The stochastic version of conjugate gradient is a potential optimization method and is still worth studying.

VI. Conclusion

This paper introduces and summarizes the frequently used optimization methods from the perspective of machine learning, and studies their applications in various fields of machine learning. Firstly, we describe 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 describe the applications of the optimization methods in different machine learning scenarios and the approaches to improve their performance. Finally, we discuss some challenges and open problems in machine learning optimization methods.

References

[1] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, 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," *Journal of Machine Learning Research*, vol. 18, 2018.

[3] D. F. Shanno, "Conditioning of quasi-Newton methods for function minimization," *Mathematics of Computation*, vol. 24, 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 Journal on Scientific Computing*, vol. 41, pp. 2239–2269, 2019.

[5] J. Pajarinen, H. L. Thai, R. Akrour, J. Peters, and G. Neumann, "Compatible natural gradient policy search," *Machine Learning*, pp. 1–24, 2019.

[6] J. E. Dennis, Jr, and J. J. Moré, "Quasi-Newton methods, motivation and theory," *SIAM Review*, vol. 19, pp. 46–89, 1977.

[7] J. Martens, "Deep learning via Hessian-free optimization," in *International Conference on Machine Learning*, 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*, 2016.

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

[10] R. Bollapragada, R. H. Byrd, and J. Nocedal, "Exact and inexact subsampled newton methods for optimization," *IMA Journal of Numerical Analysis*, vol. 1, pp. 1–34, 2018.

[11] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations," *Journal of Global Optimization*, vol. 56, 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 Journal on Optimization*, vol. 29, pp. 965–993, 2019.

[13] Y. LeCun and L. Bottou, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 1998.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[15] P. Sermanet and D. Eigen, "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *International Conference on Learning Representations*, 2014.

[16] A. Karpathy and G. Toderici, "Large-scale video classification with convolutional neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[17] Y. Kim, "Convolutional neural networks for sentence classification," in *Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1746–1751.

[18] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, pp. 221–231, 2012.

[19] S. Lai, L. Xu, and K. Liu, "Recurrent convolutional neural networks for text classification," in *Association for the Advancement of Artificial Intelligence*, 2015, pp. 2267–2273.

[20] K. Cho and B. Van Merriënboer, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1724–1734.

[21] P. Liu and X. Qiu, "Recurrent neural network for text classification with multi-task learning," in *International Joint Conferences on Artificial Intelligence*, 2016, pp. 2873–2879.

[22] A. Graves and A.-r. Mohamed, "Speech recognition with deep recurrent neural networks," in *International Conference on Acoustics, Speech and Signal processing*, 2013, pp. 6645–6649.

[23] K. Gregor and I. Danihelka, "Draw: A recurrent neural network for image generation," *arXiv preprint arXiv:1502.04623*, 2015.

[24] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *arXiv preprint arXiv:1601.06759*, 2016.

[25] A. Ullah and J. Ahmad, "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 Transactions on Neural Networks and Learning Systems*, vol. 27, no. 2, pp. 214–224, 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 Transactions on Neural Networks and Learning Systems*, vol. 26, no. 12, pp. 3227–3238, 2015.

[28] Y. Xia and J. Wang, "Robust regression estimation based on low-dimensional recurrent neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 5935–5946, 2018.

[29] Y. Xia, J. Wang, and W. Guo, "Two projection neural networks with reduced model complexity for nonlinear programming," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–10, 2019.

[30] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[31] M. D. Zeiler, "AdaDelta: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[32] T. Tieleman and G. Hinton, "Divide the gradient by a running average of its recent magnitude," COURSERA: Neural Networks for Machine Learning, pp. 26–31, 2012.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2014, pp. 1–15.

[34] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and beyond," in *International Conference on Learning Representations*, 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*, 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 *Advances in Neural Information Processing Systems*, 2012, pp. 2663–2671.

[37] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems*, 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*, 2017.

[39] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. 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 *International Conference on Neural Information Processing*, 2012, pp. 126–133.

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[42] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[43] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, pp. 1–127, 2009.

[44] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: an overview," in *SAI Intelligent Systems Conference*, 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, Technische Universität München, München, Germany, 1987.

[46] T. Schaul and J. Schmidhuber, "Metalearning," *Scholarpedia*, vol. 5, pp. 46–50, 2010.

[47] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*, 2017, pp. 1126–1135.

[48] O. Vinyals, "Model vs optimization meta learning," http://metalearning-symposium.ml/files/vinyals.pdf, 2017.

[49] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," in *Advances in Neural Information Processing Systems*, 1994, pp. 737–744.

[50] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *International Conference on Machine Learning WorkShop*, 2015, pp. 1–30.

[51] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 3630–3638.

[52] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4077–4087.

[53] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *International Conference on Machine Learning*, 2016, pp. 1842–1850.

[54] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in *International Conference on Learning Representations*, 2015, pp. 1–15.

[55] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, 2016, pp. 3981–3989.

[56] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *International Conference on Learning Representations*, 2016, pp. 1–11.

[57] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[58] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *Journal of Machine Learning Research*, vol. 14, pp. 1303–1347, 2013.

[59] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Computer Vision and Pattern Recognition*, 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 *Advances in Neural Information Processing Systems*, 2017, pp. 5279–5288.

[61] T. Chen, E. Fox, and C. Guestrin, "Stochastic gradient Hamiltonian Monte Carlo," in *International Conference on Machine Learning*, 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," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[63] W. Yin and H. Schütze, "Multichannel variable-size convolution for sentence classification," in *Conference on Computational Language Learning*, 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 *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 1794–1801.

[65] Y. Bazi and F. Melgani, "Gaussian process approach to remote sensing image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, pp. 186–197, 2010.

[66] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3642–3649.

[67] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, pp. 100–108, 1979.

[68] S. Guha, R. Rastogi, and K. Shim, "ROCK: A robust clustering algorithm for categorical attributes," *Information Systems*, vol. 25, pp. 345–366, 2000.

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

[70] M. Guillaumin and J. Verbeek, "Multimodal semi-supervised learning for image classification," in *Computer Vision and Pattern Recognition*, 2010, pp. 902–909.

[71] O. Chapelle and A. Zien, "Semi-supervised classification by low density separation." in *International Conference on Artificial Intelligence and Statistics*, 2005, pp. 57–64.

[72] Z.-H. Zhou and M. Li, "Semi-supervised regression with co-training." in *International Joint Conferences on Artificial Intelligence*, 2005, pp. 908–913.

[73] A. Demiriz and K. P. Bennett, "Semi-supervised clustering using genetic algorithms," *Artificial Neural Networks in Engineering*, vol. 1, pp. 809–814, 1999.

[74] B. Kulis and S. Basu, "Semi-supervised graph clustering: a kernel approach," *Machine Learning*, vol. 74, pp. 1–22, 2009.

[75] D. Zhang and Z.-H. Zhou, "Semi-supervised dimensionality reduction," in *SIAM International Conference on Data Mining*, 2007, pp. 629–634.

[76] P. Chen and L. Jiao, "Semi-supervised double sparse graphs based discriminant analysis for dimensionality reduction," *Pattern Recognition*, vol. 61, pp. 361–378, 2017.

[77] K. P. Bennett and A. Demiriz, "Semi-supervised support vector machines," in *Advances in Neural Information processing systems*, 1999, pp. 368–374.

[78] E. Cheung, *Optimization Methods for Semi-Supervised Learning*. University of Waterloo, 2018.

[79] O. Chapelle, V. Sindhwani, and S. S. Keerthi, "Optimization techniques for semi-supervised support vector machines," *Journal of Machine Learning Research*, vol. 9, pp. 203–233, 2008.

[80] ——, "Branch and bound for semi-supervised support vector machines," in *Advances in Neural Information Processing Systems*, 2007, pp. 217–224.

[81] Y.-F. Li and I. W. Tsang, "Convex and scalable weakly labeled svms," *Journal of Machine Learning Research*, vol. 14, pp. 2151–2188, 2013.

[82] F. Murtagh, "A survey of recent advances in hierarchical clustering algorithms," *The Computer Journal*, vol. 26, pp. 354–359, 1983.

[83] V. Castro and J. Yang, "A fast and robust general purpose clustering algorithm," in *Knowledge Discovery in Databases and Data Mining*, 2000, pp. 208–218.

[84] G. H. Ball and D. J. Hall, "A clustering technique for summarizing multivariate data," *Behavioral Science*, vol. 12, pp. 153–155, 1967.

[85] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 2, pp. 37–52, 1987.

[86] I. Jolliffe, "Principal component analysis," in *International Encyclopedia of Statistical Science*, 2011, pp. 1094–1096.

[87] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, pp. 611–622, 1999.

[88] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[89] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[90] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[91] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University 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 *Advances in Neural Information Processing Systems*, 1993, pp. 836–844.

[93] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 2006.

[94] A. S. Nemirovsky and D. B. Yudin, *Problem Complexity and Method Efficiency in Optimization*. John Wiley & Sons, 1983.

[95] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM Journal on Optimization*, vol. 19, 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 *Advances in Neural Information Processing Systems*, 2009, pp. 1–9.

[97] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, pp. 400–407, 1951.

[98] C. Darken, J. Chang, and J. Moody, "Learning rate schedules for faster stochastic gradient search," in *Neural Networks for Signal Processing*, 1992, pp. 3–12.

[99] I. Sutskever, "Training recurrent neural networks," Ph.D. dissertation, University of Toronto, Ontario, Canada, 2013.

[100] Z. Allen-Zhu, "Natasha 2: Faster non-convex optimization than SGD," in *Advances in Neural Information Processing Systems*, 2018, pp. 2675–2686.

[101] R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping from saddle pointsonline stochastic gradient for tensor decomposition," in *Conference on Learning Theory*, 2015, pp. 797–842.

[102] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, pp. 1–17, 1964.

[103] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[104] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International Conference on Machine Learning*, 2013, pp. 1139–1147.

[105] 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, pp. 543–547, 1983.

[106] L. C. Baird III and A. W. Moore, "Gradient descent for general reinforcement learning," in *Advances in Neural Information Processing Systems*, 1999, pp. 968–974.

[107] C. Darken and J. E. Moody, "Note on learning rate schedules for stochastic optimization," in *Advances in Neural Information Processing Systems*, 1991, pp. 832–838.

[108] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient," *Mathematical Programming*, vol. 162, pp. 83–112, 2017.

[109] Z. Allen-Zhu and E. Hazan, "Variance reduction for faster non-convex optimization," in *International Conference on Machine Learning*, 2016, pp. 699–707.

[110] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. Smola, "Stochastic variance reduction for nonconvex optimization," in *International Conference on Machine Learning*, 2016, pp. 314–323.

[111] A. Defazio, F. Bach, and S. Lacoste-Julien, "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Advances in Neural Information Processing Systems*, 2014, pp. 1646–1654.

[112] M. J. Powell, "A method for nonlinear constraints in minimization problems," *Optimization*, pp. 283–298, 1969.

[113] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, pp. 1–122, 2011.

[114] A. Nagurney and P. Ramanujam, "Transportation network policy modeling with goal targets and generalized penalty functions," *Transportation Science*, vol. 30, pp. 3–13, 1996.

[115] B. He, H. Yang, and S. Wang, "Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities," *Journal of Optimization Theory and Applications*, vol. 106, pp. 337–356, 2000.

[116] D. Hallac, C. Wong, S. Diamond, A. Sharang, S. Boyd, and J. Leskovec, "Snapvx: A network-based convex optimization solver," *Journal of Machine Learning Research*, vol. 18, pp. 1–5, 2017.

[117] B. Wahlberg, S. Boyd, M. Annergren, and Y. Wang, "An ADMM algorithm for a class of total variation regularized estimation problems," *arXiv preprint arXiv:1203.1828*, 2012.

[118] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *Naval Research Logistics Quarterly*, vol. 3, pp. 95–110, 1956.

[119] M. Jaggi, "Revisiting Frank-Wolfe: Projection-free sparse convex optimization," in *International Conference on Machine Learning*, 2013, pp. 427–435.

[120] M. Fukushima, "A modified Frank-Wolfe algorithm for solving the traffic assignment problem," *Transportation Research Part B: Methodological*, vol. 18, pp. 169–177, 1984.

[121] M. Patriksson, *The Traffic Assignment Problem: Models and Methods*. Dover Publications, 2015.

[122] K. L. Clarkson, "Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm," *ACM Transactions on Algorithms*, vol. 6, pp. 63–96, 2010.

[123] J. Mairal, F. Bach, J. Ponce, G. Sapiro, R. Jenatton, and G. Obozinski, "SPAMS: A sparse modeling software, version 2.3," http://spams-devel.gforge.inria.fr/downloads.html, 2014.

[124] N. N. Schraudolph, J. Yu, and S. Günter, "A stochastic quasi-Newton method for online convex optimization," in *Artificial Intelligence and Statistics*, 2007, pp. 436–443.

[125] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer, "A stochastic quasi- method for large-scale optimization," *SIAM Journal on Optimization*, vol. 26, pp. 1008–1031, 2016.

[126] P. Moritz, R. Nishihara, and M. Jordan, "A linearly-convergent stochastic L-BFGS algorithm," in *Artificial Intelligence and Statistics*, 2016, pp. 249–258.

[127] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*. NBS Washington, DC, 1952.

[128] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," Carnegie Mellon University, Tech. Rep., 1994.

[129] M. Avriel, *Nonlinear Programming: Analysis and Methods*. Dover Publications, 2003.

[130] P. T. Harker and J. Pang, "A damped-Newton method for the linear complementarity problem," *Lectures in Applied Mathematics*, vol. 26, pp. 265–284, 1990.

[131] P. Y. Ayala and H. B. Schlegel, "A combined method for determining reaction paths, minima, and transition state geometries," *The Journal of Chemical Physics*, vol. 107, pp. 375–384, 1997.

[132] M. Raydan, "The barzilai and borwein gradient method for the large scale unconstrained minimization problem," *SIAM Journal on Optimization*, vol. 7, pp. 26–33, 1997.

[133] W. C. Davidon, "Variable metric method for minimization," *SIAM Journal on Optimization*, vol. 1, pp. 1–17, 1991.

[134] R. Fletcher and M. J. Powell, "A rapidly convergent descent method for minimization," *The Computer Journal*, vol. 6, pp. 163–168, 1963.

[135] C. G. Broyden, "The convergence of a class of double-rank minimization algorithms: The new algorithm," *IMA Journal of Applied Mathematics*, vol. 6, pp. 222–231, 1970.

[136] R. Fletcher, "A new approach to variable metric algorithms," *The Computer Journal*, vol. 13, pp. 317–322, 1970.

[137] D. Goldfarb, "A family of variable-metric methods derived by variational means," *Mathematics of Computation*, vol. 24, pp. 23–26, 1970.

[138] J. Nocedal, "Updating quasi-Newton matrices with limited storage," *Mathematics of Computation*, vol. 35, pp. 773–782, 1980.

[139] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical programming*, vol. 45, pp. 503–528, 1989.

[140] W. Sun and Y. X. Yuan, *Optimization theory and methods: nonlinear programming*. Springer Science & Business Media, 2006.

[141] A. S. Berahas, J. Nocedal, and M. Takác, "A multi-batch L-BFGS method for machine learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 1055–1063.

[142] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Advances in Neural Information Processing Systems*, 2008, pp. 161–168.

[143] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *Society for Industrial and Applied Mathematics Review*, vol. 60, pp. 223–311, 2018.

[144] A. Mokhtari and A. Ribeiro, "Res: Regularized stochastic BFGS algorithm," *IEEE Transactions on Signal Processing*, vol. 62, pp. 6089–6104, 2014.

[145] ——, "Global convergence of online limited memory BFGS," *Journal of Machine Learning Research*, vol. 16, pp. 3151–3181, 2015.

[146] R. Gower, D. Goldfarb, and P. Richtárik, "Stochastic block BFGS: Squeezing more curvature out of data," in *International Conference on Machine Learning*, 2016, pp. 1869–1878.

[147] R. H. Byrd, G. M. Chin, W. Neveitt, and J. Nocedal, "On the use of stochastic Hessian information in optimization methods for machine learning," *SIAM Journal on Optimization*, vol. 21, pp. 977–995, 2011.

[148] S. I. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, pp. 251–276, 1998.

[149] J. Martens, "New insights and perspectives on the natural gradient method," *arXiv preprint arXiv:1412.1193*, 2014.

[150] R. Grosse and R. Salakhudinov, "Scaling up natural gradient by sparsely factorizing the inverse fisher matrix," in *International Conference on Machine Learning*, 2015, pp. 2304–2313.

[151] J. Martens and R. Grosse, "Optimizing neural networks with Kronecker-factored approximate curvature," in *International Conference on Machine Learning*, 2015, pp. 2408–2417.

[152] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A trust region method based on interior point techniques for nonlinear programming," *Mathematical Programming*, vol. 89, pp. 149–185, 2000.

[153] L. Hei, "Practical techniques for nonlinear optimization," Ph.D. dissertation, Northwestern University, America, 2007.

[154] M. I. Lourakis, "A brief description of the levenberg-marquardt algorithm implemented by levmar," *Foundation of Research and Technology*, vol. 4, pp. 1–6, 2005.

[155] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics, 2009.

[156] C. Audet and M. Kokkolaras, *Blackbox and Derivative-Free Optimization: Theory, Algorithms and Applications*. Springer, 2016.

[157] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations," *Journal of Global Optimization*, vol. 56, pp. 1247–1293, 2013.

[158] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[159] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT press, 1998.

[160] M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stützle, and A. Winfield, *Ant Colony Optimization and Swarm Intelligence*. Springer, 2008.

[161] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific Belmont, 1999.

[162] P. Richtárik and M. Takáč, "Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function," *Mathematical Programming*, vol. 144, pp. 1–38, 2014.

[163] I. Loshchilov, M. Schoenauer, and M. Sebag, "Adaptive coordinate descent," in *Annual Conference on Genetic and Evolutionary Computation*, 2011, pp. 885–892.

[164] T. Huckle, "Approximate sparsity patterns for the inverse of a matrix and preconditioning," *Applied Numerical Mathematics*, vol. 30, pp. 291–303, 1999.

[165] M. Benzi, "Preconditioning techniques for large linear systems: a survey," *Journal of Computational Physics*, vol. 182, pp. 418–477, 2002.

[166] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," http://cvxr.com/cvx, 2014.

[167] S. Diamond and S. Boyd, "Cvxpy: A python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, pp. 2909–2913, 2016.

[168] M. Andersen, J. Dahl, and L. Vandenberghe, "Cvxopt: A python package for convex optimization, version 1.1.6," https://cvxopt.org/, 2013.

[169] J. D. Hedengren, R. A. Shishavan, K. M. Powell, and T. F. Edgar, "Nonlinear modeling, estimation and predictive control in apmonitor," *Computers & Chemical Engineering*, vol. 70, pp. 133–148, 2014.

[170] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard, "Tensorflow: a system for large-scale machine learning," in *USENIX Symposium on Operating Systems Design and Implementations*, 2016, pp. 265–283.

[171] T. Dozat, "Incorporating nesterov momentum into adam," in *International Conference on Learning Representations*, 2016, pp. 1–14.

[172] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in Adam," *arXiv preprint arXiv:1711.05101*, 2017.

[173] Z. Zhang, L. Ma, Z. Li, and C. Wu, "Normalized direction-preserving Adam," *arXiv preprint arXiv:1709.04546*, 2017.

[174] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee, "Recent advances in recurrent neural networks," *arXiv preprint arXiv:1801.01078*, 2017.

[175] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 8624–8628.

[176] J. Martens and I. Sutskever, "Training deep and recurrent networks with Hessian-free optimization," in *Neural Networks: Tricks of the Trade*, 2012, pp. 479–535.

[177] N. N. Schraudolph, "Fast curvature matrix-vector products for second-order gradient descent," *Neural Computation*, vol. 14, pp. 1723–1738, 2002.

[178] J. Martens and I. Sutskever, "Learning recurrent neural networks with Hessian-free optimization," in *International Conference on Machine Learning*, 2011, pp. 1033–1040.

[179] A. Likas and A. Stafylopatis, "Training the random neural network using quasi-Newton methods," *European Journal of Operational Research*, vol. 126, pp. 331–339, 2000.

[180] X. Liu and S. Liu, "Limited-memory bfgs optimization of recurrent neural network language models for speech recognition," in *International Conference on Acoustics, Speech and Signal Processing*, 2018, pp. 6114–6118.

[181] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[182] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[183] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, and M. Lanctot, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[184] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[185] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Cambridge University Engineering Department, Tech. Rep., 1994.

[186] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[187] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor-critic algorithms," *Automatica*, vol. 45, pp. 2471–2482, 2009.

[188] S. Thrun and L. Pratt, *Learning to Learn*. Springer Science & Business Media, 2012.

[189] M. Abdullah Jamal and G.-J. Qi, "Task agnostic meta-learning for few-shot learning," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 1–11.

[190] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," *Machine Learning*, vol. 37, pp. 183–233, 1999.

[191] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends in Machine Learning*, vol. 1, pp. 1–305, 2008.

[192] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American Statistical Association*, vol. 112, pp. 859–877, 2017.

[193] L. Bottou and Y. L. Cun, "Large scale online learning," in *Advances in Neural Information Processing Systems*, 2004, pp. 217–224.

[194] J. C. Spall, *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Wiley-Interscience, 2005.

[195] J. Hensman, N. Fusi, and N. Lawrence, "Gaussian processes for big data," in *Conference on Uncertainty in Artificial Intellegence*, 2013, pp. 282–290.

[196] J. Hensman, A. G. d. G. Matthews, and Z. Ghahramani, "Scalable variational gaussian process classification," in *International Conference on Artificial Intelligence and Statistics*, 2015, pp. 351–360.

[197] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid monte carlo," *Physics Letters B*, vol. 195, pp. 216–222, 1987.

[198] R. Neal, "MCMC using Hamiltonian dynamics," *Handbook of Markov Chain Monte Carlo*, vol. 2, pp. 113–162, 2011.

[199] M. Girolami and B. Calderhead, "Riemann manifold langevin and hamiltonian monte carlo methods," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 73, pp. 123–214, 2011.

[200] M. Betancourt, "The fundamental incompatibility of scalable Hamiltonian monte carlo and naive data subsampling," in *International Conference on Machine Learning*, 2015, pp. 533–540.

[201] S. Ahn, A. Korattikara, and M. Welling, "Bayesian posterior sampling via stochastic gradient fisher scoring," in *International Conference on Machine Learning*, 2012, pp. 1591–1598.

[202] M. D. Hoffman and A. Gelman, "The No-U-turn sampler: adaptively setting path lengths in Hamiltonian monte carlo," *Journal of Machine Learning Research*, vol. 15, pp. 1593–1623, 2014.

[203] Y. Nesterov, "Primal-dual subgradient methods for convex problems," *Mathematical Programming*, vol. 120, pp. 221–259, 2009.

[204] C. Andrieu and J. Thoms, "A tutorial on adaptive MCMC," *Statistics and Computing*, vol. 18, pp. 343–373, 2008.

[205] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: A probabilistic programming language," *Journal of Statistical Software*, vol. 76, pp. 1–37, 2017.

[206] S. L. Cotter, G. O. Roberts, A. M. Stuart, and D. White, "MCMC methods for functions: modifying old algorithms to make them faster," *Statistical Science*, vol. 28, pp. 424–446, 2013.

[207] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient Langevin dynamics," in *International Conference on Machine Learning*, 2011, pp. 681–688.

[208] N. Ding, Y. Fang, R. Babbush, C. Chen, R. D. Skeel, and H. Neven, "Bayesian sampling using stochastic gradient thermostats," in *Advances in Neural Information Processing Systems*, 2014, pp. 3203–3211.

[209] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.

[210] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[211] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1345–1359, 2010.

[212] P. Jain and P. Kar, "Non-convex optimization for machine learning," *Foundations and Trends in Machine Learning*, vol. 10, pp. 142–336, 2017.

[213] C. S. Adjiman and S. Dallwig, "A global optimization method, αbb, for general twice-differentiable constrained NLPs–I. theoretical advances," *Computers & Chemical Engineering*, vol. 22, pp. 1137–1158, 1998.

[214] C. Adjiman, C. Schweiger, and C. Floudas, "Mixed-integer nonlinear optimization in process synthesis," in *Handbook of combinatorial optimization*, 1998, pp. 1–76.

[215] T. Pock, A. Chambolle, D. Cremers, and H. Bischof, "A convex relaxation approach for computing minimal partitions," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 810–817.

[216] L. Xu and D. Schuurmans, "Unsupervised and semi-supervised multi-class support vector machines," in *Association for the Advancement of Artificial Intelligence*, 904-910, p. 13.

[217] Y. Chen and M. J. Wainwright, "Fast low-rank estimation by projected gradient descent: General statistical and algorithmic guarantees," *arXiv preprint arXiv:1509.03025*, 2015.

[218] D. Park and A. Kyrillidis, "Provable non-convex projected gradient descent for a class of constrained matrix optimization problems," *arXiv preprint arXiv:1606.01316*, 2016.

[219] P. Jain, P. Netrapalli, and S. Sanghavi, "Low-rank matrix completion using alternating minimization," in *ACM Annual Symposium on Theory of Computing*, 2013, pp. 665–674.

[220] M. Hardt, "Understanding alternating minimization for matrix completion," in *IEEE Annual Symposium on Foundations of Computer Science*, 2014, pp. 651–660.

[221] M. Hardt and M. Wootters, "Fast matrix completion without the condition number," in *Conference on Learning Theory*, 2014, pp. 638–678.

[222] S. Balakrishnan, M. J. Wainwright, and B. Yu, "Statistical guarantees for the em algorithm: From population to sample-based analysis," *The Annals of Statistics*, vol. 45, pp. 77–120, 2017.

[223] Z. Wang, Q. Gu, Y. Ning, and H. Liu, "High dimensional expectation-maximization algorithm: Statistical optimization and asymptotic normality," *arXiv preprint arXiv:1412.8729*, 2014.

[224] 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," *arXiv preprint arXiv:1609.04836*, 2016.

[225] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[226] J. Martens, *Second-Order Optimization For Neural Networks*. University of Toronto (Canada), 2016.

[227] N. N. Schraudolph and T. Graepel, "Conjugate directions for stochastic gradient descent," in *International Conference on Artificial Neural Networks*, 2002, pp. 1351–1356.

[228] A. Bordes, L. Bottou, and P. Gallinari, "SGD-QN: Careful quasi-Newton stochastic gradient descent," *Journal of Machine Learning Research*, vol. 10, pp. 1737–1754, 2009.
[229] X. Jin, X. Zhang, K. Huang, and G. Geng, "Stochastic conjugate gradient algorithm with variance reduction," *IEEE transactions on Neural Networks and Learning Systems*, pp. 1–10, 2018.