Journal of the Operations Research Society of China (2020) 8:341–351
https://doi.org/10.1007/s40305-019-00285-6

How Can Machine Learning and Optimization Help Each Other Better?

Zhou-Chen Lin[1]

Received: 23 January 2019 / Revised: 30 June 2019 / Accepted: 22 November 2019 /
Published online: 19 December 2019

Abstract
Optimization is an indispensable part of machine learning as machine learning needs to solve mathematical models efficiently. On the other hand, machine learning can also provide new momenta and new ideas for optimization. This paper aims at investigating how to make the interactions between optimization and machine learning more effective.

Keywords Optimization · Machine learning · Generalization ability · Data

Mathematics Subject Classification 49M37 · 65K05 · 65K10 · 90C06 · 90C15 · 90C26 · 90C30 · 90C35 · 90C90

1 Introduction

It is well known in the machine learning community that optimization is an indispensable part of machine learning. P. Domingos, AAAI Fellow and Professor of University of Washington, once wrote a famous equation [1]:

$$\text{Machine Learning} = \text{Representation} + \text{Optimization} + \text{Evaluation},$$

which clearly shows the critical role of optimization in machine learning. The recent boom of deep learning is also partially attributed to the advance of optimization techniques [2]. On the other hand, the huge demand of machine learning on optimization techniques also gives the optimization community huge impetus, even new ideas. For

Zhou-Chen Lin was supported by the National Natural Science Foundation of China (Nos. 61625301 and 61731018).

✉ Zhou-Chen Lin
zlin@pku.edu.cn

[1] Key Laboratory of Machine Perception, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China

example, the popularity of support vector machines (SVMs) in 1990s brought back the attention on quadratic programming. An SVM is actually the following optimization problem:

$$\min_{w} \frac{1}{2}\|w\|_2^2 \quad \text{s.t.} \quad y_i \langle w, x_i \rangle \geqslant 1, \quad i = 1, \cdots, N, \tag{1.1}$$

where $\{(x_i, y_i)\}_{i=1}^N$ is the training set with $y_i \in \{1, -1\}$ being the label of sample x_i. The recent boom of deep learning further stimulated the interest on stochastic gradient descent, nonconvex optimization and distributed optimization. Deep learning mainly finds the weights of a deep neural network (DNN) such that the difference between the outputs of the DNN and the ground truth is minimized over the training set. A typical DNN is the fully connected feed-forward neural network (Fig. 1)

$$x_i^{k+1} = \phi(W^k x_i^k), \quad k = 0, \cdots, K-1,$$

where x_i^k is the response of nodes (called neurons) at layer k when the input is the ith sample (x_i^0 and x_i^K are the input and the output, respectively), W^k are the weights between layers k and $k+1$, and ϕ is a nonlinear non-decreasing function (called the activation function). Then the average difference $\frac{1}{N}\sum_{i=1}^N l(x_i^K, y_i)$ between the output x_i^K of the DNN and the ground truth y_i, called loss, is minimized w.r.t. the weights $\{W^k\}_{k=0}^{K-1}$, where $l(\cdot, \cdot)$ is the loss function. Some machine learning techniques also appear in optimization algorithms. This paper aims at investigating, in my personal and limited view, how the two communities can help each other better.

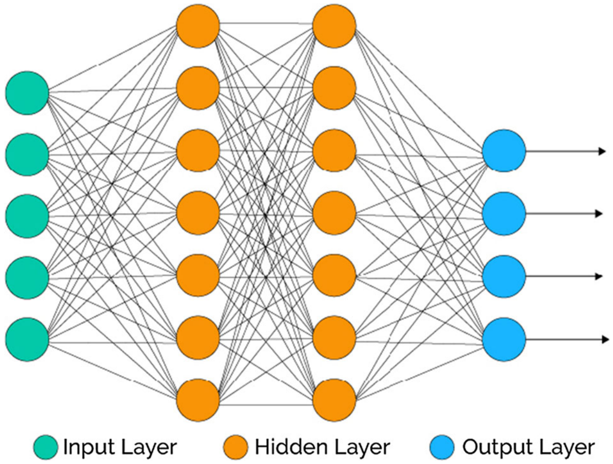

Fig. 1 An illustration of a fully connected feed-forward DNN. The number of hidden layers can be arbitrary

2 Different Optimization Goals between the Machine Learning and the Optimization Communities

To proceed, one needs to know the difference of optimization goals in the machine learning and the pure optimization communities. In machine learning, computing a set of parameters that control a learning model at a high numerical accuracy is not the goal, at least not the only one. Of course, if using the same time algorithm A achieves a higher numerical accuracy than algorithm B then algorithm A is surely preferred over algorithm B. However, very often machine learning people would stop algorithm A when a lower but still acceptable numerical accuracy is achieved, in order to save time. The reason is that machine learning tasks typically emphasize on low classification or clustering errors, which is usually measured by percentage or other criteria. These numerical quantities typically are reported in low numerical accuracies due to limited number of data, which are usually expensive to collect. For example, the percentage usually uses at most two decimal digits. So computing the parameters is just a midway, their high numerical accuracy does not matter much in the latter evaluation of classification or clustering errors.

The classification error is further differentiated as training error and testing error. The training error is the classification error on the training set, while the testing error is on the testing set. The testing error is what machine learning people care the most as it is related to the generalization ability of a learning model, i.e., how well a trained learning model can perform on new data. If the training error is high, a learning model is said to underfit. If the training error is low but the testing error is high, a learning mode is deemed to overfit. Ideally we want to have a learning model that is neither underfit nor overfit. In contrast, pure optimization people, who may not care much about the physical meaning of learning models, may focus on minimizing the training error.

Even if one is aware that minimizing the testing error is the most important, there is still a caveat. Very often, a learning model also has hyper-parameters which are not included in the optimization algorithm, such as the number of layers, the number of filters in each layer and the width of filters in DNNs (in this example the parameters that the back-propagation algorithm optimizes are the weights of the neural network). Then there should be a *validation set*, which is a subset of the training set, for tuning the hyper-parameters. Namely, the hyper-parameters are chosen as those which together with the parameters that the optimization algorithm computes on the remaining part of the training set result in the lowest classification error on the validation set. Even if there are no hyper-parameters, a validation set may also be necessary. For example, the optimization algorithm computing on the (remaining part of) training set may be terminated when the validation error starts to increase. This is a kind of regularization called early stopping in statistics. Such a way is a remedy of the imperfection of the learning model whose optimal parameters do not correspond to the optimal classification/clustering performance.

That machine learning people care the testing error rather than the training error can add significant difficulties to optimization. A remarkable example is that when training deep neural networks, machine learning people may prefer a local minimizer in a relatively flat region to the global minimizer in a deep valley (Fig. 2). This is because the objective function on the testing data, assuming that the distribution of

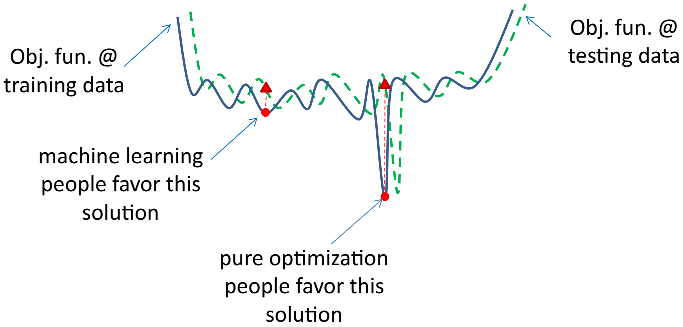
Obj. fun. @ training data
Obj. fun. @ testing data
machine learning people favor this solution
pure optimization people favor this solution

Fig. 2 When training deep neural networks, pure optimization people favor the global minimizer which may be in a deep valley, but machine learning people favor a local minimizer in a relatively flat region. This is because a local minimizer in a relatively flat region may result in a much less increase in classification error (indicated by the dashed segments between triangles and circles). The solid curve and the dashed curve indicate the objective functions on the training data and the testing data, respectively

the testing data is the same as the training data, should be a perturbation of that on the training data. So a local minimizer in a relatively flat region will result in less increase in the classification error than the global minimizer in a deep valley does, which means that the former solution has a better generalization ability while the latter one may overfit.

The differences of optimization concept in the machine learning and the optimization communities are also discussed in [3], where the authors listed some desirable properties of an optimization algorithm from the machine learning perspective:

- good generalization,
- scalability to large problems,
- good performance in practice in terms of execution times and memory requirements,
- simple and easy implementation of algorithm,
- exploitation of problem structure,
- fast convergence to an approximate solution of model,
- robustness and numerical stability for the class of machine learning models attempted,
- theoretically known convergence and complexity.

3 How Can Optimization Help Machine Learning Better?

It is interesting to note that optimization does not just help machine learning in computing more efficiently. It can also help in other ways. For example, the iteration process can be unfolded into deep neural networks. So optimization can contribute to deep neural network architecture design [4–6]. Figure 3 shows some optimization inspired DNNs for image recognition. Nonetheless, in the following I still focus on the computing aspect.

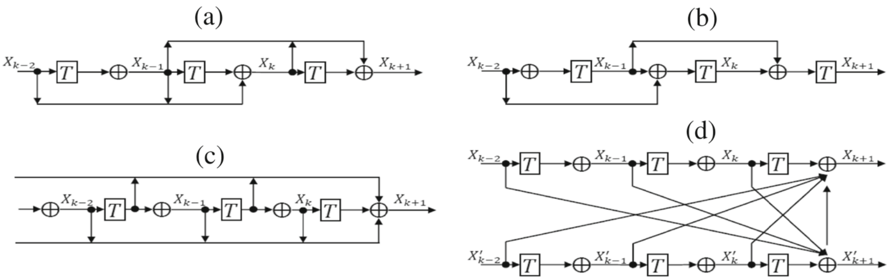

Fig. 3 Four DNNs inspired by optimization algorithms, adapted from [5]. (a) is inspired by the heavy-ball algorithm [7]. (b) and (c) are inspired by Nesterov's accelerated gradient descent method [8], written in different but equivalent forms. (d) is inspired by the ADMM method

3.1 Consider Characteristics of Data

Optimization people know well that an efficient algorithm should take the structure of problem into consideration. However, examining the structure of problems may not be enough. Similar to the celebrated "No Free Lunch" theorem for machine learning [9], which states that no learning model is universally better than other models, there is also "No Free Lunch" theorem for optimization [10] proposed by the same author, saying that no optimization algorithm is uniformly faster than other algorithms. This conclusion assumes that algorithms are tested on all possible problems, which include all possible choices of "data" even for a particular problem. For example, in the LASSO problem:

$$\min_x \|Ax - b\|_2 \quad \text{s.t.} \quad \|x\|_1 \leqslant \varepsilon, \tag{3.1}$$

A, b and ε are all "data". Thus, just considering the structure of the problem is not enough to have an efficient algorithm. The characteristics of "data" should also be considered when designing an efficient algorithm for a particular problem. Of course, some algorithms do consider part of the characteristic of "data" in some way. For example, strong convexity is usually assumed by many algorithms. However, using a single strong convexity modulus in the algorithm is no doubt too rough to depict the detailed characteristics of "data". Take the LASSO problem (3.1) above as a concrete example, if A is of full column rank then the objective function is strongly convex. However, that A is of full column rank cannot depict the detailed distribution of data. Thus, we cannot fully exploit the characteristics of data. Even worse, in most cases A should have more columns than rows, thus the strong convexity does not hold if one does not realize that data are normally low-dimensional.

Unfortunately, how to characterize a data set completely is still an unsolved problem. In statistics, mean and variance are typical quantities. Higher-order statistics include skew and kurtosis. For more complex data, we may have to investigate their manifold structure as manifold learning does [11] and even the multi-manifold structure [12,13] (Fig. 4).

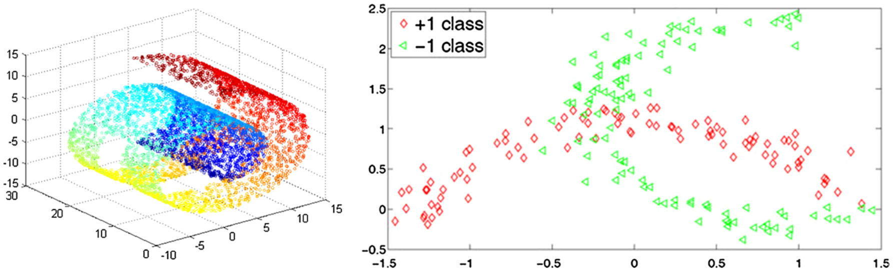

Fig. 4 Manifold data (left) and multi-manifold data (right)

3.2 Structured Nonconvex Optimization

Although convex optimization is beautiful and is effective for many machine learning models, such as SVMs (1.1) and sparse learning (e.g., feature selection (3.1)), nowadays nonconvex learning models are prevalent. The most prominent example is the DNNs which have more or less revolutionized many areas of artificial intelligence. The reason is that people build models out of their intuition and insights, rather than focusing on their mathematical properties. Then the traditional general nonconvex optimization theories become quite inadequate as they do not consider the structure of the problems much, as done in convex optimization. Only superficial results can be obtained, such as only KKT points (for constrained problems) or saddle points (for unconstrained problems) can be achieved by subsequence. Such conclusions are not quite helpful for machine learning people who want to know the quality of the solution, e.g., how close is the solution to the globally optimal one, what are the distributions of the saddle points, and how robust is the solution subject to data perturbation, etc.? Utilizing the structure of problems definitely helps and is necessary. For example, it can be proven that for some nonconvex low-rank problems, such as matrix sensing, matrix completion and robust PCA

$$\min_{U,V} f(UV^T) + Q(U, V),$$

where U and V both have r columns and $\mathbf{M} = UV^T$ is the low-rank matrix to estimate, there is no spurious local minima [14,15]. The linear convergence or even the accelerated linear convergence in solving these problems can be proven by exploiting the special structures such as restricted strong convexity [16]: f is restricted μ-strongly convex on a set S if

$$f(y) \geqslant f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|^2, \quad \forall x, y \in S.$$

The landscape of some problems can also be analyzed [17]. In the machine learning community, there has been much work on approaching and escaping saddle points [18]. Some convex optimization methods can be adapted for nonconvex problems, such as nonconvex accelerated proximal gradient [19] and nonconvex alternating direction method of multipliers (ADMM) [20,21]. The recent introduction of

Kurdyka–Łojasiewicz condition for nonconvex problems is a remarkable advance [22]. However, more machinery should be invented for analyzing nonconvex problems.

3.3 Consider Real Computing Environments

Machine learning is a practical field. When proposing a learning model, it is expected that it can be efficiently implemented in real computing environments, which typically have hard limitation on computing resources, such as storage, memory, communication bandwidths and even fault. Without considering the computing environments, an ideal optimization algorithm, e.g., the interior point method, that performs well on a single computer may be ineffective or even useless when dealing with big data. There has been regained interest on randomized, distributed and asynchronous algorithms, which considers computing with partial data (e.g., stochastic gradient descent [23] and stochastic ADMM [24]), minimizing communication [25], dealing with the delays in communication [26] and even handling breakdown of some computing agents [27]. Nonetheless, more involvement in optimization people is needed. Most of the work by optimization people still assumes ideal computing environments or investigates too much details on some parameter setting (e.g., dynamic choice of stepsize) which is actually not quite important in reality.

3.4 Automated Machine Learning

Automated machine learning (AutoML) is the process of automating the end-to-end process of applying machine learning to real-world problems. In a typical machine learning application, one must do data preprocessing, feature extraction and feature selection, etc., appropriately such that the data fit for a target machine learning task. To achieve satisfactory results, practitioners further have to try different learning models, tune their hyper-parameter and optimize their normal parameters. Most of these steps are often beyond the abilities of non-experts. So AutoML was proposed as an artificial intelligence based solution to the ever-growing challenge of applying machine learning. Automating the end-to-end process of applying machine learning offers the advantages of producing simpler solutions, faster creation of those solutions and models that often outperform models that are manually designed. AutoML can significantly lower the bar of applying machine learning techniques and make machine learning much more accessible to non-experts. Thus, it represents one of the future directions of machine learning [28].

There have been some softwares that can do part of the AutoML steps. For example, H2O AutoML and MLR can do hyper-parameter optimization and model selection; Auto-WEKA, Auto-SKLEARN, Firefly.ai, TPOT, TransmogrifAI and RECIPE can do full pipeline optimization; DEVOL, Google AutoML (Fig. 5) and Auto Keras can do deep neural network architecture search for different types of data (such as image/video, text and natural language). However, in general AutoML is still in its infancy and is using very naive methods. For example, in hyper-parameter optimization, grid search is often used; in pipeline optimization and deep neural network architecture search, genetic programming is often used. For deep neural network

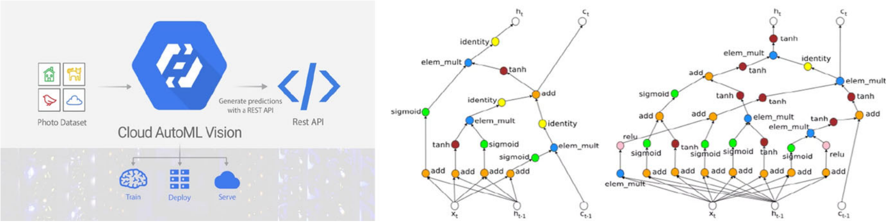

Fig. 5 Google AutoML for Vision (left) and networks found by AutoML (right)

architecture search, remarkable progress has been made [29]. For example, by using differential architecture search, a good architecture can be found in 4 GPU*days, while the evolutionary learning based search and reinforcement learning based search need 3,150 GPU*days and 1,800 GPU*days, respectively [30]. However, more efficient and more universal search strategies are still desired. Generally speaking, AutoML is a highly complex optimization system (Fig. 6). It involves multiple kinds of optimization, such as discrete optimization (e.g., for searching the hyper-parameters and topology of networks), nonconvex optimization (e.g., for optimizing the weights of networks), bi-level optimization (e.g., for evaluating the performance on the test set) and distributed optimization, etc. Therefore, AutoML is a great test bed for various optimization techniques.

4 How Can Machine Learning Help Optimization Better?

While the demand from machine learning on more practical optimization techniques itself is a propeller of the optimization community, e.g., the revival of ADMM [20,21, 24,31], machine learning can also bring up new optimization techniques. For example, the algorithms like stochastic variance reduced gradient [32] (which is to use

$$\tilde{\nabla} f(x_k^s) = \nabla f_{i_{k,s}}(x_k^s) - \nabla f_{i_{k,s}}(\tilde{x}^s) + \frac{1}{M}\sum_{i=1}^{M} \nabla f_i(\tilde{x}^s),$$

as the descent direction of the objective function $\frac{1}{M}\sum_{i=1}^{M} f_i(x)$, where i_k is uniformly randomly chosen in $\{1, \cdots, M\}$) and AdaGrad [33] (which is

$$x_{k+1} = x_k - \eta(G_k + \varepsilon I)^{-1/2}\nabla f(x_k),$$

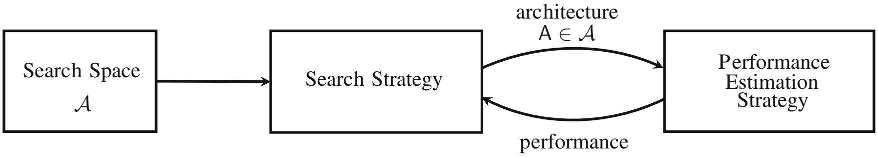

Fig. 6 Abstract illustration of Neural Architecture Search methods, adapted from [29]

where G_k is a diagonal matrix whose diagonal entries are the sum of the squares of the past gradients) are proposed by the machine learning people but are relatively new to the optimization people. Other kinds of new techniques come from the data-driven nature of machine learning. Unfortunately, currently machine learning inspired optimization algorithms still fall in the category of heuristic algorithms. Rigorous analysis is yet to be done. This is somewhat related to the issue described in Sect. 3.1.

4.1 Optimization by Learning

For many problems with high computational complexity, such as combinatorial problems which are NP hard and nonconvex problems with numerous stationary points, there is no efficient way to find their globally optimal solutions. Many heuristic approaches can apply, including machine learning methods which work directly as the solver. The basic idea follows the traditional machine learning paradigm: first prepare input–output training samples and then train a nonlinear mapping. Then given a new input the mapping can predict a solution. There has been some preliminary work along this line [34,35].

4.2 Learning Based Optimization

The other way that machine learning can contribute is to use machine learning techniques to design traditional optimization methods whose parameters are best for a particular type of data. For example, Li and Malik [36] noticed that most of the gradient descent methods use linear combination of past gradients. So they utilized reinforcement learning to learn the combination coefficients. The other independent work by Andrychowicz et al. [37] used a special type of recurrent neural network called long short-term memory (LSTM) to design gradient descent algorithms. Both work showed faster convergence on give data than hand-crafted algorithms. It is not hard to see that some critical parameters, such as step sizes, can also be learned rather than adopting a heuristic rule. Some preliminary work includes Differentiable Linearized ADMM [6] and Flexible Iterative Modularization Algorithm [38]. They both relax some parameters as learnable ones and even introduce neural networks to replace linear transforms. Notably, they both have convergence guarantees under some mild assumptions.

4.3 Automatic Code Generation

There has been some work on automatic code generation using machine learning techniques. For example, Google DeepMind developed a Neural Programmer-Interpreter, a recurrent and compositional neural network that learns to represent and execute programs [39]. Although it is still only capable of generating simple programs, it is expected that more and more complex programs can be generated by AI in the future [40,41]. It is possible that optimization codes can also be generated automatically, at least for some types of problems.

5 Conclusions

By the exposition above, it is clear that optimization and machine learning will benefit each other better if more researchers are keen on the development in the other field. Both fields will grow stronger, if they are open to each other rather than being defensive due to worrying of being invaded.

Acknowledgements This paper is based on my talk at Wuyishan Workshop “New Computing-Driven Opportunities for Optimization”, Aug. 16, 2018. I thank Prof. Ya-Xiang Yuan for encouraging me to enrich the materials and publish. Huan Li also contributed to the paper.

References

[1] Domingos, P.: A few useful things to know about machine learning. Commun. ACM **55**, 78–87 (2012)

[2] LeCun, Y., Bengio, Y., Hinton, G.E.: Deep learning. Nature **521**, 436–444 (2015)

[3] Bennett, K.P., Parrado-Hernández, E.: The interplay of optimization and machine learning research. Journal of Machine Learning Research **7**, 1265–1281 (2006)

[4] Yang, Y., Sun, J., Li, H., Xu, Z.: Deep ADMM-net for compressive sensing MRI. In: Advances in Neural Information Processing Systems, pp. 10–18 (2016)

[5] Li, H., Yang, Y., Lin, Z.: Optimization algorithm inspired deep neural network structure design. In: Asian Conference on Machine Learning, pp. 614–629 (2018)

[6] Xie, X., Wu, J., Zhong, Z., Liu, G., Lin, Z.: Differentiable linearized ADMM. In: International Conference on Machine Learning, pp. 6902–6911 (2019)

[7] Polyak, B.: Some methods of speeding up the convergence of iteration methods. USSR Comput. Math. Math. Phys. **4**, 1–17 (1964)

[8] Nesterov, Y.: A method of solving a convex programming problem with convergence rate $O(1/k^2)$. Soviet Mathematics Doklady **27**, 372–376 (1983)

[9] Wolpert, D.H.: The lack of a prior distinctions between learning algorithms. Neural Comput. **8**, 1341–1390 (1996)

[10] Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**, 67–82 (1997)

[11] Wang, J.: Geometric Structure of High-Dimensional Data and Dimensionality Reduction. Springer, Berlin (2012)

[12] Liu, G., Lin, Z., Yan, S., Sun, J., Ma, Y.: Robust recovery of subspace structures by low-rank representation. IEEE Trans. Pattern Anal. Mach. Intell. **35**, 171–184 (2013)

[13] Yin, M., Gao, J., Lin, Z.: Laplacian regularized low-rank representation and its applications. IEEE Trans. Pattern Anal. Mach. Intell. **38**, 504–517 (2016)

[14] Sun, R., Luo, Z.-Q.: Guaranteed matrix completion via nonconvex factorization. In: The IEEE Symposium on Foundations of Computer Science, pp. 270–289 (2015)

[15] Ge, R., Jin, C., Zheng, Y.: No spurious local minima in nonconvex low rank problems: a unified geometric analysis. In: International Conference on Machine Learning, pp. 1233–1242 (2017)

[16] Li, H., Lin, Z.: Provable accelerated gradient method for nonconvex low rank optimization. Mach. Learn. **103**, 1–30 (2019)

[17] Ge, R., Ma, T.: On the optimization landscape of tensor decompositions. In: Advances in Neural Information Processing Systems, pp. 3656–3666 (2017)

[18] Jin, C., Ge, R., Netrapalli, P., Kakade, S.M., Jordan, M.I.: How to escape saddle points efficiently. In: International Conference on Machine Learning, pp. 1724–1732 (2017)

[19] Li, H., Lin, Z.: Accelerated proximal gradient methods for nonconvex programming, In: Advances in Neural Information Processing Systems, pp. 379–387 (2015)

[20] Hong, M., Luo, Z., Razaviyayn, M.: Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. SIAM J. Optim. **26**, 337–364 (2016)

[21] Wang, Y., Yin, W., Zeng, J.: Global convergence of ADMM in nonconvex nonsmooth optimization. J. Sci. Comput. **78**, 29–63 (2019)

[22] Attouch, H., Bolte, J., Redont, P., Soubeyran, A.: Proximal alternating minimization and projection methods for nonconvex problems: an approach based on the Kurdyka–Lojasiewicz inequality. Math. Oper. Res. **35**, 438–457 (2010)

[23] Bottou, L., Curtis, F.E., Nocedal, J.: Optimization methods for large-scale machine learning. SIAM Rev. **360**, 223–311 (2018)

[24] Fang, C., Cheng, F., Lin, Z.: Faster and non-ergodic $O(1/K)$ stochastic alternating direction method of multipliers In: Advances in Neural Information Processing Systems, pp. 4479–4488 (2017)

[25] Lan, G., Lee, S., Zhou, Y.: Communication-efficient algorithms for decentralized and stochastic optimization. Math. Program. **158**, 1–48 (2018)

[26] Fang, C., Lin, Z.: Parallel asynchronous stochastic variance reduction for nonconvex optimization. In: AAAI Conference on Artificial Intelligence, pp. 794–800 (2017)

[27] Su, L., Vaidya, N.H.: Fault-tolerant multi-agent optimization: Optimal iterative distributed algorithms. In: Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, pp. 425–434 (2016)

[28] Hutter, F., Caruana, R., Bardenet, R., Bilenko, M., Guyon, I., Kegl, B., Larochelle, H.: AutoML 2014 @ ICML. In: AutoML 2014 Workshop @ ICML (2014)

[29] Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: a survey. J. Mach. Learn. Res. **20**, 1–21 (2019)

[30] Liu, H., Simonyan, K., Yang, Y.: DARTS: Differentiable architecture search, In: International Conference on Learning Representations (2019) arXiv:1806.09055v2

[31] Lin, Z., Liu, R., Su, Z.: Linearized alternating direction method with adaptive penalty for low-rank representation. In: Advances in Neural Information Processing Systems, pp. 612–620 (2011)

[32] Johnson, R., Zhang, T.: Accelerating stochastic gradient descent using predictive variance reduction. News. Physiol. Sci. **1**(3), 315–323 (2013)

[33] Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**, 2121–2159 (2011)

[34] Rosenfeld, N., Balkanski, E., Globerson, A., Singer, Y.: Learning to optimize combinatorial functions. In: International Conference on Machine Learning, pp. 4371–4380 (2018)

[35] Cassioli, A., Lorenzo, D.D., Locatelli, M., Schoen, F., Sciandrone, M.: Machine learning for global optimization. Computational Optimization and Applications **51**, 279–303 (2012)

[36] Li, K., Malik, J.: Learning to optimize. In: International Conference on Learning Representation (2017)

[37] Andrychowicz, M., Denil, M., Colmenarejo, S.G., Hoffman, M.W., Pfau, D., Schaul, T., de Freitas, N.: Learning to learn by gradient descent by gradient descent. In: Advances in Neural Information Processing Systems, pp. 3981–3989 (2016)

[38] Liu, R., Cheng, S., He, Y., Fan, X., Lin, Z., Luo, Z.: On the convergence of learning-based iterative methods for nonconvex inverse problems. IEEE Trans. Pattern Anal. Mach. Intell (2018) arXiv:1808.05331

[39] Reed, S., de Freitas, N.: Neural programmer-interpreters. In: International Conference on Learning Representation (2016) arXiv:1511.06279

[40] Cai, J., Shin, R., Song, D.: Making neural programming architectures generalize via recursion. In: International Conference on Learning Representations (2017) arXiv:1704.06611

[41] Li, C., Tarlow, D., Gaunt, A.L., Brockschmidt, M., Kushman, N.: Neural program lattices. In: International Conference on Learning Representation, Paris, 4–6 June (2017)