ML702 Project
Literature review: Gradient-based Bi-level optimization

Zhenhao Chen, William de Vazelhes, Boyang Sun
Mohamed bin Zayed University of Artificial Intelligence
Abu Dhabi, United Arab Emirates

{Zhenhao.Chen,William.Vazelhes,Boyang.Sun}@mbzuai.ac.ae

1 Introduction

Classical optimization algorithms are essential in machine learning: one of the most common methods for training a machine learning model is empirical risk minimization, i.e. training the model to optimize a cost function on a training set. But more advanced optimization techniques are often necessary. For instance one might want to also optimize the error on a validation set, to find the hyperparameters that are best for generalization [1]. One might also want to learn a pre-trained model or hyperparameters *such that* the model will learn efficiently on any given dataset from a family of datasets, which is called meta-learning [2]. For all those problems, bi-level optimization is the method of choice. There are many approaches developed to solve the problem, including random search [3], Bayesian optimization [4] etc. Among those methods, gradient-based techniques has drawn lots of attention in solving Bi-level optimization problem due to its superiority at speed and ability handling high-dimensional data [5]. In this paper, we will focus on gradient-based methods.

Our survey is organized in the following way: we first introduce the basic formulation of bi-level optimization problem and gives a brief introduction of its background. in Section 2, we detail the different ways to express bilevel optimization from a mathematical perspective. We will first see how those methods can be unified into a single framework, which transforms the bi-level problem into a single level problem. Among this framework, there are two main ways to proceed, either using implicit gradient methods, or explicit gradient methods. In the former one, one considers that the inner problem is at equilibrium, that is, stays at the optimum, for each value of the outer variable. In the latter one, one considers the inner problem as a dynamic system, through which we can differentiate by unrolling the iterations from the inner optimization. We will first present implicit gradient methods in section 2.1. Then, we will detail explicit gradient methods in Section 2.2.

We then turn to applications of bilevel optimization in Section 3. We will first present Meta-Learning in Section 3.1, in which one (meta-)learns a model (outer problem) *such that* it will learn efficiently (inner problem) on several tasks. Then, we will present Hyper-parameter optimization in Section 3.2, in which one tries to find the best hyper-parameters for a task, on a validation set (outer problem), for models trained on a train set with those hyperparameters (inner problem). Finally, we will turn to Neural Architecture Search in Section 3.3, in which we try to learn (outer problem) a neural architecture (that is, number of connections, number of layers, etc.) *such that* they will learn efficiently (inner problem), and have low generalization error.

1.1 General background on bi-level optimization

Bi-level optimization is generally explained as optimizing two level tasks while one is nested in the constraint of the other. Mathematically, it can be expressed as:

$$\min_{x\in\mathcal{X}} F(x,y) \quad s.t. \quad y \in \min_{y\in\mathcal{Y}} f(x,y) \tag{1}$$

where $F : R^m \times R^n \to R$ is a continuous function called the upper-level(UL) problem,. and the feasible solution $x \in R^n$ referred as the UL variable. $f : R^m \times R^n \to R$ is the nested problem which is also a continuous function. Correspondingly, we consider it as the lower-level(LL) problem and $y \in R^m$ is referred as lower-level(LL) variable.

The insight behind the above formulation is easy to understand: the UL problem chooses its variable $x = \hat{x}$ first, the LL will be optimized corresponding to the given $\hat{x}$ and return a value $\hat{y}$. Afterwards, the UL problem will be reoptimized and update according to artificially designed algorithms. i.e, the optimization of LL problem must depend on the choice of UL variable. One straightforward example is hyper-parameter optimization. For example, training an SVM involves in the fine-tuning of the penalty parameter C, different value of C will decide different model weights.

1.2 Overview of the bi-level optimization literature

Early works: Early on, some hard problem that are non-convex and non-differentiability have been tried to be solved using bi-level optimization [6] and [7]. Indeed, it has been noted by the authors of [8] and [9] that even if the inner problem and the outer problem are both convex, the derived bi-level optimization problem can be non-convex. Furthermore, [10] [11] have shown that checking the local optimality of bi-level optimization is actually an NP-hard problem. Another difficulty is that bi-level optimization problem have several local minima [12], which make them even harder to analyze theoretically.

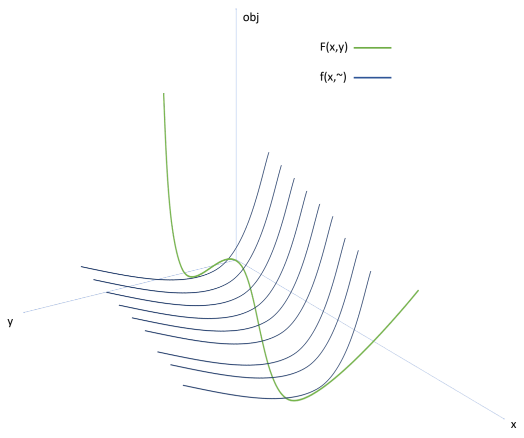

Figure 1: Illustration of bi-level optimization problem, obviously, it is not convex.

Overview of techniques: Hopefully, this has not discouraged researchers to study bi-level optimization problems, since a variety of techniques have been developed to tackle the aforementioned problems: [7], [13]. For instance, [14] use numerical methods to solve bilevel problems over polyhedra, and [15] uses steepest descent direction to solve non-linear bilevel programming problems. On still some other lines of work, [16] and [17] makes use the branch and bound technique, [18], [19] uses penalty function methods, and [20] as well as [21] uses trust-region methods (with a combined used of the penalty method and trust-region methods for [21]). Proximal gradient [22] can also be used to solve bilevel optimization problems. Finally, [23] and [24] and [25] use the Karush–Kuhn–Tucker conditions (KKT) to transform the two levels problem into a single level problem (under the right assumption), much easier to solve.

Generalizations of Bi-level optimization: There also has been some works generalizing bi-level optimization to multi-level optimization, also called hierarchical optimization, that is, the problem that arises when one faces a chain of deeper and deeper inner problems to optimize. Although this is out of the scope of this literature review, it is nonetheless worth noting. [26] and [27], for instance, provide a comprehensive review of those problems.

Automatic Differentiation: Researchers also have tried to develop more software-related techniques to solve bi-level optimization problem, building on the success of automatic differentiation to solve classical optimization problems. This is necessary in order to tackle bi-level optimization at a bigger scale, necessary for current machine learning algorithm working on large amounts of data that is often high-dimensional. For instance, [28] have developed an automatic differentiation framework that allows one to create a computational graph and define a function F that describes the optimal condition from the inner problem, and the framework will use automatic differentiation as well as implicit differentiation to allow to optimize directly the outer problem.

2 Bi-level optimization techniques

The mainstream gradient-based methods can be roughly divided into two categories: implicit gradient and explicit gradient. Implicit gradient [29] [30] utilizes the optimal condition of the inner problem, decoupling the calculation of hypergradient from the choice of inner problem optimizer. In contrast to implicit gradient method, explicit gradient method [31] [32] takes the inner problem as a iterative updating process, calculating the hypergradient as a nested updating term.

Unified Reformulation : Specifically, both of them requires to calculate the hypergradient of UL objective F with respect to UL variable x. However, it's hard to directly find out the required gradient as the UL variable x are inter-wined with LL variable y. To solve the above problem, we define the optimal solution of the LL problem as the best response of given UL variable (denoted as $y^*(x)$). The original problem now degenerates to a unified single-level problem:

$$\min_{x \in \mathcal{X}} \Phi(x) := F(x, y^*(x)) \tag{2}$$

Moving one step forward, the hypergradient of the equivalent objective $\Phi(x)$ w.r.t the UL variable x according to the chain rule:

$$\frac{\partial \Phi(x)}{\partial x} = \frac{\partial F(x, y^*(x))}{\partial x} + \left(\frac{\partial y^*(x)}{\partial x}\right)^T \frac{\partial F(x, y^*(x))}{\partial y} \tag{3}$$

where the first term $\frac{\partial F(x,y^*(x))}{\partial x}$ is the direct gradient, the second term $(\frac{\partial y^*(x)}{\partial x})^T \frac{\partial F(x,y^*(x)}{\partial y}$ known as implicit gradient is introduced due to the fact LL variable y is implicitly influenced by the UL variable x. Obviously, the direct gradient$\frac{\partial F(x,y^*(x))}{\partial x}$ and $\frac{\partial F(x,y^*(x))}{\partial y}$ in implicit gradient are easy to compute. The only challenging part is to find out gradient that the best response of LL variable w.r.t the given UL variable $\frac{y^*(x)}{\partial x}$ (best response matrix). Once we find out the hypergradient, we can manipulate using any classic gradient method to find out the standing point with feasible assumptions.

2.1 Implicit Gradient

Implicit gradient method utilizes implicit function theorem [33] to derive the best response matrix. With the assumption that inner-level objective is at least twice differentiable w.r.t x and y, we can directly calculate the implicit gradient based on the first order optimal condition, i.e. $\frac{\partial f(x,y^*(x))}{\partial y^*(x)} = 0$. We further derive the above equation w.r.t x, we have:

$$\frac{\partial^2 f(x, y^*(x))}{\partial y^*(x) \partial x} + \left(\frac{\partial y^*(x)}{\partial x}\right)^T \frac{\partial^2 f(x, y^*(x))}{\partial y^*(x)^2} = 0 \tag{4}$$

We further assume the Hessian matrix $\frac{\partial^2 f(x,y^*(x))}{\partial y^*(x)^2}$ is invertible, we can get the closed form representation of the best response matrix:

$$\left(\frac{\partial y^*(x)}{\partial x}\right)^T = -\frac{\partial^2 f(x, y^*(x))}{\partial y^*(x) \partial x} \left(\frac{\partial^2 f(x, y^*(x))}{\partial y^*(x)^2}\right)^{-1} \tag{5}$$

Implicit gradient method provides an efficient way to compute the whole gradient. However in practice, the computational burden is extremely heavy due to the involve of inverse of Hessian. The time complexity is $O(m^3)$ to exactly invert a general Hessian matrix of $m \times m$, which is totally infeasible dealing with high dimensional data. For example, it's well known that model parameters in neural networks can easily reach millions. Calculation of its Hessian will reach 10^{18} times operation, which is extremely costy and expensive. To solve the above problems, some researchers introduces techniques such as approximation methods [1] and Neumann series [29].

Conjugate Gradient: One well-known method to address the computational issue of inverse of Hessian matrix is to use transfer the problem to solve a linear. system. Specifically, we are approaching to compute the product $A^{-1}B$, which equals to solve the linear system $Ax = B$ for x. HOAG [1] constructs the tolerance sequence and proves its convergence based on the above methods.

Neumann Series: Another approach is to use Neumann series to approximate the inverse of Hessian matrix, which can be expressed as the given form:

$$\left(\frac{\partial^2 f}{\partial y^2}\right)^{-1} = \lim_{i\to\infty} \sum_{j=0}^{i} \left(I - \frac{\partial^2 f}{\partial y^2}\right)^{j} \tag{6}$$

Where I is the identity matrix with suitable size. Neumann series provides a cheaper approximation to the inverse-Hessian-vector product by using efficient vector-Jocabian products. [29] uses the above method and performs remarkable ability to handle millions hyperparameters.

2.2 Explicit Gradient

In contrast to implicit gradient method, explicit gradient based method takes the LL problem as a dynamic system and the relation between x and y is represented as iterative update[31]. We first assume at $t = 0$, the response of LL variable $y_0 = \Psi_0(x)$. With T denoting the overall iterations, the whole update process can be expressed as:

$$y_t = \Psi_t(y_{t-1}, x) \quad t = 1, \cdots, T \tag{7}$$

We can further formulate the update rule as:

$$y_t = y_{t-1} - \alpha_t d_f(y_{t-1}, x) \tag{8}$$

where α_t is the step size, $d_f(y_{t-1}, x)$ is the descent mapping. Moving one step forward, we can replace the best response $y^*(x)$ with $y_T(x)$, the optimization problem now transfers to:

$$\min_{x\in\mathcal{X}} \Phi(x) := F(x, y_T(x)) \tag{9}$$

What we are pursing is to calculate the gradient $\frac{\partial \Phi(x)}{\partial x}$. Given there are many approaches according to update rule, we only elaborate forward-mode and reverse-mode method in this paper. Other approaches like initialization-based [34] [35] and proxy-based [36] will not be introduced here.

Forward mode: To compute the required gradient $\frac{\partial \Phi(x)}{\partial x}$, we use chain rule and try to find out the gradient $\frac{\partial y_T(x)}{\partial x}$. Specifically, every y_t is decided by y_{t-1}. Appealing to the chain rule:

$$\frac{\partial y_t}{\partial x} = \frac{\partial \Psi_t(y_{t-1}, x)}{\partial y_{t-1}} \frac{\partial y_{t-1}}{\partial x} + \frac{\partial \Psi_t(y_{t-1}, x)}{\partial x} \tag{10}$$

Defining $Z_T = \frac{\partial y_t}{\partial x}$, $A_t = \frac{\partial \Psi(y_{t-1},x)}{\partial y_{t-1}}$ and $B_t = \frac{\partial \Psi_t(y_{t-1},x)}{\partial x}$. The equation now is $Z_t = A_t Z_{t-1} + B_t$. After derivation, we can get the best response matrix as following formulation:

$$\frac{\partial y_T(x)}{\partial x} = Z_T = \sum_{t=0}^{T} (\prod_{i=t+1}^{T} A_i) B_t \tag{11}$$

Reverse-mode Reverse-mode method is an extension of back-propagation with the introduction of Lagrangian formulation. Combining Eq10 with Eq3, we get the following formulation:

$$\frac{\partial \Phi(x)}{\partial x} = \frac{\partial F(x, y_T)}{\partial x} + Z_T^{'} \frac{\partial F(x, y_T)}{\partial y_T} \tag{12}$$

where $Z_T^{'}$ is the transpose of Z_T to avoid notation confusion.We first define $g_T = \frac{\partial F(x,y_T)}{\partial x}$ and $\lambda_T = \frac{\partial F(x,y_T)}{\partial y_T}$. We apply the back propagation $g_{t-1} = g_t + B_t^{'} \lambda_t$ and $\lambda_{t-1} = A_t^{'} \lambda_t$, with $t = T, ..., 0$. Finally, we get $\frac{\partial \Phi_T(x)}{\partial x} = g_{-1}$.

3 Applications of Bi-level optimization

In this section, we will give a few examples of the applications of Bi-level optimizations such as meta-learning, hyper-parameter optimization and neural architecture search (NAS). Bi-level optimization gives a bird's-eye view of the formulation of these topic and thus proposes several possible solutions to the associated tasks.

3.1 Meta learning

Meta-learning, or learning to learn, is the science of systematically observing how different machine learning approaches perform on a wide range of learning tasks, and then learning from this experience, or meta-data, to learn new tasks much faster than otherwise possible. [2]

One of the most well-known application is few-shot classification. For each task, there exist a training dataset $\mathcal{D}_i = D^i_{tr} \cup D^i_{val}$ and the full meta training data set $\mathcal{D}$ is constituted by the full set of tasks $\{D_i\}, (i = 1, 2, \cdots, N)$. In this setting, meta learning tasks can be splitted into two learning problems: meta-feature learning and meta-initialization learning. Meta-feature learning (MFL) separates the learning process vertically by splitting network into two parts: meta feature extraction and task-specific part. But meta-initialization (MIL) takes another approach, similar to fine-tuning, it want to find a general initialization for all tasks (by learning from multi-task information) and optimize the whole neural network for each specific downstream task.

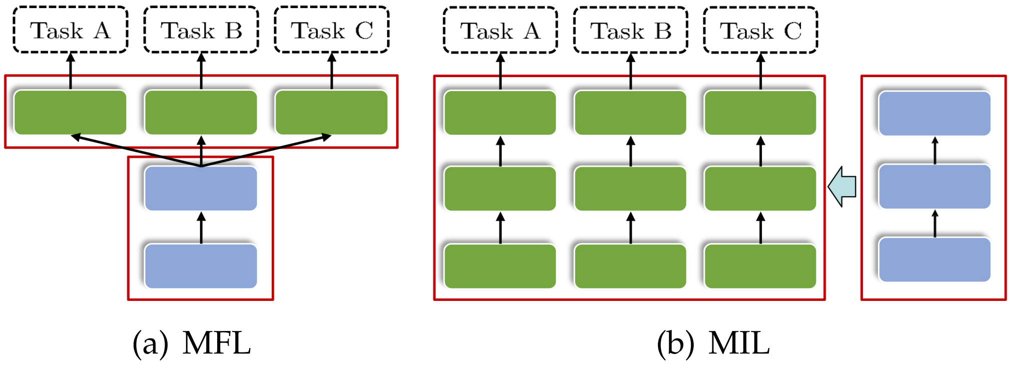

Figure 2: Meta-feature learning and meta-initialization

3.2 Hyper-parameter optimization

Hyper-parameter Optimization (HO) is about identifying the optimal set of hyper-parameters of a learning algorithm and it can't be learned from training data set directly. The naive way of optimizing hyper-parameter is to add constraint terms in objective functions (i.e. regularizer). It can be represented as a bi-level problem since it can be written as optimization on LL problem given hyper-parameters in the upper-level.

Actually, HO can be seen as the most straightforward application of bi-level optimzation. The upper level function $F(\mathbf{x}, \mathbf{y}, \mathcal{D}_{val})$ minimizes the loss on the hyper-parameters on validation data set and the lower- level function is the normal learning problem on training data set$f(\mathbf{x}, \mathbf{y}, \mathcal{D}_{train})$.

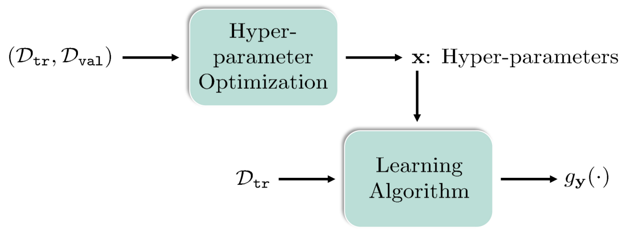
$(\mathcal{D}_{\mathtt{tr}}, \mathcal{D}_{\mathtt{val}})$
Hyper-parameter Optimization
$\mathbf{x}$: Hyper-parameters
$\mathcal{D}_{\mathtt{tr}}$
Learning Algorithm
$g_{\mathbf{y}}(\cdot)$

Figure 3: Hyper-parameter optimization as bi-level optimization

Recently, there are two main-stream methods of optimizing HO on deep neural networks. One is iterative differentiation and another is implicit differentiation. Iterative differentiation performs several steps of gradient descent and then calculates the hyper gradient by the corresponding implicit function. An application is data hyper-cleaning which chooses cross-entropy with a l_2 regularizer to learn the hyper-parameters.

3.3 Neural Architecture Search

Similar to hyper-parameter optimization, neural architecture search (NAS) is a method to automatically select the optimal set of hyper parameters of learning problems, specifically in here, the architecture of neural networks. The architecture of one neural network is determined by the type of cells, the number and the width of the layers of cells and the topology order of the calculation graph, etc. By composing these parameters into a set called search space, what NAS needs to do becomes searching for the optimal point in the search space. However, since building a neural network usually follows some particular basic strategies for human, NAS also includes the search of strategies, including building and estimating. It will reduces a lot of search space of the hyper parameters.

One of the most well-known example is DARTS [37], which relaxed the search space into continuous space to make it possible of applying gradient method in NAS. It makes each operation and architectural parameters as continuous real coefficient. $\mathbf{x}^{ij}$ is the connection between two nodes and the expression of the averaged mixed operation $\bar{o}_{ij}(\cdot)$ is based on the softmax function:

$$\bar{o}_{ij}(\cdot) = \sum_{o \in \mathcal{O}} \frac{\exp(\mathbf{x}_o^{ij})}{\sum_{o' \in \mathcal{O}} \exp(\mathbf{x}_{o'}^{ij})}, \tag{13}$$

where o and $o^{'}$ is the operation in candidates $\mathcal{O}$. By choosing the maximum operation by $o_{ij} = \arg\max_{o \in \mathcal{O}} \mathbf{x}_o^{ij}$, the best choice of architecture parameter is figured out. However, because of other architectural tricks like residual link, many improvements have been applied, like ENAS[38], P-DARTS[39], etc.

Based on bi-level optimization, several gradient-based differentiable NAS methods have reached promising ranks among several learning benchmarks, such as image classification [40], semantic segmentation [41] and recommendation system [42], etc. When appropriate search space is given, NAS method which based on bi-level optimization can be easily formulated and processed.

Below we provide a table gathering several results on neural architecture search:

Algorithm	**Accuracy %**	**Parameters (millions)**	**GPU days**
ENAS +micro [38]	3.54	4.6	0.5
ENAS + micro + Cutout [38]	3.54	5.6	0.5
ENAS + macro [38]	4.23	21.3	0.32
PDARTS + Cutout [43]	2.50	3.4	0.3
DARTS (1st order) + Cutout [37]	3.00	3.3	1.5
DARTS (2nd order) + Cutout [37]	2.76	3.3	4

The code for those algorithms is open-sourced at the following location: `https://github.com/melodyguan/enas` (ENAS), `https://github.com/chenxin061/pdarts` (PDARTS), and `https://github.com/quark0/darts` (DARTS).

4 Conclusion

In this survey, we first described the theoretical formulation of bi-level optimization, as well as its different versions, implicit or explicit, that allow to reformulate the two levels problems into a single level problem. After some further explanation about those two settings, we described the main applications of bi-level optimization, namely meta-learning, hyper-parameter optimization, and neural architecture search. Those are computationally extensive problems, particularly in today's paradigm where models take a lot of time to train, but they can be solved tractably thanks to bi-level optimization techniques.

Finally, regarding the future developments of bi-level optimization, it could be interesting to watch. In the years to come, whether bi-level optimization will become even more ubiquitous in machine learning, for instance with the advent of automatic differentiation techniques that also include more advanced implicit and explicit differentiation techniques.

References

[1] Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *International conference on machine learning*, pages 737–746. PMLR, 2016.

[2] Joaquin Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[4] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[5] Kaiyi Ji, Junjie Yang, and Yingbin Liang. Bilevel optimization: Nonasymptotic analysis and faster algorithms. *arXiv preprint arXiv:2010.07962*, 2020.

[6] Gautam Kunapuli, Kristin P Bennett, Jing Hu, and Jong-Shi Pang. Classification model selection via bilevel programming. *Optimization Methods & Software*, 23(4):475–489, 2008.

[7] Stephan Dempe and Susanne Franke. On the solution of convex bilevel optimization problems. *Computational Optimization and Applications*, 63(3):685–703, 2016.

[8] Pierre Hansen, Brigitte Jaumard, and Gilles Savard. New branch-and-bound rules for linear bilevel programming. *SIAM Journal on scientific and Statistical Computing*, 13(5):1194–1217, 1992.

[9] Alain B Zemkoho. Solving ill-posed bilevel programs. *Set-Valued and Variational Analysis*, 24(3):423–448, 2016.

[10] Luis Vicente, Gilles Savard, and Joaquim Júdice. Descent approaches for quadratic bilevel programming. *Journal of Optimization theory and applications*, 81(2):379–399, 1994.

[11] Herminia I Calvete and Carmen Galé. Algorithms for linear bilevel optimization. In *Bilevel Optimization*, pages 293–312. Springer, 2020.

[12] Risheng Liu, Pan Mu, Xiaoming Yuan, Shangzhi Zeng, and Jin Zhang. A generic first-order algorithmic framework for bi-level programming beyond lower-level singleton. In *International Conference on Machine Learning*, pages 6305–6315. PMLR, 2020.

[13] Anuraganand Sharma. Optimistic variants of single-objective bilevel optimization for evolutionary algorithms. *International Journal of Computational Intelligence and Applications*, 19(03):2050020, 2020.

[14] Herminia I Calvete, Carmen Galé, Stephan Dempe, and Sebastian Lohse. Bilevel problems over polyhedra with extreme point optimal solutions. *Journal of Global Optimization*, 53(3):573–586, 2012.

[15] Gilles Savard and Jacques Gauvin. The steepest descent direction for the nonlinear bilevel programming problem. *Operations Research Letters*, 15(5):265–272, 1994.

[16] Jie Lu, Chenggen Shi, Guangquan Zhang, and Da Ruan. An extended branch and bound algorithm for bilevel multi-follower decision making in a referential-uncooperative situation. *International Journal of Information Technology & Decision Making*, 6(02):371–388, 2007.

[17] José Fortuny-Amat and Bruce McCarl. A representation and economic interpretation of a two-level programming problem. *Journal of the operational Research Society*, 32(9):783–792, 1981.

[18] G Anandalingam and DJ White. A solution method for the linear static stackelberg problem using penalty functions. *IEEE Transactions on automatic control*, 35(10):1170–1173, 1990.

[19] Zhongping Wan, Lijun Mao, and Guangmin Wang. Estimation of distribution algorithm for a class of nonlinear bilevel programming problems. *Information Sciences*, 256:184–196, 2014.

[20] Stephan Dempe and Jonathan F Bard. Bundle trust-region algorithm for bilinear bilevel programming. *Journal of Optimization Theory and Applications*, 110(2):265–288, 2001.

[21] Bothina El-Sobky and Y Abo-Elnaga. A penalty method with trust-region mechanism for nonlinear bilevel optimization problem. *Journal of Computational and Applied Mathematics*, 340:360–374, 2018.

[22] Seifu Endris Yimer, Poom Kumam, and Anteneh Getachew Gebrie. Proximal gradient method for solving bilevel optimization problems. *Mathematical and Computational Applications*, 25(4):66, 2020.

[23] Gemayqzel Bouza Allende and Georg Still. Solving bilevel programs with the kkt-approach. *Mathematical programming*, 138(1):309–332, 2013.

[24] Ankur Sinha, Tharo Soun, and Kalyanmoy Deb. Using karush-kuhn-tucker proximity measure for solving bilevel optimization problems. *Swarm and evolutionary computation*, 44:496–510, 2019.

[25] Ankur Sinha, Samish Bedi, and Kalyanmoy Deb. Bilevel optimization based on kriging approximations of lower level optimal value function. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018.

[26] Kailash Lachhwani and Abhishek Dwivedi. Bi-level and multi-level programming problems: taxonomy of literature review and research issues. *Archives of Computational Methods in Engineering*, 25(4):847–877, 2018.

[27] G Anandalingam and Terry L Friesz. Hierarchical optimization: An introduction. *Annals of Operations Research*, 34(1):1–11, 1992.

[28] Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*, 2021.

[29] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation, 2019.

[30] Riccardo Grazzi, Luca Franceschi, Massimiliano Pontil, and Saverio Salzo. On the iteration complexity of hypergradient computation. In *International Conference on Machine Learning*, pages 3748–3758. PMLR, 2020.

[31] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, pages 1165–1173. PMLR, 2017.

[32] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1568–1577. PMLR, 2018.

[33] K Jittorntrum. An implicit function theorem. *Journal of Optimization Theory and Applications*, 25(4):575–577, 1978.

[34] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[35] Eunbyung Park and Junier B Oliva. Meta-curvature. *Advances in Neural Information Processing Systems*, 32, 2019.

[36] Juhan Bae and Roger B Grosse. Delta-stn: Efficient bilevel optimization for neural networks using structured response jacobians. *Advances in Neural Information Processing Systems*, 33:21725–21737, 2020.

[37] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2018.

[38] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

[39] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive darts: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision*, 129(3):638–655, 2021.

[40] Hongwei Dong, Bin Zou, Lamei Zhang, and Siyu Zhang. Automatic design of cnns via differentiable neural architecture search for polsar image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 58(9):6362–6375, 2020.

[41] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 82–92, 2019.

[42] Weiyu Cheng, Yanyan Shen, and Linpeng Huang. Differentiable neural input search for recommender systems. *arXiv preprint arXiv:2006.04466*, 2020.

[43] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1294–1303, 2019.

A Plans

A.1 Task division

- **Zhenhao** Investigate applications of bi-level optimization in different areas and write the corresponding part(s) of the survey.
- **William** Write introduction and conclusion of the survey. Plot diagrams and make forms of this paper.
- **Boyang** Summarize the general mathematical form of bi-level optimization. Introduce the main methods. Search and conclude related works.

A.2 Midterm milestone

The milestone should give a almost-finished reference paper list of this survey. Each member should give a relatively detailed draft of the part they are working on. The survey for milestone should contain at least three main parts: introduction, explanation of formulation (including algorithm) and examples of application.