Gradient-based Bi-level Optimization for Deep Learning: A Survey

Anonymous authors
Paper under double-blind review

Abstract

Bi-level optimization, especially the gradient-based category, has been widely used in the deep learning community including hyperparameter optimization and meta-knowledge extraction. Bi-level optimization embeds one problem within another and the gradient-based category solves the outer level task by computing the hypergradient, which is much more efficient than classical methods such as the evolutionary algorithm. In this survey we first give a formal definition of the gradient-based bi-level optimization. Secondly, we illustrate how to formulate a research problem as a bi-level optimization problem, which is of great practical use for beginners. More specifically, there are two formulations: the single-task formulation to optimize hyperparameters such as regularization parameters and the distilled data, and the multi-task formulation to extract meta knowledge such as the model initialization. With a bi-level formulation, we then discuss four bi-level optimization solvers to update the outer variable including explicit gradient update, proxy update, implicit function update, and closed-form update. Last but not least, we conclude the survey by pointing out the great potential of gradient-based bi-level optimization on science problems (AI4Science).

1 Introduction

With the fast development of deep learning, bi-level optimization is drawing lots of research attention due to the nested problem structure in many deep learning problems, including hyperparameter optimization Rendle (2012); Chen et al. (2019); Liu et al. (2019) and meta knowledge extraction Finn et al. (2017). The bi-level optimization problem is a special kind of optimization problem where one problem is embedded within another and can be traced to two domains: one is from game theory where the leader and the follower compete on quantity in the Stackelberg game Von Stackelberg (2010); another one is from mathematical programming where the inner level problem serves as a constraint on the outer level problem Bracken & McGill (1973). Especially, compared with classical methods Sinha et al. (2017) which require strict mathematical properties or can not scale to large datasets, the efficient gradient descent methods provide a promising solution to the complicated bi-level optimization problem and thus are widely adopted in many deep learning research work to optimize hyperparameters in the single-task formulation Bertinetto et al. (2019); Hu et al. (2019); Liu et al. (2019); Rendle (2012); Chen et al. (2019); Ma et al. (2020); Zhang et al. (2023); Li et al. (2022) or extract meta knowledge in the multi-task formulation Finn et al. (2017); Andrychowicz et al. (2016); Chen et al. (2023b); Zhong et al. (2022); Chi et al. (2021; 2022); Wu et al. (2022b); Chen et al. (2022d).

In this survey, we focus on gradient-based bi-level optimization on deep neural networks with an explicitly defined objective function. This survey aims to guide researchers on their research problems involving bi-level optimization. We first define notations and give a formal definition of gradient-based bi-level optimization in Section 2. We then propose a new taxonomy in terms of task formulation in Section 3 and ways to compute the hypergradient of the outer variable in Section 4. This taxonomy provides guidance to researchers on how to formulate a task as a bi-level optimization problem and how to solve this problem. Last, we conclude the survey with promising future directions on science problems (AI4Science) in Section 5.

Table 1: Key notations used in this paper.

Notations	Descriptions
$\boldsymbol{x}_i$	Input of data point indexed by i
$\boldsymbol{y}_i$	Label of data point indexed by i
$\mathcal{D}/\mathcal{D}^{train}/\mathcal{D}^{val}$	Supervised/Training/Validation dataset
$\|\mathcal{D}\|$	Number of samples in the dataset $\mathcal{D}$
$\boldsymbol{\theta}/\boldsymbol{\Theta}$	Inner learnable variable
$\boldsymbol{\phi}/\ \boldsymbol{\Phi}$	Outer learnable variable
$\boldsymbol{\theta}^*(\boldsymbol{\phi})$	Best response of $\boldsymbol{\theta}$ given $\boldsymbol{\phi}$
$l(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}_i, y_i)$	Loss on the i_{th} data point $\boldsymbol{x}_i, \boldsymbol{y}_i$
$l(\boldsymbol{\theta}, \boldsymbol{x}_i, y_i)$	Loss on data $\boldsymbol{x}_i, \boldsymbol{y}_i$ without $\boldsymbol{\phi}$
$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D})$	Loss on the whole dataset $\mathcal{D}$
$\mathcal{L}(\boldsymbol{\theta}, \mathcal{D})$	Loss on dataset $\mathcal{D}$ without $\boldsymbol{\phi}$
$\mathcal{L}^{in}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D})$	Inner level loss on dataset $\mathcal{D}$
$\mathcal{L}^{out}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D})$	Outer level loss on dataset $\mathcal{D}$
$\frac{d\mathcal{L}^{out}}{d\boldsymbol{\phi}}$	Hypergradient regarding $\boldsymbol{\phi}$
η	Inner level learning rate
M	Number of inner level tasks
$\Omega(\boldsymbol{\theta}, \boldsymbol{\phi})$	Regularization parameterized by $\boldsymbol{\phi}$
OPT	Some optimizer like Adam
$\mathcal{D}_{real}/\mathcal{D}_{syn}$	Real/Synthetic dataset
$p(\cdot)$	Product price in the market
$C_l(\phi)/C_f(\theta)$	Cost of leader and follower
D	Predicted atom-atom distances
$\mathcal{D}^{prf}/\mathcal{D}^{ft}$	Pretraining/finetuning data
E	Some equation constraint
λ/γ	Regularization strength parameter
ϵ/τ	Some small positive constant
$P_{\boldsymbol{\alpha}}$	Proxy network parameterized by $\boldsymbol{\alpha}$
T	Number of iterations in optimization

2 Definition

In this section, we define the gradient-based bi-level optimization, which focuses on neural networks with an explicit objective function. For convenience, we list some notations and their descriptions in Table 1.

Assume there is a dataset $\mathcal{D} = \{(\boldsymbol{x}_i,\ y_i)\}$ under supervised learning setting where $\boldsymbol{x}_i$ and y_i represent the i^{th} input and label, respectively. Besides, $\mathcal{D}^{train}$ and $\mathcal{D}^{val}$ represent the training set and the validation set, respectively. We use $\boldsymbol{\theta}$ ($\boldsymbol{\Theta}$ for matrix form) to parameterize the inner learnable variable which often refers to the model parameters and use $\boldsymbol{\phi}$ ($\boldsymbol{\Phi}$ for matrix form) to parameterize the outer learnable variable including the hyperparameters and the meta knowledge. In this paper, the hyperparameters are not limited to the regularization and the learning rate but refer to any knowledge in a single task formulation including network architecture, and distilled data samples, as we will illustrate more detailedly in Section 3. Denote the loss function on $(\boldsymbol{x}_i, y_i)$ as $l(\boldsymbol{\theta}, \boldsymbol{x}_i, y_i)$, which refers to a certain format of objectives depending on the tasks such as Cross-Entropy loss or Mean Square Error (MSE) loss. Note that in some cases we use $l(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}_i, y_i)$, which is an equivalent variant of $l(\boldsymbol{\theta}, \boldsymbol{x}_i, y_i)$ under this setting. This is because the outer learnable parameters $\boldsymbol{\phi}$ can be some hyperparameters like the learning rate when calculating $l(\boldsymbol{\theta}, \boldsymbol{x}_i, y_i)$ and is thus not explicitly represented. We then use $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D})$ or $\mathcal{L}(\boldsymbol{\theta}, \mathcal{D})$ to denote the loss over the dataset $\mathcal{D}$, and represent the inner level loss and the outer level loss as $\mathcal{L}^{in}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D})$ and $\mathcal{L}^{out}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D})$, respectively. We use η to represent the learning rate adopted by the inner level optimizer. With these notations, we can formulate the following bi-level optimization problem:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \mathcal{L}^{out}(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{\phi}, \mathcal{D}^{val}). \tag{1}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{in}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D}^{train}). \tag{2}$$

The inner level problem in Eq. (2) builds the relation between the $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. Here we use the $\arg\min$ form in Eq. (2) but note that the inner level task can be extended to some equation constraints as we will further illustrate in Section 3. In the nature of neural networks, one can use gradient descent to estimate $\boldsymbol{\theta}^*(\boldsymbol{\phi})$. Moreover, by leveraging the relation built in Eq. (2), the outer level computes the hypergradient $\frac{d\mathcal{L}^{out}}{d\boldsymbol{\phi}}$ to update the outer variable $\boldsymbol{\phi}$. This is called as *gradient-based bi-level optimization*. In other words, one can think of the outer variable $\boldsymbol{\phi}$ as the decision variable of the leader-level objective and the inner variable $\boldsymbol{\theta}$ as the decision variable of the follower-level objective in the Stackelberg game Von Stackelberg (2010).

When extended to the multi-task scenario to extract meta knowledge on M different tasks, the above formulation can be rewritten as:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \sum_{i=1}^{M} \mathcal{L}^{out}(\boldsymbol{\theta}_i^*(\boldsymbol{\phi}), \boldsymbol{\phi}, \mathcal{D}_i^{val}). \tag{3}$$

$$\text{s.t.} \quad \boldsymbol{\theta}_i^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_i^{in}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D}_i^{train}). \tag{4}$$

where $\boldsymbol{\phi}$ represents meta knowledge such as the model initialization across tasks.

3 Task Formulation

The bi-level optimization task formulation has two types: the single-task formulation and the multi-task formulation, as shown in Figure 1. The choice of formulation depends on the specific research problem, as we will illustrate in this section.

3.1 Single-task formulation

The single-task formulation applies bi-level optimization on a single task and aims to learn hyperparameters for the task. Note that in this paper, the meaning of hyperparameter is not limited to its traditional meaning like regularization but has a broader meaning, referring to all single-task knowledge including network architecture, distilled data, etc. Regarding $\boldsymbol{\theta}$, the inner objective and the outer objective may have the same math formula (e.g., $\mathcal{L}^{in}(\boldsymbol{x}) = \boldsymbol{\theta_1}\boldsymbol{x} + \boldsymbol{\theta_2}$ and $\mathcal{L}^{out}(\boldsymbol{x}) = \boldsymbol{\theta_3}\boldsymbol{x} + \boldsymbol{\theta_4}$), or different math formulas (e.g., $\mathcal{L}^{in}(\boldsymbol{x}) = \boldsymbol{\theta_1}\boldsymbol{x} + \boldsymbol{\theta_2}$ and $\mathcal{L}^{out}(\boldsymbol{x}) = sin(\boldsymbol{\theta_3}\boldsymbol{x}) + \boldsymbol{\theta_4}$). Another criterion is whether the hypergradient computation only relies on the built inner level connection: $\frac{d\mathcal{L}^{out}(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \mathcal{D}^{val})}{d\boldsymbol{\phi}}$, or not: $\frac{d\mathcal{L}^{out}(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{\phi}, \mathcal{D}^{val})}{d\boldsymbol{\phi}}$. To sum up, we consider the following four cases and discuss the corresponding examples for better illustration:

- 1). $\mathcal{L}^{in}$ and $\mathcal{L}^{out}$ share the same mathematical formula and the hypergradient only comes from the inner level connection $\boldsymbol{\theta}(\boldsymbol{\phi})$;
- 2). $\mathcal{L}^{in}$ and $\mathcal{L}^{out}$ share the same mathematical formula and the hypergradient comes from both the inner level connection $\boldsymbol{\theta}(\boldsymbol{\phi})$ and the outer level objective explicitly;
- 3). $\mathcal{L}^{in}$ and $\mathcal{L}^{out}$ have different mathematical formulas and the hypergradient only comes from the inner level connection $\boldsymbol{\theta}(\boldsymbol{\phi})$;
- 4). $\mathcal{L}^{in}$ and $\mathcal{L}^{out}$ have different mathematical formulas and the hypergradient comes from both the inner level connection $\boldsymbol{\theta}(\boldsymbol{\phi})$ and the outer level objective explicitly;

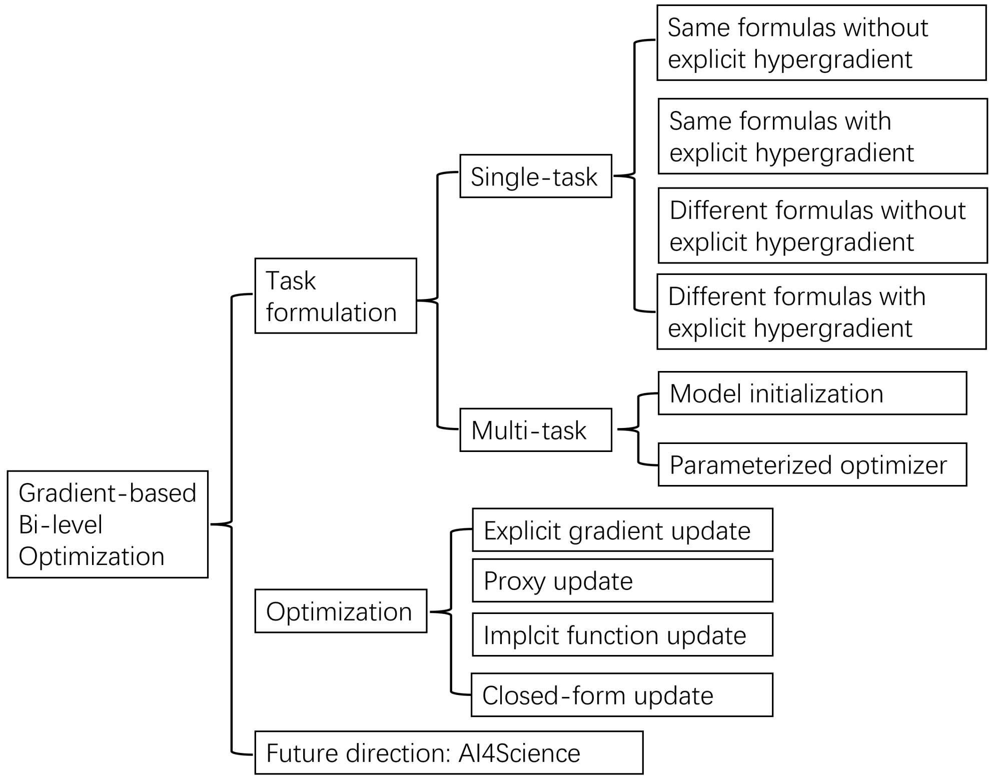

Figure 1: Summary of gradient-based bi-level optimization.

3.1.1 Same formula without explicit outer level hypergradient

When the inner level and the outer level adopt the same mathematical formula, the task aims to optimize a single objective, where the outer variable often describes some aspects of the training process besides the model parameters. Usually, treating the outer variable as constant in the outer level provides supervision signals to the update of the outer variable in the inner level task, which further improves the update of the inner variable. In this case, the hypergradient of $\frac{d\mathcal{L}^{out}}{d\boldsymbol{\phi}} = \frac{\partial \mathcal{L}^{out}}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}}^{\top}$. Based on the specific meanings of ϕ, existing outer variables can be categorized into two types: model-related outer variables and data-related outer variables.

Model-related. Model-related outer variables often describe the model optimization process, including regularization parameters Franceschi et al. (2018), learning rate Franceschi et al. (2017), etc, which are more common compared with data-related ones.

Regularization plays an important role to avoid overfitting, and finding a good regularization term is non-trivial, since evaluating a single regularization term requires training the whole model. The work in Franceschi et al. (2018) formulates choosing regularization as a bi-level optimization problem, where a proper regularization term results in a low error on the validation set. More specifically, this can be written as:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}^{val}} l(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{x}_i, \boldsymbol{y}_i). \tag{5}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}^{train}} l(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{x}_i, \boldsymbol{y}_i) + \Omega(\boldsymbol{\theta}, \boldsymbol{\phi}). \tag{6}$$

where $\Omega(\boldsymbol{\theta}, \boldsymbol{\phi})$ denotes the regularization term parameterized by $\boldsymbol{\phi}$ on $\boldsymbol{\theta}$. A simple case is L2 regularization where $\Omega(\boldsymbol{\theta}, \phi) = \phi \|\boldsymbol{\theta}\|^2$. It has been observed that learning a good regularization parameter is effective in recommender system Rendle (2012); Chen et al. (2019). The method in Rendle (2012) adopts dimension-

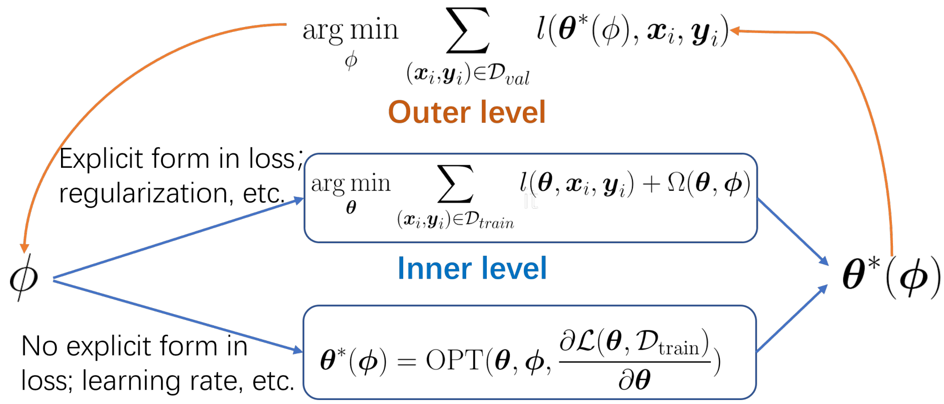

Figure 2: Model-related outer variables.

wise regularization for embeddings and the approach in Chen et al. (2019) further proposes fine-grained regularization. These algorithm variants further boost the model performance a lot.

Besides, the work Franceschi et al. (2017) treats the learning rate and the momentum as the outer variable $\boldsymbol{\phi}$ and updates the outer variable by minimizing the validation loss. This can also be formulated as a bi-level programming problem,

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \sum_{(\boldsymbol{x}_i,\boldsymbol{y}_i)\in\mathcal{D}_{val}} l(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{x}_i, \boldsymbol{y}_i). \tag{7}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\phi}) = \mathrm{OPT}(\boldsymbol{\theta}, \boldsymbol{\phi}, \frac{\partial \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_{\text{train}})}{\partial \boldsymbol{\theta}}). \tag{8}$$

where OPT represents some optimization process of minimizing the $\mathcal{L}_{train}$. A simple case is SGD with one step, which can be written as,

$$\boldsymbol{\theta}^*(\boldsymbol{\phi}) = \boldsymbol{\theta} - \phi \frac{\partial \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_{train})}{\partial \boldsymbol{\theta}^\top}. \tag{9}$$

where the outer variable ϕ refers to the learning rate η. This learning rate-learning procedure shares some similarities with the meta-learned optimization as we will illustrate in Sec 3.2.2. Compared with the regularization parameters explicitly existing in the loss objective, the learning rates only exist in the optimization process as shown in Figure 2.

Data-related. As shown in Figure 3, the data point $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ itself or weights can be viewed as the outer variable and updated via bi-level optimization in some research problems including adversarial attack, data distillation, label learning, etc.

Adversarial generalization attack aims to find data perturbations where the model is trained to perform poorly on the validation set, and this forms a traditional bi-level optimization problem Biggio et al. (2012); Yuan & Wu (2021),

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \sum_{(\boldsymbol{x}_j,\boldsymbol{y}_j)\in\mathcal{D}_{val}} -l(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{x}_j, \boldsymbol{y}_j). \tag{10}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \sum_{(\boldsymbol{x}_i,y_i)\in\mathcal{D}_{train}} l(\boldsymbol{\theta}, \boldsymbol{x}_i + \boldsymbol{\phi}_i, \boldsymbol{y}_i). \tag{11}$$

where $\boldsymbol{\phi}_i$ denotes the perturbation added on $\boldsymbol{x}_i$. The inner level builds the connection between the perturbation $\boldsymbol{\phi}_i$ and model parameters by minimizing the training loss and the outer level updates the perturbation $\boldsymbol{\phi}$ by maximizing the validation loss. In this way, we find the attacks $\boldsymbol{\phi}_i$.

Data distillation methods Wang et al. (2018); Lei & Tao (2023) try to distill the knowledge from a large training dataset into a small one $\boldsymbol{\phi}$, and these methods expect the model trained on the small one achieves good performance on the large one. The inner level builds the connection between the i_{th} data point $\boldsymbol{\phi}_i$ and the model parameters $\boldsymbol{\theta}$ by minimizing the training loss, and the outer level updates the synthesized $\boldsymbol{\phi}$ to obtain good performance on the real data. In this way, the model achieves good performance after gradient descent steps only on the small synthesized dataset. The above process can be formulated as a bi-level optimization problem:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \sum_{(\boldsymbol{x_j}, \boldsymbol{y_j}) \in \mathcal{D}_{real}} l(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{x_j}, \boldsymbol{y_j}). \tag{12}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \sum_{(\boldsymbol{\phi}_i, \boldsymbol{y_i}) \in \mathcal{D}_{syn}} l(\boldsymbol{\theta}, \boldsymbol{\phi}_i, \boldsymbol{y_i}). \tag{13}$$

where $\mathcal{D}_{real}$ represents the dataset containing the real data and $\mathcal{D}_{syn}$ represents the dataset containing the synthesized data. Furthermore, the work Nguyen et al. (2020) leverages the correspondence between infinitely-wide neural network and kernel and obtains impressive performances on data distillation. In their work, the label $\boldsymbol{y_i}$ can also be learned in a bi-level optimization framework similar to that of the input $\boldsymbol{x_i}$ for distillation. Chen et al. (2022b; 2023a) apply the data distillation idea to the black-box optimization and achieve impressive results.

Label learning methods Algan & Ulusoy (2021); Wu et al. (2021) treat the label $\boldsymbol{y_i}$ as the outer variable parameterized by $\boldsymbol{\phi}$ and focus on improving the model performance under label noise. These methods do not reduce the size of the dataset. Similarly, this can be formulated as a bi-level optimization problem:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \sum_{(\boldsymbol{x}_j, \boldsymbol{y}_j) \in \mathcal{D}_{val}} l(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{x}_j, \boldsymbol{y}_j). \tag{14}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \sum_{(\boldsymbol{x}_i, \boldsymbol{\phi}_i) \in \mathcal{D}_{train}} l(\boldsymbol{\theta}, \boldsymbol{x}_i, \boldsymbol{\phi}_i). \tag{15}$$

In this case, the validation data is clean, which guides the label learning of the noisy training data.

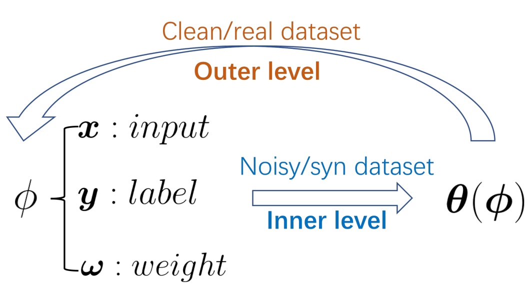

Figure 3: Data-related outer variables.

Besides $(\boldsymbol{x_i}, \boldsymbol{y_i})$ itself, some works Ren et al. (2018); Hu et al. (2019) propose to assign an instance weight ϕ_i to every $(\boldsymbol{x_i}, \boldsymbol{y_i})$ and these instance weights can also be treated as the outer variable. Instance weighting methods play an important role in solving the two major issues coexisting in real-world datasets: label noise and class imbalance. State-of-the-art methods assign each instance a weight ϕ_i and treat instance weights $\boldsymbol{\phi}$ as the outer variable. The inner level builds the connection between the model parameters $\boldsymbol{\theta}$ and the instance weights $\boldsymbol{\phi}$. The instance weight ϕ_i for each instance is treated as a constant one in the outer level since the validation set is clean and unbiased. Then $\boldsymbol{\phi}$ is updated by the validation set in the outer level through the connection built in the inner level. More specifically, the above process can be formulated as:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \sum_{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D}_{val}} l(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{x}_i, \boldsymbol{y}_i). \tag{16}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \sum_{(\boldsymbol{x_i}, \boldsymbol{y_i}) \in \mathcal{D}_{train}} \phi_i l(\boldsymbol{\theta}, \boldsymbol{x_i}, \boldsymbol{y_i}). \tag{17}$$

Many research works adopt this bi-level formulation to learn instance weights. The method in Ren et al. (2018) assigns weights to instances only based on their gradient directions. The approach in Hu et al. (2019) treats every instance weight as a learnable parameter and thus suffers from scalability for large datasets. Meanwhile, the works Shu et al. (2019); Wang et al. (2020) adopt a weighting network to output weights for instances and use bi-level optimization to jointly update $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ which can scale to large datasets.

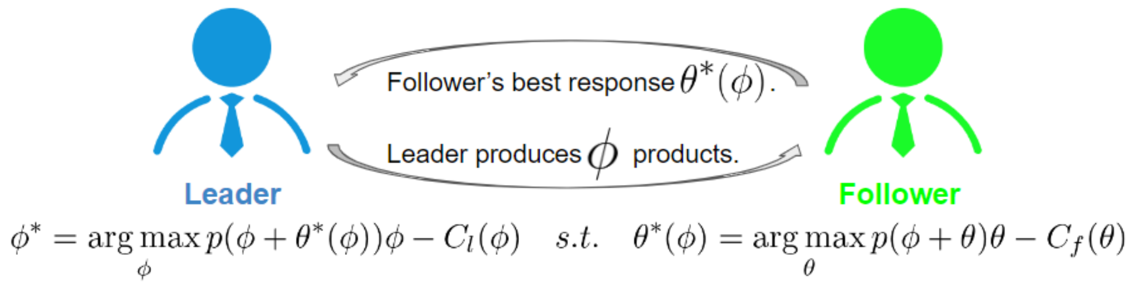

Figure 4: Stackberg game as bi-level optimization.

The approach Liu et al. (2021) proposes to build a more accurate relationship between $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ by implicit function. Last but not least, GDW Chen et al. (2021) finds that a scalar weight for every instance cannot capture class-level information, and introduces a class-level weight for every gradient flow in the chain rule to better use class-level information. Class-level weights are updated in the bi-level framework. Similar techniques Chen et al. (2022a)Wu et al. (2022a) are also used in recommender systems to model user-item pairs and triplets, respectively.

3.1.2 Same formula with explicit outer level hypergradient

When the inner level and the outer level share the same mathematical objective, there are cases where the hypergradient directly comes from the outer level. Compared with the previous case, these cases are not very common. We suppose this is because we can directly update the outer variable along with the inner variable by alternate optimization or joint optimization without turning to bi-level optimization in most cases.

We will first illustrate this case with the Stackberg game Von Stackelberg (2010). As shown in Figure 4, there are two companies, the leader company and the follower company, which compete on quantity. The leader company produces ϕ products and the follower company produces θ products. The overall price can be written as $p(\phi + \theta)$. The cost functions of the leader company and the follower company are $C_l(\phi)$ and $C_f(\theta)$, respectively. Thus, the profits of the leader company and the follower company can be written as $p(\phi+\theta)\phi - C_l(\phi)$ and $p(\phi+\theta)\theta - C_l(\theta)$. The Stackelberg game assumes the leader knows the best response of the follower and this can be formulated as a bi-level optimization problem:

$$\phi^* = \arg\min_{\phi} p(\phi + \theta^*(\phi))\phi - C_l(\phi). \tag{18}$$

$$\text{s.t.} \quad \theta^*(\phi) = \arg\min_{\phi} p(\phi + \theta)\theta - C_f(\theta). \tag{19}$$

Note that we can not directly optimize ϕ in the outer level without considering the inner level constraint, which necessitates the bi-level optimization formulation.

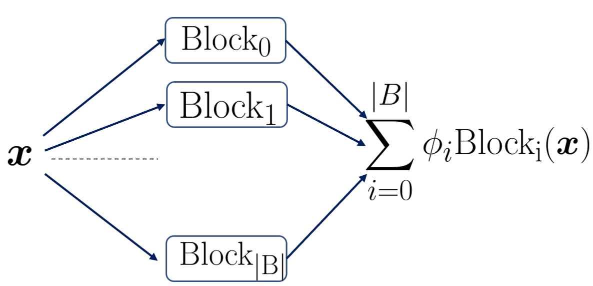

Figure 5: Continuous relaxation of the neural architecture in DARTS.

In the deep learning case, searching a neural network architecture in a defined search space can be formulated as a bi-level optimization problem. Generally speaking, the inner level yields some network architectures by some learned strategy, and the outer level optimizes the strategy by evaluating the performance of the searched network architectures. The traditional example DARTS Liu et al. (2019) proposes a continuous relaxation of the architecture representation parameterized by $\boldsymbol{\phi}$ as shown in Figure 5 and updates $\boldsymbol{\phi}$ to identify good neural network architectures. This is a bi-level optimization problem where the inner level builds the relationship between the model parameters $\boldsymbol{\theta}$ and network architecture parameterized by $\boldsymbol{\phi}$ by minimizing the training loss, and the outer level updates $\boldsymbol{\phi}$ by minimizing the validation loss through the inner level connection. The above process can be written as:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \sum_{(\boldsymbol{x}_i,\boldsymbol{y}_i)\in\mathcal{D}_{val}} l(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{\phi}, \boldsymbol{x}_i, \boldsymbol{y}_i). \tag{20}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \sum_{(\boldsymbol{x}_i,\boldsymbol{y}_i)\in\mathcal{D}_{train}} l(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}_i, \boldsymbol{y}_i). \tag{21}$$

There is no ground truth for neural architectures and thus the outer variable $\boldsymbol{\phi}$ can not be treated as constant in the previous instance weighting cases Shu et al. (2019); Chen et al. (2021). Note that the first-order DARTS treat $\boldsymbol{\theta}^*(\boldsymbol{\phi})$ as $\boldsymbol{\theta}^*$ independent of $\boldsymbol{\phi}$, and in this case, bi-level optimization reduces to alternate optimization. Accordingly, the first-order DARTS performs relatively worse than the original second-order DARTS. Besides the first-order variant, stochastic methods Giovannelli et al. (2021) are proposed to improve DARTS. Furthermore, network architecture search techniques are also used to search feature interaction modeling Lyu et al. (2021).

Besides network architecture search, dictionary learning methods Mairal et al. (2010) also belong to this category. These methods aim to find the sparse code $\boldsymbol{\Theta} \in \mathbb{R}^{n\times d}$ and the dictionary $\boldsymbol{\Phi} \in \mathbb{R}^{m\times n}$ to reconstruct the noise measurements $\boldsymbol{Y} \in \mathbb{R}^{m\times d}$. Note that d represents the dataset size, n represents the dictionary size and m represents the feature size of the dictionary feature. The loss function can be written as:

$$\arg\min_{\boldsymbol{\Theta},\boldsymbol{\Phi}} \mathcal{L}(\boldsymbol{\Theta}, \boldsymbol{\Phi}) = \frac{1}{2}\|\boldsymbol{\Phi}\boldsymbol{\Theta} - \boldsymbol{Y}\|^2 + \gamma\|\boldsymbol{\Theta}\|_1. \tag{22}$$

where γ is a regularization parameter. Alternate optimization over $\boldsymbol{\Theta}$ and $\boldsymbol{\Phi}$ is very slow since they neglect the explict relation between $\boldsymbol{\Theta}$ and $\boldsymbol{\Phi}$: given a dictionary $\boldsymbol{\Phi}$, the sparse code $\boldsymbol{\Theta}$ should be determined. Instead, this can be formulated in a bi-level manner:

$$\boldsymbol{\Phi}^* = \arg\min_{\boldsymbol{\Phi}} \mathcal{L}(\boldsymbol{\Theta}(\boldsymbol{\Phi}), \boldsymbol{\Phi}). \tag{23}$$

$$\text{s.t.} \quad \boldsymbol{\Theta}^*(\boldsymbol{\Phi}) = \arg\min_{\boldsymbol{\Theta}} \mathcal{L}(\boldsymbol{\Theta}, \boldsymbol{\Phi}). \tag{24}$$

which greatly accelerates the convergence rate. Recent work Zhang et al. (2022) adopts a similar bi-level formulation for model pruning where the sparse model is the inner variable, and the pruning mask is the outer variable.

3.1.3 Different formulas without explicit hypergradient

The inner level objective and the outer level objective usually have the same mathematical form where the validation set is used to evaluate the performance of the model parameters. Yet, in some cases, the inner level objective and the outer level objective can be different.

The work in Xu et al. (2021) proposes to decompose molecular conformation prediction into two levels. The inner level problem aims to construct the molecular conformation by leveraging the predicted atom distances and the outer level problem aims to align the molecular conformation with the ground truth conformations. Using $\mathcal{L}^{in}$ and $\mathcal{L}^{out}$ to denote the reconstruction loss and the alignment loss respectively, we have,

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \mathcal{L}^{out}(\boldsymbol{\theta}(\boldsymbol{\phi})). \tag{25}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{in}(\boldsymbol{\theta}, \boldsymbol{D}_{\boldsymbol{\phi}}). \tag{26}$$

where $\boldsymbol{D}_{\boldsymbol{\phi}}$ are the predicted atom distances parameterized by $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ is the predicted molecule conformation. This final output is a trained NN that could predict atoms' distances correctly.

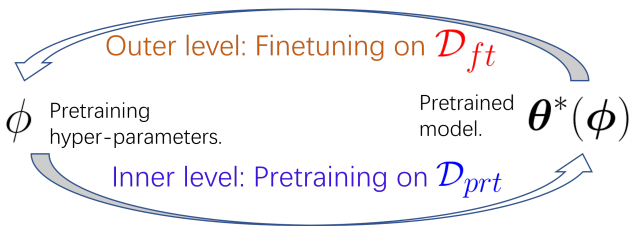

Figure 6: Pretraining and finetuning as bi-level optimization.

A similar formulation is in Zhang & Wonka (2021) which formulates 3d shape reconstruction as a bi-level optimization problem. This work Zhang & Wonka (2021) studies multi-task settings but the formulation process is very similar to Xu et al. (2021) so we discuss it here. At the inner level, the data generating network takes a single image as input and outputs the training data with coordinates and labels. Then the predictor is updated by leveraging the training data. In the outer level, the predictor expects to behave well on the validation set. The final output of this formulation is a good prediction network initialization.

Besides, the work Raghu et al. (2021) formulates the pretraining and finetuning process as a bi-level optimization problem where the outer variable is the pretraining hyperparameters. The inner level builds the connection between the model parameters and the hyperparameters by minimizing the pretraining loss. The outer level optimizes the pretraining hyperparameters to minimize the finetuning loss. The above process can be written as:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \mathcal{L}^{out}(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{\phi}, \mathcal{D}^{ft}). \tag{27}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{in}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D}^{prt}). \tag{28}$$

where $\mathcal{L}^{in}$, $\mathcal{L}^{out}$ represents the pretraining loss and finetuing loss respectively, and $\mathcal{D}^{prt}$, $\mathcal{D}^{ft}$ represents the pretraining data and the finetuing data respectively.

Furthermore, some work Christiansen et al. (2001); Zehnder et al. (2021), focusing on topology design, minimize the energy $\mathcal{L}^{in}(\boldsymbol{\theta}, \boldsymbol{\phi})$ in the inner level to achieve an equilibrium state $\boldsymbol{\theta}^*(\boldsymbol{\phi})$ and the outer level optimizes the cost function $\mathcal{L}^{out}(\boldsymbol{\theta}(\boldsymbol{\phi}), \boldsymbol{\phi})$ to obtain the desired topology $\boldsymbol{\phi}$. This process can be written as:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \mathcal{L}^{out}(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{\phi}). \tag{29}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{in}(\boldsymbol{\theta}, \boldsymbol{\phi}). \tag{30}$$

Note that similar formulations are also used to better simulate soft-body physics Rojas et al. (2021).

This formula can extend to implicit layers including Deep Equilibrium Models (DEQ) Bai et al. (2019) and Neural Ordinary Differential Equations (NeuralODE) Chen et al. (2018) where the inner level is not a minimization problem but an equation constraint. Specifically, the DEQ formulation can be written as,

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \mathcal{L}^{out}(\boldsymbol{\theta}(\boldsymbol{\phi}), \boldsymbol{x}, \boldsymbol{y}). \tag{31}$$

$$\text{s.t.} \quad \boldsymbol{\theta} = E(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}). \tag{32}$$

where the inner level implements an infinite-depth layer E to build the relationship between the equilibrium point $\boldsymbol{\theta}$ and the model parameters $\boldsymbol{\phi}$ and the outer level updates model parameters. The final output of DEQ is the trained model parameters $\boldsymbol{\phi}$. For NeuralODE, we have a similar formula,

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \mathcal{L}^{out}(\boldsymbol{\theta}(t, \boldsymbol{\phi}), \boldsymbol{x}, \boldsymbol{y}). \tag{33}$$

$$\text{s.t.} \quad \dot{\boldsymbol{\theta}(t)} = E(\boldsymbol{\theta}(t), \boldsymbol{\phi}, t) \quad \boldsymbol{\theta}(0) = \boldsymbol{\theta}_0. \tag{34}$$

where the inner level builds the relationship between the layer output $\boldsymbol{\theta}$ and the layer parameters $\boldsymbol{\phi}$ and the outer level updates the layer parameters. In both cases, the hyper-gradient regarding $\boldsymbol{\phi}$ is computed only through $\boldsymbol{\theta}(\boldsymbol{\phi})$.

3.1.4 Different formulas with explicit hypergradient

When the inner level objective and the outer level objective are different, there are cases where the explicit hypergradient exists, including surrogate loss function learning Grabocka et al. (2019), protein representation learning Chen et al. (2022c), adversarial training Jiang et al. (2021), etc. Take the surrogate loss learning as an example. Machine learning often adopts the proxies of misclassification rate(e.g. cross-entropy) to approximate the real losses due to the non-differential and non-continuous nature of these losses. To bridge the gap, the work Grabocka et al. (2019) proposes a surrogate neural network to approximate the true losses. To better learn the true loss with a dataset-specific distribution, a bi-level optimization formulation is proposed. The inner level optimizes model parameters $\boldsymbol{\theta}$ to minimize the surrogate loss $\mathcal{L}_{in}$ and the outer level updates the surrogate loss $\boldsymbol{\phi}$ to approximate the true loss $\mathcal{L}$. The above process can be written as:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \mathcal{L}^{out}(\mathcal{L}(\boldsymbol{\theta}, \mathcal{D}^{val})), \mathcal{L}^{in}(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{\phi}, \mathcal{D}^{val})). \tag{35}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{in}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D}^{train}). \tag{36}$$

where $\mathcal{L}^{out}$ measures the distance between the true loss $\mathcal{L}$ and the surrogate loss $\mathcal{L}^{in}$. In this way, any non-differentiable and non-decomposable loss function (e.g. misclassification rate) can be minimized virtually and effectively.

This bi-level formulation is also used in protein representation learning Chen et al. (2022c). In this task, the protein modeling neural network has two kinds of information: sequential representation parameterized by $\boldsymbol{\theta}$ and the structural representation parameterized by $\boldsymbol{\phi}$. The inner level identifies the relationship between the sequential information and the structural information by minimizing the negative of the mutual information loss as $\mathcal{L}^{out}$ and the outer level updates structural parameters by minimizing the pretraining loss as $\mathcal{L}^{in}$. This can be written as,

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \mathcal{L}^{in}(\boldsymbol{\theta}^*(\boldsymbol{\phi}), \boldsymbol{\phi}). \tag{37}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{out}(\boldsymbol{\theta}, \boldsymbol{\phi}). \tag{38}$$

This proposed pretraining scheme improves the performance of protein representation learning.

For adversarial training, the outer level may adopt a different loss function and thus adversarial training Jiang et al. (2021) also belongs to this category. In this case, the inner variable is the adversarial distribution and the outer variable is the classifier.

3.2 Multi-task formulation

The multi-task formulation aims to extract meta knowledge across tasks. There are mainly two kinds of meta knowledge represented by the outer variable: model initialization and optimizer.

3.2.1 Model initialization

The first kind of meta knowledge is model initialization, which is useful in the data-scarce scenario Hiller et al. (2022); Liu et al. (2020); Li et al. (2017). The work Finn et al. (2017) proposes MAML to train the parameters of the model across a family of tasks to generate a good model initialization. A good model initialization means a few steps on this initialization reach a good solution. Given a model initialization $\boldsymbol{\phi}$, the model parameters fine-tuned on the task i after a gradient descent step can be written as:

$$\boldsymbol{\theta}_i(\boldsymbol{\phi}) = \boldsymbol{\phi} - \eta \frac{\partial \mathcal{L}_i^{in}(\boldsymbol{\phi}, \mathcal{D}_i^{train})}{\partial \boldsymbol{\phi}^\top}. \tag{39}$$

The updated model parameters are expected to perform well on the validation set:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \sum_{i=1}^{M} \mathcal{L}^{out}(\boldsymbol{\theta}_i(\boldsymbol{\phi}), \mathcal{D}_i^{val}). \tag{40}$$

$$\text{s.t.} \quad \boldsymbol{\theta}_i(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_i^{in}(\boldsymbol{\phi}, \mathcal{D}_i^{train}). \tag{41}$$

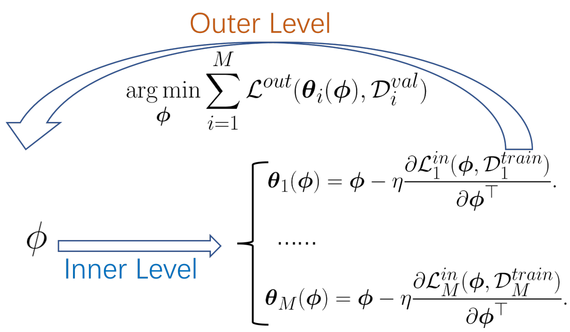

Figure 7: Illustration of MAML.

This forms a bi-level optimization problem as shown in Figure 7. MAML is applicable to a wide range of learning tasks including classification, regression, etc.

A single initialization may not be able to generalize to all tasks and thus some research work propose to learn different initialization for different tasks. The method Vuorio et al. (2019) modulates the initialization according to the task mode and adapts quickly by gradient updates. The approach in Yao et al. (2019) clusters tasks hierarchically and adapts the initialization according to the cluster. According to the task formulation, maybe only part of the model parameters need to be updated, which can save much memory considering millions of parameters for the model. The work in Lee et al. (2019) fixes the user embedding and item embedding and only updates the interaction parameters in the meta-learned phase. The method Rusu et al. (2018) proposes to map the high dimensional parameter space to a low-dimensional latent where they can perform MAML. Besides, some analyses Zou et al. (2021) are also proposed for choosing the inner loop learning rate. Last but not least, domain knowledge such as biological prior Yao et al. (2021) can also be incorporated into the MAML modeling where they propose a region localization network to customize the initialization to each assay. Tack et al. (2022) adapt the temporal ensemble of the meta-learner to generate the target model. Hiller et al. (2022) develop a novel method to increase adaptation speed inspired by preconditioning. Guan et al. (2022) analyze modern meta-learning algorithms and give a detailed analysis of stability and generalization.

3.2.2 Optimizer

Another kind of meta knowledge across tasks is an optimizer. Previous optimizers such as Adam are designed by hand and may be sub-optimal. The work in Andrychowicz et al. (2016) proposes to learn an optimizer for a family of tasks. As shown in Figure 8, the learned optimizer is parameterized by $\boldsymbol{\phi}$ where an LSTM takes the state as input and outputs the update. In this way, for the i_{th} task, the model parameters $\boldsymbol{\theta}$ are connected with the optimizer parameters via:

$$\boldsymbol{\theta}_{t+1}^i(\boldsymbol{\phi}) = \boldsymbol{\theta}_t^i + g_t^i(\boldsymbol{\phi}). \tag{42}$$

$$[g_t^i(\boldsymbol{\phi}), h_{t+1}^i] = \mathrm{OPT}(\nabla_{\boldsymbol{\theta}_t^i} \mathrm{l}_t, \mathrm{h}_t^i, \boldsymbol{\phi}). \tag{43}$$

where OPT denotes the learned LSTM optimizer and h represents the hidden state. The optimizer is updated to improve the validation performance over a horizon T, and this can be written as:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \sum_{t=1}^{T} \mathcal{L}_{out}(\boldsymbol{\theta}_t^i(\boldsymbol{\phi}), \mathcal{D}_{val}^i). \tag{44}$$

Overall, this can be formulated as a bi-level optimization problem where the inner level builds the connection between the model parameters $\boldsymbol{\theta}$ and the optimizer parameters $\boldsymbol{\phi}$ by minimizing the training loss, and the outer level updates the optimizer by minimizing the validation loss. Formally, the formulation of bi-level optimization across a family of M tasks can be written as:

$$\phi^* = \arg\min_{\phi} \sum_{i=1}^{M} \sum_{t=1}^{\top} \mathcal{L}_{out}(\theta_t^i(\phi), \mathcal{D}_{val}^i). \tag{45}$$

$$\text{s.t.} \quad \theta_{t+1}^i(\phi) = \theta_t^i + g_t^i(\phi); [g_t^i(\phi), h_{t+1}^i] = \mathrm{OPT}(\triangledown_{\theta_t^i} l_t, h_t^i, \phi). \tag{46}$$

To optimize millions of parameters, the LSTM is designed to be coordinatewise, which means every parameter shares the same LSTM. This greatly alleviates the computational burden. Besides, some preprocessing and postprocessing techniques are proposed to rescale the inputs and the outputs of LSTM into a normal range. One key challenge of learning to optimize is the generalization to longer horizons or unseen optimizees and many research works try to mitigate this challenge. The work in Metz et al. (2019) proposes to use an MLP layer instead of an LSTM to parameterize the optimizer and smooth the loss scope by dynamic gradient reweighting. The approach in Wichrowska et al. (2017) proposes a hierarchical RNN formed by three RNN layers that could communicate from bottom to up and this hierarchical design achieves better generalization. The work Lv et al. (2017) proposes training tricks such as random scaling to improve the generalization. Besides the above, some work Knyazev et al. (2021)Kang et al. (2021) proposes to directly predict parameters, which can be seen as a specially learned optimizer without any gradient update.

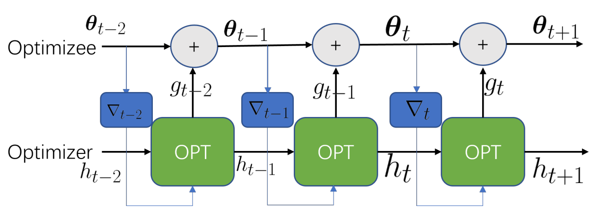

Figure 8: LSTM optimizer.

Recent works also try to incorporate the existing optimizer into the optimizer learning, which can leverage both the existing prior and the learning capacity. The key is to replace the constant (i.e. scalar/vector/matrix) in the existing optimizer with learnable parameters. HyperAdam Wang et al. (2019) learns the combination weights and decay rates according to the task. The method in Gregor & LeCun (2010) first writes the ISTA as a recurrent formula and then parameterizes the coefficients as the outer variable. The approach in Shu et al. (2020) designs a Meta-LR-Schedule-Net which takes the loss value and the state as input and outputs the learning rate for the current iteration. The work in Ravi & Larochelle (2016) proposes to parameterize the weight coefficient and the learning rate.

Besides the model initialization and the parameterized optimizer, there is some other meta knowledge like the loss function learning Gao et al. (2022). They propose to learn a generic loss function to train a robust DNN model that can perform well on out-of-distribution tasks. Given a parametric loss function as the outer variable ϕ, the inner level yields the optimized model parameters $\theta(\phi)$ by minimizing the training loss on the source domain. Then a good parametric loss can be identified by minimizing the validation loss on the target domain. The above process formulates a bi-level optimization problem, which is given by

$$\phi^* = \arg\min_{\phi} \sum_{i=1}^{N} \mathcal{L}^{out}(\theta^*(\phi), \mathcal{D}_i^{val}). \tag{47}$$

$$\text{s.t.} \quad \theta^*(\phi) = \arg\min_{\theta} \sum_{i=1}^{M} \mathcal{L}^{in}(\theta, \phi, \mathcal{D}_i^{train}). \tag{48}$$

where $\mathcal{L}^{out}$ is a loss function to measure the performance on target domains and N, and M represent the number of target domain and source domain tasks, respectively. They further propose to compute the hyper-gradient by leveraging the implicit function theorem.

4 Optimization

Gradient-based bi-level optimization requires the hypergradient computation of $\frac{d\mathcal{L}^{out}}{d\boldsymbol{\phi}}$ in the outer level. The hypergradient $\frac{d\mathcal{L}^{out}}{d\boldsymbol{\phi}}$ can be unrolled via the chain rule as:

$$\frac{d\mathcal{L}^{out}}{d\boldsymbol{\phi}} = \frac{\partial\mathcal{L}^{out}}{\partial\boldsymbol{\theta}}\frac{\partial\boldsymbol{\theta}(\boldsymbol{\phi})}{\partial\boldsymbol{\phi}} + \frac{\partial\mathcal{L}^{out}}{\partial\boldsymbol{\phi}}. \tag{49}$$

$\frac{\partial\boldsymbol{\theta}(\boldsymbol{\phi})}{\partial\boldsymbol{\phi}}$ often involves second-order gradient computation and thus are resource demanding. There are generally four types of methods to calculate $\frac{\partial\boldsymbol{\theta}(\boldsymbol{\phi})}{\partial\boldsymbol{\phi}}$: explicit gradient update, explicit proxy update, implicit function update, and closed-form method, where the previous three are approximation methods for general functions with the difference in how to build the connection between $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ and the last one is an accurate method for certain functions.

4.1 Explicit gradient update

The explicit gradient update is the most straight-forward one which approximates $\boldsymbol{\theta}$ via some optimizer directly:

$$\boldsymbol{\theta}_t = \mathrm{OPT}(\boldsymbol{\theta}_{\mathrm{t}-1}, \boldsymbol{\phi}), \quad \mathrm{t} = 1, \cdots, \mathrm{T}. \tag{50}$$

where T denotes the number of iterations, $\boldsymbol{\theta}$ represents the model parameters and other optimization variables like momentum, OPT represents the optimization algorithm like SGD, and ϕ denotes the outer variable in the training process. Note that when OPT is the SGD optimizer and only one gradient descent step is considered, Eq. (50) becomes

$$\boldsymbol{\theta}(\boldsymbol{\phi}) = \boldsymbol{\theta} - \eta\frac{\partial\mathcal{L}^{in}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D}^{train})}{\partial\boldsymbol{\theta}^\top}. \tag{51}$$

In this case, we can compute the hypergradient as:

$$\frac{\partial\boldsymbol{\theta}(\boldsymbol{\phi})}{\partial\boldsymbol{\phi}} = -\eta\frac{\partial^2\mathcal{L}^{in}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D}^{train})}{\partial\boldsymbol{\theta}^\top\partial\boldsymbol{\phi}}. \tag{52}$$

which often requires the second-order gradient computation. In some cases, the first-order approximation can be adopted to replace the second-order gradient in Liu et al. (2019); Finn et al. (2017); Nichol et al. (2018). Besides, Liu et al. (2019) uses the finite difference approximation technique to compute the second-order gradient efficiently.

$$\frac{\partial^2\mathcal{L}^{in}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D}^{train})}{\partial\boldsymbol{\theta}^\top\partial\boldsymbol{\phi}}\frac{\partial\mathcal{L}^{out}}{\partial\boldsymbol{\theta}} \approx \frac{\frac{\partial\mathcal{L}^{in}(\boldsymbol{\theta}^+, \boldsymbol{\phi}, \mathcal{D}^{train})}{\partial\phi} - \frac{\partial\mathcal{L}^{in}(\boldsymbol{\theta}^-, \boldsymbol{\phi}, \mathcal{D}^{train})}{\partial\boldsymbol{\phi}}}{2\epsilon}. \tag{53}$$

where $\boldsymbol{\theta}^\pm = \boldsymbol{\theta} \pm \epsilon\frac{\partial\mathcal{L}^{out}(\boldsymbol{\theta}, \boldsymbol{\phi})}{\partial\boldsymbol{\theta}}$. This avoids the expensive computational cost of the Hessian matrix. Furthermore, the work Deleu et al. (2022) proposes to adopt infinitely small gradient steps to solve the inner level task, which leads to a continuous-time bi-level optimization solver:

$$\frac{d\boldsymbol{\theta}(t)}{dt} = \frac{\partial\mathcal{L}^{in}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D}^{train})}{\partial\boldsymbol{\theta}^\top}. \tag{54}$$

In this way, the final output is the solution of an ODE. One great advantage of this formulation is making the fixed and discrete number of gradient steps the length of the trajectory, which serves as a continuous variable and is also learnable. This work also proposes to use forward mode differentiation to compute the hypergradient where the memory does not scale with the length of the trajectory. A similar continuous bi-level solver is used in Yuan & Wu (2021).

Generally speaking, the update is not limited to one step nor SGD optimizer, which makes the hypergradient computation process complicated. There are generally two modesFranceschi et al. (2017) to compute the hypergradient: forward mode and reverse mode.

Forward mode. Forward mode methods apply the chain rule to the composite functions:

$$\frac{d\boldsymbol{\theta}_t}{d\boldsymbol{\phi}} = \frac{\partial \mathrm{OPT}(\boldsymbol{\theta}_{\mathrm{t}-1}, \boldsymbol{\phi})}{\partial \boldsymbol{\theta}_{t-1}} \frac{d\boldsymbol{\theta}_{t-1}}{d\boldsymbol{\phi}} + \frac{\partial \mathrm{OPT}(\boldsymbol{\theta}_{\mathrm{t}-1}, \boldsymbol{\phi})}{\partial \boldsymbol{\phi}}. \tag{55}$$

Then the matrics are defined as:

$$\boldsymbol{Z}_t = \frac{d\boldsymbol{\theta}_t}{d\boldsymbol{\phi}}, \quad \boldsymbol{A}_t = \frac{\partial \mathrm{OPT}(\boldsymbol{\theta}_{\mathrm{t}}, \boldsymbol{\phi})}{\partial \boldsymbol{\theta}_t}, \quad \boldsymbol{B}_t = \frac{\partial \mathrm{OPT}(\boldsymbol{\theta}_{\mathrm{t}}, \boldsymbol{\phi})}{\partial \boldsymbol{\phi}_t}. \tag{56}$$

Thus the Eq.(55) can be written as:

$$\boldsymbol{Z}_t = \boldsymbol{A}_t \boldsymbol{Z}_{t-1} + \boldsymbol{B}_{t-1}. \tag{57}$$

In this way, $\boldsymbol{Z}_T$ can be written as:

$$\boldsymbol{Z}_T = \boldsymbol{A}_T \boldsymbol{Z}_{T-1} + \boldsymbol{B}_{T-1} \tag{58}$$

$$= \sum_{t=1}^{\top} \left(\prod_{s=t+1}^{\top} \boldsymbol{A}_s \right) \boldsymbol{B}_t. \tag{59}$$

which yields the final hypergradient.

Reverse mode. The reverse mode is derived from Lagrangian optimization. The Lagrangian of bi-level problems can be written as:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \alpha) = \mathcal{L}^{out}(\boldsymbol{\theta}_T) + \sum_{t=1}^{\top} \gamma_t (\mathrm{OPT}(\boldsymbol{\theta}_{\mathrm{t}-1}, \boldsymbol{\phi}) - \boldsymbol{\theta}_{\mathrm{t}}). \tag{60}$$

By setting the partial derivates as zeros, we can have,

$$\boldsymbol{\gamma}_t = \boldsymbol{\gamma}_{t+1} \boldsymbol{A}_{t+1}. \tag{61}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\gamma}} = \sum_{t=1}^{T} \alpha_t \boldsymbol{B}_t. \tag{62}$$

We can see this reverse mode yields the same solution as the forward mode. The works in Shaban et al. (2019); Luketina et al. (2016) propose to ignore the long-term dependencies for efficiency.

4.2 Explicit proxy update

Besides using explicit gradient update to solve the inner level task, a more direct way MacKay et al. (2019); Bae & Grosse (2020); Lorraine & Duvenaud (2018) is to fit a proxy network $P_\alpha(\cdot)$ which takes the outer variable as input and outputs the inner variable,

$$\boldsymbol{\theta}^* = P_\alpha(\boldsymbol{\phi}). \tag{63}$$

There are two ways to train the proxy: global and local. The global way aims to learn a proxy for all $\boldsymbol{\phi}$ by minimizing $\mathcal{L}^{in}(P_\alpha(\boldsymbol{\phi}), \boldsymbol{\phi}, \mathcal{D}^{train})$ for all $\boldsymbol{\phi}$ against $\boldsymbol{\alpha}$ while the local way minimizes $\mathcal{L}^{in}(P_\alpha(\boldsymbol{\phi}), \boldsymbol{\phi}, \mathcal{D}^{train})$ against $\boldsymbol{\alpha}$ for a neighborhood of $\boldsymbol{\phi}$.

A special case Bohdal et al. (2021) is to design the proxy as a weighted average of the perturbed inner variable. This work adopts an evolutionary algorithm to obtain an approximate solution for $\boldsymbol{\theta}$. By perturbing $\boldsymbol{\theta}$ to $\boldsymbol{\theta}_k$ for K times, they compute the training losses as $\{l_k(\boldsymbol{\phi})\}_{k=1}^K$ where $l_k(\boldsymbol{\phi}) = \mathcal{L}^{in}(\boldsymbol{\theta}_k, \boldsymbol{\phi}, \mathcal{D}^{train})$. Then the weights for each perturbed loss are,

$$\omega_1, \omega_2, \cdots, \omega_K = \mathrm{softmax}([-\mathrm{l}_1(\boldsymbol{\phi}), -\mathrm{l}_2(\boldsymbol{\phi}), \cdots, -\mathrm{l}_\mathrm{K}(\boldsymbol{\phi})]/\tau). \tag{64}$$

where $\tau > 0$ is a hyperparameter. Last, the proxy is obtained as

$$\boldsymbol{\theta}^* = \omega_1 \boldsymbol{\theta}_1 + \omega_2 \boldsymbol{\theta}_1 + \cdots + \omega_K \boldsymbol{\theta}_K. \tag{65}$$

Compared with explicit gradient update methods, these proxy methods can adopt deep learning modules to directly build the relationship between the inner variable and the outer variable. Thus these methods generally require less memory while are less accurate due to the rough approximation brought by the deep learning module.

4.3 Implicit function update

The hypergradient computation of the explicit gradient update methods relies on the path taken at the inner level, while the implicit function update makes use of the implicit function theorem to derive a more accurate hypergradient without vanishing gradients or memory constraints issues. First, the derivate of the inner level is zero:

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})}{\partial \boldsymbol{\theta}^{\top}} = \mathbf{0}. \tag{66}$$

Then according to the implicit function theorem, we have:

$$\frac{\partial^2 \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})}{\partial \boldsymbol{\theta}^{\top} \partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}} + \frac{\partial \mathcal{L}^2(\boldsymbol{\theta}, \boldsymbol{\phi})}{\partial \boldsymbol{\theta}^{\top} \partial \boldsymbol{\phi}} = \mathbf{0}. \tag{67}$$

At last, we can compute the hypergradient as:

$$\frac{\partial \boldsymbol{\theta}(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}} = -(\frac{\partial^2 \mathcal{L}(\boldsymbol{\theta}(\boldsymbol{\phi}), \boldsymbol{\phi})}{\partial \boldsymbol{\theta}^{\top} \partial \boldsymbol{\theta}})^{-1} \frac{\partial \mathcal{L}^2(\boldsymbol{\theta}, \boldsymbol{\phi})}{\partial \boldsymbol{\theta}^{\top} \partial \boldsymbol{\phi}}. \tag{68}$$

Take iMAMLRajeswaran et al. (2019) as an example. To keep the dependency between the model parameters $\boldsymbol{\theta}$ and the meta parameters $\boldsymbol{\phi}$, iMAML proposes a constraint on the inner level task, which can be written as:

$$\boldsymbol{\theta}_i(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_i^{in}(\boldsymbol{\theta}, \mathcal{D}_i^{train}) + \frac{\lambda}{2}\|\boldsymbol{\phi} - \boldsymbol{\theta}\|^2. \tag{69}$$

The regularization strength λ controls the strength of the prior $\boldsymbol{\phi}$ relative to the dataset. The iMAML bi-level optimization task can be formulated as:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \sum_{i=1}^{M} \mathcal{L}^{out}(\boldsymbol{\theta}_i(\boldsymbol{\phi}), \mathcal{D}_i^{val}). \tag{70}$$

$$\text{s.t.} \quad \boldsymbol{\theta}_i(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_i^{in}(\boldsymbol{\theta}, \mathcal{D}_i^{train}) + \frac{\lambda}{2}\|\boldsymbol{\phi} - \boldsymbol{\theta}\|^2, i = 1, \cdots, M. \tag{71}$$

The hyper-gradient can be computed as:

$$\frac{d\boldsymbol{\theta}_i(\boldsymbol{\phi})}{d\boldsymbol{\phi}} = (\mathbf{I} + \frac{1}{\lambda} \frac{\partial^2 \mathcal{L}^{in}(\boldsymbol{\theta}_i, \mathcal{D}^{train})}{\partial \boldsymbol{\theta}_i^{\top} \partial \boldsymbol{\theta}_i})^{-1}. \tag{72}$$

which is independent of the inner level optimization path. In this case, the hypergradient can be computed as the solution $\boldsymbol{g}$ to a linear system $\frac{\partial^2 \mathcal{L}^{in}(\boldsymbol{\theta}, \mathcal{D}^{train})}{\partial \boldsymbol{\theta}^{\top} \partial \boldsymbol{\theta}} \boldsymbol{g} = \frac{\partial \mathcal{L}^{out}}{\partial \boldsymbol{\theta}}$. More specifically, $\boldsymbol{g}$ can be seen as the approximate solution to the following optimization problem:

$$\arg\min_{\boldsymbol{\omega}} \boldsymbol{\omega}^{\top}(\boldsymbol{I} + \frac{1}{\lambda} \frac{\partial^2 \mathcal{L}^{in}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^{\top} \partial \boldsymbol{\theta}})\boldsymbol{\omega} - \boldsymbol{\omega}^{\top} \frac{\partial \mathcal{L}^{out}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^{\top}}. \tag{73}$$

Conjugate gradient methods can be applied to solve this problem where only Hessian-vector products are computed and the hessian matrix is not explicitly formed. This efficient algorithm is also used in HOAG Pedregosa (2016) to compute the hypergradient. Besides the linear system way to compute hypergradient, the work in Lorraine et al. (2020) proposes to unroll the above term into the Neumann Series:

$$(\frac{\partial^2 \mathcal{L}^{in}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^{\top} \partial \boldsymbol{\theta}})^{-1} = \sum_{i=0}^{\inf} (\boldsymbol{I} - \frac{\partial^2 \mathcal{L}^{in}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^{\top} \partial \boldsymbol{\theta}})^i. \tag{74}$$

The first i_{th} steps' result are used to approximate the computation if $\boldsymbol{I} - \frac{\partial^2 \mathcal{L}(\boldsymbol{\theta}(\boldsymbol{\phi}), \boldsymbol{\phi})}{\partial \boldsymbol{\theta}^{\top} \partial \boldsymbol{\theta}}$ is contractive. This can avoid the expensive computation of the inverse Hessian.

4.4 Closed-form update

While the above three methods provide an approximate solution for general loss functions, we here consider deriving a closed-form connection between $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ from

$$\boldsymbol{\theta}(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{in}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D}^{train}). \tag{75}$$

which is only applicable for some special cases. Bertinetto et al. (2019) propose ridge regression as part of its internal model for closed-form solutions. Assume a linear predictor f parameterized by $\boldsymbol{\theta}$ is considered as the final layer of a CNN parameterized by $\boldsymbol{\phi}$. Asume the input $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ and the output $\boldsymbol{Y} \in \mathbb{R}^{n\times o}$ where n represents the number of data points and d, o represents the input dimension and the output dimension, respectively. Denote the CNN as $\boldsymbol{\phi}(\boldsymbol{X})$: $\mathbb{R}^d \to \mathbb{R}^e$ and then the predictor's output is $\boldsymbol{\phi}(\boldsymbol{X})\boldsymbol{\theta}$ where $\boldsymbol{\phi}(\boldsymbol{X}) \in \mathbb{R}^{n\times e}$ and $\boldsymbol{\theta} \in \mathbb{R}^{e\times o}$. The i_{th} inner level optimization task can be written as:

$$\boldsymbol{\theta}_i(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{\phi}(\boldsymbol{X}_i)\boldsymbol{\theta} - \boldsymbol{Y}_i\|^2 + \lambda\|\boldsymbol{\theta}\|^2. \tag{76}$$

where λ controls the strength of L^2 regularization. The closed form solution of $\boldsymbol{\theta}$ is

$$\boldsymbol{\theta}_i(\boldsymbol{\phi}) = (\boldsymbol{\phi}(\boldsymbol{X}_i)^\top \boldsymbol{\phi}(\boldsymbol{X}_i) + \lambda \mathbf{I})^{-1} \boldsymbol{\phi}(\boldsymbol{X}_i)^\top \boldsymbol{Y}_i. \tag{77}$$

The solution can also be written as:

$$\boldsymbol{\theta}_i(\boldsymbol{\phi}) = \boldsymbol{\phi}(\boldsymbol{X}_i)^\top (\boldsymbol{\phi}(\boldsymbol{X}_i)\boldsymbol{\phi}(\boldsymbol{X}_i)^\top + \lambda \mathbf{I})^{-1} \boldsymbol{Y}_i. \tag{78}$$

In this way, $\boldsymbol{\phi}(X_i)\boldsymbol{\phi}(X_i)^\top \in \mathbb{R}^{n\times n}$ saves much memory since n is small in the few-shot setting. To sum up, to extract meta knowledge $\boldsymbol{\phi}$ in the feature extractor, we have the bi-level optimization formulation as:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \sum_{i=1}^{M} \mathcal{L}^{out}(\boldsymbol{\theta}_i(\boldsymbol{\phi}), \mathcal{D}_i^{val}). \tag{79}$$

$$\text{s.t.} \quad \boldsymbol{\theta}_i(\boldsymbol{\phi}) = \boldsymbol{\phi}(\boldsymbol{X}_i)^\top (\boldsymbol{\phi}(\boldsymbol{X}_i)\boldsymbol{\phi}(\boldsymbol{X}_i)^\top + \lambda \mathbf{I})^{-1} \boldsymbol{Y}_i, i = 1, \cdots, M. \tag{80}$$

Applying Newton's method to logistic regression, yields a series of weighted least squares (or ridge regression) problems. This is also a closed-form solution but requires a few steps.

Another special case is to assume the model to be wide enough. Recent workJacot et al. (2018); Lee et al. (2017) build the correspondence between the NNGP kernel and the Bayesian Neural Network and the correspondence between the NTK kernel and the gradient trained Neural Network with MSE loss. In this case, the inner level can have a closed-form solution. The works Nguyen et al. (2020); Yuan & Wu (2021) treat the data as the outer variable for data distillation and adversarial attack tasks respectively, which yield better-distilled samples and adversarial attacks. These algorithms can be achieved by NTK tool Novak et al. (2019) easily. The approach Dukler et al. (2021) treats the instance weights as the outer variable and assumes the pretrained model with linear representation to yield a closed-form solution for the inner level task. Besides assuming the inner level as ridge regression and least squares, some works Ghadimi & Wang (2018); Yang et al. (2021) also assume the inner level loss function is strongly convex and propose effective algorithms to better solve the bi-level optimization problem.

5 Conclusion and Future Direction

Bi-level optimization embeds one problem within another and the gradient-based category solves the outer level task via gradient descent methods. We first discuss how to formulate a research problem from a bi-level optimization perspective. There are two formulations: the single-task formulation to optimize hyperparameters and the multi-task formulation to extract meta knowledge. Further, we discuss four possible ways to compute the hypergradient in the outer level, including explicit gradient update, proxy update, implicit function update, and closed-form update. This could serve as a good guide for researchers on applying gradient-based bi-level optimization.

We end our survey by pointing out the great potential of gradient-based bi-level optimization in the science area. This merging AI4Science direction is attracting lots of research attention recently, including topology design Christiansen et al. (2001); Zehnder et al. (2021), molecular conformation prediction Xu et al. (2021), differentiable simulation Rojas et al. (2021), protein representation learning Chen et al. (2022c), low-resource drug discovery Yao et al. (2021), etc. The reason why this bi-level formulation is popular in the science area is that many science problems have a nested problem structure and differentiable nature, and the deep learning methods endow these traditional problems with the expressive representation ability. We also note that the great success of bi-level optimization often needs careful design of the outer variable. Like in DARTS Liu et al. (2019), we may need to transform the outer variable to a continuous form to enable direct optimization. To sum up, the combination between gradient-based bi-level optimization and science problems deserves more research attention and we believe there will be many high-impact research on this area in the future.

References

Görkem Algan and Ilkay Ulusoy. Meta soft label generation for noisy labels. In *International Conference on Pattern Recognition*, 2021.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, 2016.

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

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 2019.

Luca Bertinetto, Joao F. Henriques, Philip Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*, 2019.

Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

Ondrej Bohdal, Yongxin Yang, and Timothy Hospedales. Evograd: Efficient gradient-based meta-learning and hyperparameter optimization. *Advances in Neural Information Processing Systems*, 2021.

Jerome Bracken and James T McGill. Mathematical programs with optimization problems in the constraints. *Operations Research*, 1973.

Can Chen, Shuhao Zheng, Xi Chen, Erqun Dong, Xue Steve Liu, Hao Liu, and Dejing Dou. Generalized dataweighting via class-level gradient manipulation. *Advances in Neural Information Processing Systems*, 2021.

Can Chen, Chen Ma, Xi Chen, Sirui Song, Hao Liu, and Xue Liu. Unbiased implicit feedback via bi-level optimization. *arXiv preprint arXiv:2206.00147*, 2022a.

Can Chen, Yingxue Zhang, Jie Fu, Xue Liu, and Mark Coates. Bidirectional learning for offline infinite-width model-based optimization. In *Advances in Neural Information Processing Systems*, 2022b.

Can Chen, Jingbo Zhou, Fan Wang, Xue Liu, and Dejing Dou. Structure-aware protein self-supervised learning. *Bioinformatics*, 2022c.

Can Chen, Yingxue Zhang, Xue Liu, and Mark Coates. Bidirectional learning for offline model-based biological sequence design, 2023a. URL `https://openreview.net/forum?id=luEG3j9LW5-`.

Lisha Chen, Songtao Lu, and Tianyi Chen. Understanding benign overfitting in gradient-based meta learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022d.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 2018.

Wenlin Chen, Austin Tripp, and José Miguel Hernández-Lobato. Meta-learning adaptive deep kernel gaussian processes for molecular property prediction. In *ICLR*, 2023b.

Yihong Chen, Bei Chen, Xiangnan He, Chen Gao, Yong Li, Jian-Guang Lou, and Yue Wang. λopt: Learn to regularize recommender models in finer levels. In *Special Interest Group on Knowledge Discovery and Data Mining*, 2019.

Zhixiang Chi, Yang Wang, Yuanhao Yu, and Jin Tang. Test-time fast adaptation for dynamic scene deblurring via meta-auxiliary learning. In *CVPR*, 2021.

Zhixiang Chi, Li Gu, Huan Liu, Yang Wang, Yuanhao Yu, and Jin Tang. Metafscil: A meta-learning approach for few-shot class incremental learning. In *CVPR*, 2022.

Snorre Christiansen, Michael Patriksson, and Laura Wynter. Stochastic bilevel programming in structural optimization. *Structural and multidisciplinary optimization*, 2001.

Tristan Deleu, David Kanaa, Leo Feng, Giancarlo Kerg, Yoshua Bengio, Guillaume Lajoie, and Pierre-Luc Bacon. Continuous-time meta-learning with forward mode differentiation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=57PipS27Km.

Yonatan Dukler, Alessandro Achille, Giovanni Paolini, Avinash Ravichandran, Marzia Polito, and Stefano Soatto. Diva: Dataset derivative of a learning task. *arXiv preprint arXiv:2111.09785*, 2021.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, 2017.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, 2017.

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*, 2018.

Boyan Gao, Henry Gouk, Yongxin Yang, and Timothy Hospedales. Loss function learning for domain generalization by implicit gradient. In *International Conference on Machine Learning*, 2022.

Saeed Ghadimi and Mengdi Wang. Approximation methods for bilevel programming. *arXiv preprint arXiv:1802.02246*, 2018.

Tommaso Giovannelli, Griffin Kent, and Luis Nunes Vicente. Bilevel stochastic methods for optimization and machine learning: Bilevel stochastic descent and darts. *arXiv preprint arXiv:2110.00604*, 2021.

Josif Grabocka, Randolf Scholz, and Lars Schmidt-Thieme. Learning surrogate losses. *arXiv preprint arXiv:1905.10108*, 2019.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, 2010.

Jiechao Guan, Yong Liu, and Zhiwu Lu. Fine-grained analysis of stability and generalization for modern meta learning algorithms. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022.

Markus Hiller, Mehrtash Harandi, and Tom Drummond. On enforcing better conditioned meta-learning for rapid few-shot adaptation. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022.

Zhiting Hu, Bowen Tan, Russ R Salakhutdinov, Tom M Mitchell, and Eric P Xing. Learning data manipulation for augmentation and weighting. In *Advances in Neural Information Processing Systems*, 2019.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 2018.

Haoming Jiang, Zhehui Chen, Yuyang Shi, Bo Dai, and Tuo Zhao. Learning to defend by learning to attack. In *International Conference on Artificial Intelligence and Statistics*, 2021.

Jikun Kang, Miao Liu, Abhinav Gupta, Chris Pal, Xue Liu, and Jie Fu. Learning multi-objective curricula for deep reinforcement learning. *arXiv preprint arXiv:2110.03032*, 2021.

Boris Knyazev, Michal Drozdzal, Graham W Taylor, and Adriana Romero Soriano. Parameter prediction for unseen deep architectures. *Advances in Neural Information Processing Systems*, 2021.

Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. Melu: Meta-learned user preference estimator for cold-start recommendation. In *Special Interest Group on Knowledge Discovery and Data Mining*, 2019.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

Shiye Lei and Dacheng Tao. A comprehensive survey to dataset distillation. *arXiv preprint arXiv:2301.05603*, 2023.

Jiangmeng Li, Wenwen Qiang, Yanan Zhang, Wenyi Mo, Changwen Zheng, Bing Su, and Hui Xiong. Metamask: Revisiting dimensional confounder for self-supervised learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

Lin Liu, Shanxin Yuan, Jianzhuang Liu, Liping Bao, Gregory Slabaugh, and Qi Tian. Self-adaptively learning to demoiré from focused and defocused image pairs. *Advances in Neural Information Processing Systems*, 2020.

Yuqi Liu, Bin Cao, and Jing Fan. Improving the accuracy of learning example weights for imbalance classification. In *International Conference on Learning Representations*, 2021.

Jonathan Lorraine and David Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*, 2018.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, 2020.

Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. Scalable gradient-based tuning of continuous regularization hyperparameters. In *International conference on machine learning*, 2016.

Kaifeng Lv, Shunhua Jiang, and Jian Li. Learning gradient descent: Better generalization and longer horizons. In *International Conference on Machine Learning*, 2017.

Fuyuan Lyu, Xing Tang, Huifeng Guo, Ruiming Tang, Xiuqiang He, Rui Zhang, and Xue Liu. Memorize, factorize, or be naıve: Learning optimal feature interaction methods for ctr prediction. *International Conference on Data Engineering*, 2021.

Chen Ma, Liheng Ma, Yingxue Zhang, Ruiming Tang, Xue Liu, and Mark Coates. Probabilistic metric learning with adaptive margin for top-k recommendation. In *Special Interest Group on Knowledge Discovery and Data Mining*, 2020.

Matthew MacKay, Paul Vicol, Jon Lorraine, David Duvenaud, and Roger Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. *arXiv preprint arXiv:1903.03088*, 2019.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 2010.

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, 2019.

Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel ridge-regression. *arXiv preprint arXiv:2011.00050*, 2020.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A Alemi, Jascha Sohl-Dickstein, and Samuel S Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. *arXiv preprint arXiv:1912.02803*, 2019.

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *International conference on machine learning*, 2016.

Aniruddh Raghu, Jonathan Lorraine, Simon Kornblith, Matthew McDermott, and David K Duvenaud. Meta-learning to improve pre-training. *Advances in Neural Information Processing Systems*, 2021.

Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in Neural Information Processing Systems*, 2019.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to Reweight Examples for Robust Deep Learning. In *International Conference on Machine Learning*, 2018.

Steffen Rendle. Learning recommender systems with adaptive regularization. In *Web Search and Data Mining*, 2012.

Junior Rojas, Eftychios Sifakis, and Ladislav Kavan. Differentiable implicit soft-body physics. *arXiv preprint arXiv:2102.05791*, 2021.

Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.

Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. In *International Conference on Artificial Intelligence and Statistics*, 2019.

Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. Meta-Weight-Net: Learning an Explicit Mapping For Sample Weighting. In *Advances in Neural Information Processing Systems*, 2019.

Jun Shu, Yanwen Zhu, Qian Zhao, Deyu Meng, and Zongben Xu. Meta-lr-schedule-net: learned lr schedules that scale and generalize. 2020.

Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. A review on bilevel optimization: from classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 2017.

Jihoon Tack, Jongjin Park, Hankook Lee, Jaeho Lee, and Jinwoo Shin. Meta-learning with self-improving momentum target. In *Advances in Neural Information Processing Systems*, 2022.

Heinrich Von Stackelberg. *Market structure and equilibrium.* Springer Science & Business Media, 2010.

Risto Vuorio, Shao-Hua Sun, Hexiang Hu, and Joseph J Lim. Multimodal model-agnostic meta-learning via task-aware modulation. *Advances in Neural Information Processing Systems*, 2019.

Shipeng Wang, Jian Sun, and Zongben Xu. Hyperadam: A learnable task-adaptive adam for network training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Xinyi Wang, Hieu Pham, Paul Michel, Antonios Anastasopoulos, Graham Neubig, and J. Carbonell. Optimizing Data Usage via Differentiable Rewards. In *International Conference on Machine Learning*, 2020.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International Conference on Machine Learning*, 2017.

Haolun Wu, Chen Ma, Yingxue Zhang, Xue Liu, Ruiming Tang, and Mark Coates. Adapting triplet importance of implicit feedback for personalized recommendation, 2022a.

Yichen Wu, Jun Shu, Qi Xie, Qian Zhao, and Deyu Meng. Learning to purify noisy labels via meta soft label corrector. In *AAAI*, 2021.

Yichen Wu, Long-Kai Huang, and Ying Wei. Adversarial task up-sampling for meta-learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022b.

Minkai Xu, Wujie Wang, Shitong Luo, Chence Shi, Yoshua Bengio, Rafael Gomez-Bombarelli, and Jian Tang. An end-to-end framework for molecular conformation generation via bilevel programming. In *International Conference on Machine Learning*, 2021.

Junjie Yang, Kaiyi Ji, and Yingbin Liang. Provably faster algorithms for bilevel optimization. *Advances in Neural Information Processing Systems*, 2021.

Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. Hierarchically structured meta-learning. In *International Conference on Machine Learning*, 2019.

Huaxiu Yao, Ying Wei, Long-Kai Huang, Ding Xue, Junzhou Huang, and Zhenhui Jessie Li. Functionally regionalized knowledge transfer for low-resource drug discovery. *Advances in Neural Information Processing Systems*, 2021.

Chia-Hung Yuan and Shan-Hung Wu. Neural tangent generalization attacks. In *International Conference on Machine Learning*, 2021.

Jonas Zehnder, Yue Li, Stelian Coros, and Bernhard Thomaszewski. Ntopo: Mesh-free topology optimization using implicit neural representations. *Advances in Neural Information Processing Systems*, 2021.

An Zhang, Fangfu Liu, Wenchang Ma, Zhibo Cai, Xiang Wang, and Tat-Seng Chua. Boosting causal discovery via adaptive sample reweighting. In *ICLR*, 2023.

Biao Zhang and Peter Wonka. Training data generating networks: Shape reconstruction via bi-level optimization. In *International Conference on Learning Representations*, 2021.

Yihua Zhang, Yuguang Yao, Parikshit Ram, Pu Zhao, Tianlong Chen, Mingyi Hong, Yanzhi Wang, and Sijia Liu. Advancing model pruning via bi-level optimization. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022.

Tao Zhong, Zhixiang Chi, Li Gu, Yang Wang, Yuanhao Yu, and Jin Tang. Meta-dmoe: Adapting to domain shift by meta-distillation from mixture-of-experts. *arXiv preprint arXiv:2210.03885*, 2022.

Yingtian Zou, Fusheng Liu, and Qianxiao Li. Unraveling model-agnostic meta-learning via the adaptation learning rate. In *International Conference on Learning Representations*, 2021.