DELVING INTO THE HIERARCHICAL STRUCTURE FOR EFFICIENT LARGE-SCALE BI-LEVEL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent years have witnessed growing interest and emerging successes of bi-level learning in a wide range of applications, such as meta learning and hyper-parameter optimization. While current bi-level learning approaches suffer from high memory and computation costs especially for large-scale deep learning scenarios, which is due to the hierarchical optimization therein. *It is therefore interesting to know whether the hierarchical structure can be untied for efficient learning*. To answer this question, we introduce NSGame that, transforming the hierarchical bi-level learning problem into a parallel Nash game, incorporates the tastes of hierarchy by a very small scale Stackelberg game. We prove that strong differential Stackelberg equilibrium (SDSE) of the bi-level learning problem corresponds to local Nash equilibrium of the NSGame. To obtain such SDSE from NSGame, we introduce a two-time scale stochastic gradient descent (TTS-SGD) method, and provide theoretical guarantee that local Nash equilibrium obtained by the TTS-SGD method is SDSE of the bi-level learning problem. We compare NSGame with representative bi-level learning models, such as MWN and MLC, experimental results on class imbalance learning and noisy label learning have verified that the proposed NSGame achieves comparable and even better results than the corresponding meta learning models, while NSGame is computationally more efficient.

1 INTRODUCTION

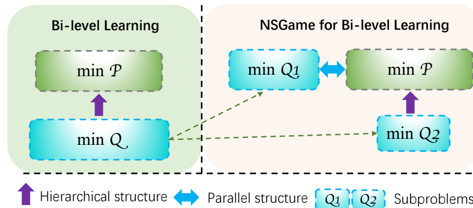

Figure 1: Comparison of the current method for bi-level learning with the proposed NSGame.

Bi-level learning, which models learning problem with leader and follower hierarchical structure by a bi-level optimization problem, has achieved great success in a diverse set of deep learning scenarios, such as few shot learning Finn et al. (2017); Chen et al. (2021b), robust deep learning Ren et al. (2018); Shu et al. (2019), learning to optimize Chen et al. (2021a), adversarial learning Tian et al. (2021) as well as reinforcement learning Stadie et al. (2020). Bi-level learning is also a promising method to replace the hand-crafted learning preconditions, such as the hyper-parameters Franceschi et al. (2018); Wang et al. (2021), network architectures Shaw et al. (2019), data augmentation Hataya et al. (2022) as well as label correction Wang et al. (2020), with those learned in a data-driven way. Recent years have witnessed the continue breakthrough of bi-level learning in more research fields Hospedales et al. (2021); Vanschoren (2018), especially for learning and vision problem Liu et al. (2021).

Mathematically, bi-level learning is formulated as a two-player Stackelberg game with leader and follower hierarchical optimization structure, as

$$\begin{cases} \min\limits_{\boldsymbol{x}\in\mathbb{R}^n} f_1(\boldsymbol{x},\boldsymbol{y}) \\ s.t.\ \ \boldsymbol{y}\in \arg\min\limits_{\boldsymbol{z}\in\mathbb{R}^m} f_2(\boldsymbol{x},\boldsymbol{z}) \end{cases} \tag{1}$$

The leader aims to minimize its cost $f_1(\boldsymbol{x},\boldsymbol{y})$ through its strategy $\boldsymbol{x}$ and the best response $\boldsymbol{y}\in\arg\min_{\boldsymbol{z}} f_2(\boldsymbol{x},\boldsymbol{z})$ of the follower given $\boldsymbol{x}$. The scale of the follower's problem is decisive to the optimization of Eq. (1). When the dimension of the optimization variable $\boldsymbol{y}$ is not very large,

gradient descent algorithm can be used to solve Eq. (1) efficiently. While for practical applications in deep learning as aforementioned, the follower aims to train a very deep neural network with millions of network parameters (denoted by $\boldsymbol{y}$). In these situations, the hierarchical optimization structure makes the learning process of Eq. (1) computationally more challenging Dempe (2018), even though the dimension of $\boldsymbol{x}$ is in general very small compared to that of $\boldsymbol{y}$ ($m \gg n$). To be more specific, minimizing $f_1(\boldsymbol{x}, \boldsymbol{y})$ with simple first-order gradient based method need to compute the total derivative $Df_1 = D_1 f_1 + D\boldsymbol{y}^\top D_2 f_1$ [1]. According to implicit function theorem, $D\boldsymbol{y}$ involves computing the inverse of matrix of size $m \times m$, which is both time and memory consuming, especially for large m. Although simplified methods have been developed to approximate the Jacobian matrix $D\boldsymbol{y}$ with convergence guarantee Dempe (2018); Shaban et al. (2019); Baydin et al. (2018), they need to compute the second-order derivatives and extending the computational graph, particularly for learning with deep neural networks. In consequence, the computation and memory costs of these methods are still demanding, which leads to the super-slow training of current meta learning approaches, such as Ren et al. (2018); Shu et al. (2019); Wang et al. (2020).

Suffering from the computational difficulties of the large-scale bi-level optimization, *it is therefore very interesting and important to know whether the hierarchical optimization structure in the large-scale bi-level learning can be untied for efficient learning.* In this paper, we aim to answer this question by introducing a NSGame model which unties the hierarchical structure by splitting the follower's large-scale problem into two sub-problems. Then we transform the Stackelberg game into hybrid Nash and Stackelberg game based on these two sub-problems. NSGame incorporates some tastes of hierarchy by a small-scale Stackelberg game, as illustrated in Fig. 1. In Fig. 1, the left part is a bi-level learning problem with hierarchical optimization structure, the right part is our NSGame with parallel optimization structure among two sub-problems where one is a small scale hierarchical optimization problem. NSGame alleviates the computational difficulties of current bi-level learning as long as the scale of its Stackelberg game sub-problem is small.

Related works. The Stackelberg game Eq. (1) offers an elegant and mathematically solid framework to study meta learning and hyper-parameter learning Liu et al. (2021); Franceschi et al. (2018). Specifically, the leader's variable can be the initialization of the network parameters as MAML Finn et al. (2017), the learning rate in SGD Shu et al. (2020), parameters of the meta weight net MWN Shu et al. (2019) for sample re-weighting, and parameters of the meta label corrector for noisy label learning Wang et al. (2020). Please refer to Hospedales et al. (2021); Vanschoren (2018) on bi-level meta learning for more applications. Although the bi-level hierarchical structure in Eq. (1) benefits the mathematical modeling of a variety of learning problems, it brings challenge for computation especially for deep learning problems as aforementioned.

In recent years, great efforts have been made to improve the computational efficiency of algorithms for solving bi-level learning model Eq. (1). According to the computation of the Jacobian matrix $D\boldsymbol{y}$, most of the existing methods can be classified into two categories: implicit method and explicit method. The works in Rajeswaran et al. (2019); Lorraine et al. (2020) use implicit function theorem to obtain Df_1 without storing the internal variables and extending computational graph, while they assume that the inner loop converges to the optimal solution of the follower's problem, which is not practical for large-scale problems. Explicit approximated gradient methods such as Luketina et al. (2016) uses one inner loop, few inner loops Maclaurin et al. (2015) or many inner loop steps Shaban et al. (2019); Hong et al. (2020) to approximate the $D\boldsymbol{y}$. Although the works in Luketina et al. (2016); Maclaurin et al. (2015); Shaban et al. (2019) have been practically used as in Shu et al. (2019); Wang et al. (2020); Lee et al. (2019); Yao et al. (2021), these methods need to compute the second-order derivative $D_{21} f_2$ as an approximation of $D\boldsymbol{y}$, which extends the computation graph and imposes computational and memory burden. Very recently, Bohdal et al. (2021) introduced EvoGrad to compute the gradient Df_1 using evolutionary techniques. Specifically, EvoGrad uses zeroth-order evolutionary method for solving the follower's problem and first-order gradient descent for minimizing the leader's cost. Yet EvoGrad is still not applicable to very large scale models when using larger model population.

Contribution. Different to these existing methods, we introduce a new model to characterize the bi-level learning problem Eq. (1). The main contributions of this paper are three-fold: (1) We propose a theoretically sound and practically efficient approach, NSGame, for bi-level learning.

[1] Although the best response of the follower may not be unique, sufficient conditions such as $D_2 f_2(\boldsymbol{x}, \boldsymbol{y}) = 0$ and $\det(D_2^2 f_2) \neq 0$ guarantee that $D\boldsymbol{y}$ is well defined.

The computation of NSGame involves only a small scale bi-level learning sub-problem which is easy to solve compared to the original large-scale bi-level learning model; (2) We prove that strong differential Stackelberg equilibrium (SDSE) of the bi-level learning problem corresponds to local Nash equilibrium of our NSGame. To obtain SDSE, we develop a two-time scale stochastic gradient (TTS-SGD) method to solve NSGame, and provide theoretical guarantee for the convergence of TTS-SGD to SDSE; (3) Experimental results on toy example, class imbalance learning and noisy label learning verify the effectiveness and efficiency of our proposed TTS-SGD for solving NSGame. Specifically, NSGame achieves comparable performance with representative methods such as MWN Shu et al. (2019), TBP Shaban et al. (2019) for MWN Shu et al. (2019) and MLC Wang et al. (2020).

Organization. This paper is organized as follows. Section 2 introduces some preliminary works and definitions. Section 3 presents our NSGame model and theoretical analysis on the relation between NSGame and bi-level model, develops a two-time scale stochastic gradient descent method (TTS-SGD). Section 4 gives the theoretical analysis on the convergence of TTS-SGD algorithm to the optimal solution of the bi-level model. Section 5 demonstrates the effectiveness of NSGame on toy example and real applications. Finally, we conclude our work at Section 6.

Notation. Throughout this paper, we denote by $D_i f$ as the derivative of f *w.r.t.* the i-th variable. $D_{ij} f$ represents the partial derivative of $D_i f$ *w.r.t.* the j-th variable ($i \neq j$), $D_i^2 f = D_{ii} f$ represents the second-order derivative of f *w.r.t.* the i-th variable and Df represents the total derivative. Specifically, for $f_1(\boldsymbol{x}, \boldsymbol{y})$ in Eq. (1), $D_1 f_1$ indicates the derivative of f_1 *w.r.t.* the first variable $\boldsymbol{x}$. $D_2 f_1$ means the derivative of f_1 *w.r.t.* the second variable $\boldsymbol{y}$. As $\boldsymbol{y}$ is an implicit function of $\boldsymbol{x}$, Df_1 is $D_1 f_1 + D\boldsymbol{y}^\top D_2 f_1$ with $D\boldsymbol{y}$ the derivative of $\boldsymbol{y}$ *w.r.t.* $\boldsymbol{x}$ as $D\boldsymbol{y} = -(D_2^2 f_2) \circ D_{21} f_2$ (Implicit function theorem). $D^2 f_1$ is the second-order total derivative. $M \succ 0$ denotes the matrix M is positive definite. $a \ll b$ means a is much smaller than b.

2 Preliminaries

We now introduce the games related to the bi-level learning and our proposed model, present the concepts of equilibriums of these games, and characterize their optimality condition. Specifically, consider a non-cooperative game between two players, with costs $f_1(\boldsymbol{x}, \boldsymbol{y}) : X \times Y \to \mathbb{R}$ for the first player and $f_2(\boldsymbol{x}, \boldsymbol{y}) : X \times Y \to \mathbb{R}$ for the second player, where $X \subseteq \mathbb{R}^n$, $Y \subseteq \mathbb{R}^m$ denote the action space of first and second player respectively. Throughout this paper, we assume f_1 and f_2 are sufficiently smooth, and the action space $X \times Y$ are continuous.

The games we study can be classified into Stackelberg game and Nash game according to the behavior of these two players: in leader-follower play or simultaneous play. Eq. (1) presents a **Stackelberg game** where the two players play in leader-follower manner. This contrasts with **Nash game** where the two players play simultaneously as

$$(\boldsymbol{P}_1) \ \min_{\boldsymbol{x} \in X} f_1(\boldsymbol{x}, \boldsymbol{y}) \qquad\qquad (\boldsymbol{P}_2) \ \min_{\boldsymbol{y} \in Y} f_2(\boldsymbol{x}, \boldsymbol{y}). \tag{2}$$

In Eq. (2) the two players minimize their costs through their own strategy given the other, and the optimization is in single level. The learning algorithm for both the Stackelberg game and the Nash game can be implemented through myopic update roles such as gradient descent (GD/SGD).

Given these two kinds of games, we then introduce the corresponding equilibriums which are critical for our optimization and follow up analysis.

Definition 2.1. (Differential Stackelberg Equilibrium (DNE)). Fiez et al. (2020) *For the Stackelberg game in Eq.* (1)*, the strategy* $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ *is a differential Stackelberg equilibrium, if* $Df_1(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0, D^2 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) \succ 0$*, and* $D_2 f_2(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0, D_2^2 f_2(\boldsymbol{x}^*, \boldsymbol{y}^*) \succ 0$.

Definition 2.2. (Differential Nash Equilibrium (DSE)). Ratliff et al. (2016) *For the Nash game in Eq.* (2)*, the strategy* $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ *is a differential Nash equilibrium, if* $D_1 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0, D_1^2 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) \succ 0$*, and* $D_2 f_2(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0, D_2^2 f_2(\boldsymbol{x}^*, \boldsymbol{y}^*) \succ 0$.

The equilibriums are the only optimal that we can obtain by first-order algorithm such as GD/SGD. It has been proved in Mazumdar et al. (2020); Jin et al. (2020) that DNE and DSE are closely related to local Nash equilibrium and local Stackelberg equilibrium for general sum game and zero sum games.

3 METHODOLOGY

3.1 THE PROPOSED NSGAME MODEL

To alleviate the computation and memory burden for solving Eq. (1), we propose to split the follower's problem into two subproblems with variables $\boldsymbol{y}_1 \in Y_1 \subseteq \mathbb{R}^{m_1}$ and $\boldsymbol{y}_2 \in Y_2 \subseteq \mathbb{R}^{m_2}$ such that $Y = Y_1 \times Y_2 \subseteq \mathbb{R}^m$. Then we introduce the following hybrid Nash and Stackelberg game model (NSGame) as

$$(\boldsymbol{P}_1)\ \min_{\boldsymbol{y}_1 \in Y_1} f_2(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2) \qquad (\boldsymbol{P}_2)\ \begin{cases} \min_{\boldsymbol{x} \in X} f_1(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2) \\ s.t.\ \ \boldsymbol{y}_2 \in \arg\min_{\boldsymbol{z} \in Y_2} f_2(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{z}) \end{cases} \tag{3}$$

The model Eq. (3) is a Nash game with two players where $(\boldsymbol{P}_1)$ aims to minimize f_2 *w.r.t.* $\boldsymbol{y}_1$ given $\boldsymbol{x}$ and $\boldsymbol{y}_2$, and $(\boldsymbol{P}_2)$ aims to minimize f_1 *w.r.t.* $\boldsymbol{x}$ given $\boldsymbol{y}_1$. Particularly, the problem $(\boldsymbol{P}_2)$ is a Stackerlberg game with very small scale lower-level subproblem ($m_2 < m_1$). For bi-level learning in deep neural networks, $\boldsymbol{y}_2$ can be the parameter a sublayer (e.g. "FC layer" in ResNet) and $\boldsymbol{y}_1$ is the parameter of the remaining layers. In this case, the follower's problem of $(\boldsymbol{P}_2)$ can be strongly convex (with $\|\boldsymbol{y}_2\|_2^2$ regularization) with single optimal.

NSGame in Eq. (3) unties the bi-level hierarchical structure of the Eq. (1) to single level parallel structure, therefore first-order methods for solving Eq. (3) can be more efficient than solving Eq. (1), especially for bi-level optimization with very large-scale lower-level subproblem such as meta learning Liu et al. (2021); Shu et al. (2019); Lee et al. (2019). Specifically, one only needs to compute and store the second-order derivate $D_{31}f_2$ of a small scale subproblem where $m_2 \ll m$. Since Eq. (3) is easy to solve, *it is therefore very important to know can we obtain the solution of bi-level model in Eq.* (1) *from NSGame in Eq.* (3). The answer is affirmative under some mild conditions.

3.2 THEORETICAL ANALYSIS

Before presenting our main results, we first give a sketch of our analysis and then introduce the definition of strong differential Stackelberg equilibrium which is critical to our theoretical results.

Sketch of analysis. To answer the aforementioned question, we need to connect DSE of Eq. (1) with DNE of Eq. (3) and try to obtain the DSE of Eq. (1) by solving Eq. (3). To this end, we present three theorems in this section (Theorem 3.1, Theorem 3.2 and Theorem 3.3) to show the relation between DSE of Eq. (1) and DNE of Eq. (3). Theorem 3.1 reveals that the strong differential Stackelberg equilibrium (SDSE) of Eq. (1) corresponds to the DNE of Eq. (3). Theorem 3.2 shows that under the coherent condition every DSE of Eq. (1) is a DSDE, thus it corresponds to the DNE of Eq. (3). Theorem 3.3 indicates that the DNE with certain structure corresponds to the DSE of Eq. (1).

Definition 3.1. (Strong Differential Stackelberg Equilibrium(SDSE)) *For the Stackelberg game Eq.* (1)*, a joint strategy* $(\boldsymbol{x}^*, \boldsymbol{y}^*) \in X \times Y$ *is a strong differential Stackelberg equilibrium if* $D_1 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0$, $D_2 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0$, $D_2 f_2(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0$, $D_2^2 f_2(\boldsymbol{x}^*, \boldsymbol{y}^*) \succ 0$ *and*

$$\begin{bmatrix} D_1^2 f_1 & D_{12} f_1 \\ D_{21} f_1 & D_2^2 f_1 \end{bmatrix} \succ 0. \tag{4}$$

It is easy to verify that for Eq. (1) every SDSE is a DSE, but not vice versa. Moreover, SDSE is very important equilibrium for Stackelberg game and our NSGame, we have the following results.

Proposition 3.1. *If the Stackelberg game Eq.* (1) *has SDSE, then its global optimal must be a SDSE.*

Theorem 3.1. *Let* $(\boldsymbol{x}^*, \boldsymbol{y}^*) \in X \times Y$ *be a SDSE of Eq.* (1) *with* $\boldsymbol{y}^* = (\boldsymbol{y}_1^*, \boldsymbol{y}_2^*) \in Y_1 \times Y_2$*, then* $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ *is a DNE of Eq.* (3)*.*

Proposition 3.2. *If the Stackelberg game Eq.* (1) *has SDSE, then the pareto optimal Nash equilibrium of the corresponding NSGame Eq.* (3) *is the global optimal of Eq.* (1)*.*

The Proposition 3.1 and Proposition 3.2 show some interesting properties of SDSE. The Theorem 3.1 indicates that SDSE of the Stackelberg game Eq. (1) is a DNE of Eq. (3). While in general, not every DSE of a Stackelberg game Eq. (1) is SDSE. Next, we show that under some mild conditions Stackelberg game has only SDSE.

Definition 3.2. (Coherent Condition) *Two functions* $f_1(\boldsymbol{x}, \boldsymbol{y})$ *and* $f_2(\boldsymbol{x}, \boldsymbol{y})$ *are said to be coherent if there exits* $\boldsymbol{x}^*$ *such that local optimality of* $\boldsymbol{y}^*$ *for* $f_2(\boldsymbol{x}^*, \boldsymbol{y})$ *implies the local optimality of* $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ *for* $f_1(\boldsymbol{x}, \boldsymbol{y})$*.*

Theorem 3.2. *If the two functions $f_1(\boldsymbol{x}, \boldsymbol{y})$ and $f_2(\boldsymbol{x}, \boldsymbol{y})$ in Eq.* (1) *are coherent, then every DSE of Eq.* (1) *is a SDSE and thus is a DNE of Eq.* (3)*.*

Coherent is a mild condition for a lot of real applications such as meta learning for hyper-parameter optimization Franceschi et al. (2018), meta learning for class imbalance learning Menon et al. (2020); Shu et al. (2019) and noisy label learning Patrini et al. (2017); Wang et al. (2020). Specifically, in hyper-parameter optimization, $\boldsymbol{x}$ is the hyper-parameter, $\boldsymbol{y}$ can be the parameters of a deep network. Then given an optimal $\boldsymbol{x}^*$, we aim to learn the optimal $\boldsymbol{y}^*$ which not only minimizes the training loss, but also minimizes the validation loss, thus the training loss and the validation loss are coherent. With coherent condition, Theorem 3.2 indicates that the DSE of Eq. (1) is a DNE of the corresponding NSGame Eq. (3).

The above results reveal that every SDSE must be a DNE (SDSE $\Rightarrow$ DNE). While in general not every DNE is a SDSE (DNE $\nRightarrow$ SDSE). The following questions then arise naturally: *1) What kind of DNE in Eq.* (3) *is SDSE? 2) Can we guarantee to obtain such DNE?* To answer the first question, we need to introduce the following assumption:

Assumption 3.1. *1) Functions $f_1(\boldsymbol{x}, \boldsymbol{y})$ and $f_2(\boldsymbol{x}, \boldsymbol{y})$ are coherent and $f_1(\boldsymbol{x}, \boldsymbol{y})$ is separable as $f_1(\boldsymbol{x}, \boldsymbol{y}) = h(\boldsymbol{x}) + \psi(\boldsymbol{y}_1, \boldsymbol{y}_2)$; 2) The positive definiteness of $D_2^2 f_1$ is consistent with $D_2^2 f_2$, $\forall(\boldsymbol{x}, \boldsymbol{y}) \in X \times Y$; 3) For function $f_1(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2)$, the equality $D_1 f_1 + D\boldsymbol{y}_2^\top D_3 f_1 = 0$ implies $D\boldsymbol{y}_1^\top D_2 f_1 = 0$.*

The assumptions are mild for a variety of bi-level learning problems, such as, hyper-parameter learning Franceschi et al. (2018), meta learning Menon et al. (2020); Shu et al. (2019); Patrini et al. (2017); Wang et al. (2020). With these assumptions, we have the following result:

Theorem 3.3. *Assume the above assumptions are satisfied by the Stackelberg game Eq.* (1)*. Let $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ be a DNE of Eq.* (3)*. If $\boldsymbol{y}^* = (\boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ is a function of $\boldsymbol{x}^*$, that is $\boldsymbol{y}^* = \phi(\boldsymbol{x}^*)$, and $D_2^2 f_2 \succ 0$ for $f_2(\boldsymbol{x}, \boldsymbol{y})$ at $(\boldsymbol{x}^*, \boldsymbol{y}^*)$, then $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ is a SDSE.*

Theorem 3.3 indicates that if $\boldsymbol{y}_1^*$ and $\boldsymbol{y}_2^*$ are closely related to $\boldsymbol{x}^*$ and they minimize $f_2(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2)$, then the local Nash equilibrium corresponds to the DSE of Eq. (1). Proofs of the theoretical results are provided in supplementary materials (SM) Section A. To answer the second question, we need an optimization method which is able to find an equilibrium $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ satisfying $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*) = \phi(\boldsymbol{x}^*)$.

3.3 Two-time scale SGD algorithm

First-order algorithm, such as gradient descent (GD) or stochastic gradient descent (SGD)[2], can be used to optimize Eq. (3). While to enhance the hierarchical structure between $\boldsymbol{x}$ and $\boldsymbol{y}$, such that at equilibrium $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$, $\boldsymbol{y}_1^*$ and $\boldsymbol{y}_2^*$ is function of $\boldsymbol{x}^*$ as $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*) = \phi(\boldsymbol{x}^*)$, we introduce a two-time scale SGD for our NSGame. Specifically, the variables in $(\boldsymbol{P}_1)$ Eq. (3) is updated by

$$\boldsymbol{y}_1^{t+1} = \boldsymbol{y}_1^t - \alpha_t \hat{g}_1(\boldsymbol{x}^t, \boldsymbol{y}_1^t, \boldsymbol{y}_2^t) \tag{5}$$

where $\hat{g}_1$ is the gradient of f_2 *w.r.t.* $\boldsymbol{y}_1$ as $D_2 f_2$. While $(\boldsymbol{P}_2)$ in Eq. (3) is bi-level optimization with lower-level variable $\boldsymbol{y}_2$, here we adopt an online update strategy as Shu et al. (2019) to update $\boldsymbol{y}_2$ and $\boldsymbol{x}$ through a single inner loop as [3]

$$\boldsymbol{y}_2^{t+1} = \boldsymbol{y}_2^t - \alpha_t \hat{g}_2(\boldsymbol{x}^t, \boldsymbol{y}_1^t, \boldsymbol{y}_2^t) \tag{6}$$

$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - \beta_t \hat{h}(\boldsymbol{x}^t, \boldsymbol{y}_1^t, \boldsymbol{y}_2^{t+1}) \tag{7}$$

where $\hat{g}_2$ is the gradient of f_2 *w.r.t.* $\boldsymbol{y}_2$ as $D_3 f_2$ and $\hat{h}$ is full gradient of f_1 *w.r.t.* $\boldsymbol{x}$ as $D_1 f_1 + D\boldsymbol{y}_2^\top D_3 f_1$. As the optimal $\boldsymbol{y}_2$ in Eq. (3) is approximated by $\boldsymbol{y}_2^{t+1}$ in Eq. (6), thus $D\boldsymbol{y}_2$ is approximated by the gradient of $\boldsymbol{y}_2^{t+1}$ *w.r.t.* $\boldsymbol{x}$, $D\boldsymbol{y}_2 = -\alpha_t \frac{\partial \hat{g}_2}{\partial \boldsymbol{x}} = -\alpha_t D_{31} f_2$. As the dimension of $D_{31} f_2$ for $f_2(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2)$ is generally much smaller than $D_{21} f_2$ for $f_2(\boldsymbol{x}, \boldsymbol{y})$ with $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2)$, the training of our NSGame using Eq. (7) is much more efficient than existing bi-level learning methods Menon et al. (2020); Shu et al. (2019); Lee et al. (2019); Wang et al. (2020); Yao et al. (2021).

[2] In this paper, we mainly concern the learning problems where only noisy gradient is available.

[3] Note that other advanced bi-level optimization algorithms with multiple inner loop such as Maclaurin et al. (2015); Shaban et al. (2019) can also be used to solve the lower-level problem of (P2) in Eq. (3), while we find single inner loop is sufficient to guarantee the convergence as proved in Shu et al. (2019).

More importantly we employ two-time scale learning rule Borkar (1997); Heusel et al. (2017) by setting α_t and β_t of different orders. Specifically, the learning rate β_t of the variable $\boldsymbol{x}$ is much smaller than the learning rate α_t of $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$. In this way, the variable $\boldsymbol{x}$ is almost static when updating $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ in the learning process until $(\boldsymbol{y}_1, \boldsymbol{y}_2)$ converges to an equilibrium $\phi(\boldsymbol{x})$. In consequence, the variable $\boldsymbol{x}$ will lead the entire learning process, which is important for our algorithm to converge to a SDSE of Eq. (1). We summarize our two-time scale stochastic gradient descent (TTS-SGD) algorithm in Algorithm 1.

Algorithm 1: TTS-SGD for NSGame.

Input : Initialization of $\boldsymbol{x}$, $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2)$. Number of iterations T and the learning rate α_0, β_0.
Output : Optimal variables $\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*$.
while $t \leq T$ **do**
 Update learning rate α_t and β_t ;
 Update variable $\boldsymbol{y}_1$ of $(\boldsymbol{P}_1)$ by Eq. (5) ;
 Update variable $\boldsymbol{y}_2$ of $(\boldsymbol{P}_2)$ by Eq. (6) ;
 Update variable $\boldsymbol{x}$ of $(\boldsymbol{P}_2)$ by Eq. (7);
end

4 CONVERGENCE AND COMPLEXITY ANALYSIS

Two-time scale learning plays an important role in the asymptotic convergence analysis of stochastic approximation Borkar (1997); Prasad et al. (2015), it has wide applications in learning continue games such as GAN Heusel et al. (2017) and Bi-level optimization Hong et al. (2020). In this section, we study the convergence of TTS-SGD to the SDSE of Eq. (1), and analyze the time and memory complexity of TTS-SGD.

4.1 CONVERGENCE ANALYSIS.

In our NSGame and its application to learning problems, only noisy gradient is available, therefore the update of $\boldsymbol{x}$, $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ are actually

$$\begin{cases} \boldsymbol{x}^{t+1} = \boldsymbol{x}^t - \beta_t h(\boldsymbol{x}^t, \boldsymbol{y}_1^t, \boldsymbol{y}_2^t) + \mathfrak{N}_{\boldsymbol{x}}^t \\ \boldsymbol{y}_1^{t+1} = \boldsymbol{y}_1^t - \alpha_t g_1(\boldsymbol{x}^t, \boldsymbol{y}_1^t, \boldsymbol{y}_2^t) + \mathfrak{N}_{\boldsymbol{y}_1}^t \\ \boldsymbol{y}_2^{t+1} = \boldsymbol{y}_2^t - \alpha_t g_2(\boldsymbol{x}^t, \boldsymbol{y}_1^t, \boldsymbol{y}_2^t) + \mathfrak{N}_{\boldsymbol{y}_2}^t \end{cases}, \tag{8}$$

where $h(\boldsymbol{x}^t, \boldsymbol{y}_1^t, \boldsymbol{y}_2^t)$, $g_1(\boldsymbol{x}^t, \boldsymbol{y}_1^t, \boldsymbol{y}_2^t)$ and $g_2(\boldsymbol{x}^t, \boldsymbol{y}_1^t, \boldsymbol{y}_2^t)$ are the exact gradient of $\boldsymbol{x}, \boldsymbol{y}_1$ and $\boldsymbol{y}_2$. $\mathfrak{N}_{\boldsymbol{x}}^t$, $\mathfrak{N}_{\boldsymbol{y}_1}^t$ and $\mathfrak{N}_{\boldsymbol{y}_2}^t$ are stochastic errors.

To prove the convergence of TTS-SGD, we make the following assumptions as in Heusel et al. (2017):

1. The gradients g_1, g_2 and h are Lipschitz.
2. $\sum_{t=1}^{\infty} \alpha_t = \infty$, $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$, $\sum_{t=1}^{\infty} \beta_t = \infty$, $\sum_{t=1}^{\infty} \beta_t^2 < \infty$, $\beta_t = o(\alpha_t)$.
3. The stochastic gradient error $\mathfrak{N}_{\boldsymbol{x}}^t$, $\mathfrak{N}_{\boldsymbol{y}_1}^t$ and $\mathfrak{N}_{\boldsymbol{y}_2}^t$ are martingale difference sequences *w.r.t.* the increasing σ-field $\mathcal{F}_n = \sigma(\boldsymbol{x}^t, \boldsymbol{y}_1^t, \boldsymbol{y}_2^t, \mathfrak{N}_{\boldsymbol{x}}^t, \mathfrak{N}_{\boldsymbol{y}_1}^t, \mathfrak{N}_{\boldsymbol{y}_2}^t, t \leq n), n \geq 0$ with $\mathbb{E}\left[\|\mathfrak{N}_{\boldsymbol{x}}^n\|^2 | \mathcal{F}_n\right] \leq B_0$, $\mathbb{E}\left[\|\mathfrak{N}_{\boldsymbol{y}_1}^n\|^2 | \mathcal{F}_n\right] \leq B_1$ and $\mathbb{E}\left[\|\mathfrak{N}_{\boldsymbol{y}_2}^n\|^2 | \mathcal{F}_n\right] \leq B_2$, where B_0, B_1 and B_2 are positive deterministic constant.
4. For each $\boldsymbol{x}$, the ODE $\dot{\boldsymbol{y}}(t) = g(\boldsymbol{x}, \boldsymbol{y}(t))$ where $g(\boldsymbol{x}, \boldsymbol{y}(t))$ is the gradient of $f_2(\boldsymbol{x}, \boldsymbol{y}(t))$ *w.r.t.* $\boldsymbol{y} = (\boldsymbol{y}_1(t), \boldsymbol{y}_2(t))$, has a locally asymptotically stable attractor $\phi(\boldsymbol{x})$ with the domain of f_2 such that ϕ is Lipschitz. The ODE $\dot{\boldsymbol{x}}(t) = h(\boldsymbol{x}(t), \phi(\boldsymbol{x}(t)))$ has a locally asymptotically stable attractor $\boldsymbol{x}^*$ with the domain of f_1.
5. $\sup_t \|\boldsymbol{x}^t\| \leq \infty$, $\sup_t \|\boldsymbol{y}_1^t\| \leq \infty$ and $\sup_t \|\boldsymbol{y}_2^t\| \leq \infty$.

The following theorem indicates the convergence of TTS-SGD.

Theorem 4.1. *Borkar (1997) The iterates Eq.* (5)*, Eq.* (7) *and Eq.* (6) *converges to* $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ *where* $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*) = \phi(\boldsymbol{x}^*)$ *a.s., if the above assumptions are satisfied.*

The solution $(\boldsymbol{x}^*, \phi(\boldsymbol{x}^*))$ is a stationary local Nash equilibrium of Eq. (3) Heusel et al. (2017), since $\boldsymbol{x}^*$ and $\phi(\boldsymbol{x}^*)$ are locally asymptotically stable attractors with $h(\boldsymbol{x}^*, \phi(\boldsymbol{x}^*)) = 0$ and $g(\boldsymbol{x}^*, \phi(\boldsymbol{x}^*)) = 0$. The obtained local Nash equilibrium $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ is a function of $\boldsymbol{x}^*$, therefore, there are strong hierarchical relation between $\boldsymbol{x}^*$ and the corresponding $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$. According to Theorem 3.3 and Theorem 4.1 we know that TTS-SGD for solving Eq. (3) converges to a SDSE of the Eq. (1).

Remark. Note that using two time scales is very important for our algorithm converge to the SDSE of Eq. (1), as the separated time scales enhance the hierarchical structure between the learning variables. Specifically, the slower update variable becomes the leader variable and fast update variable becomes the follower variable. Without the separated time scales, the update process may not converge as analyzed in Prasad et al. (2015); Fiez et al. (2020). Our results in Fig. 2 also verifies that using separate learning rate properly is important for our algorithm to converge.

4.2 COMPLEXITY ANALYSIS.

The computation of the vanilla bi-level model Eq. (1) using gradient based methods Luketina et al. (2016); Maclaurin et al. (2015); Shaban et al. (2019) requires higher-order gradients $D_{21}f_2$. For bi-level learning in deep networks, such as hyper-parameter learning and meta learning, higher-order gradient computation is especially time and memory consuming when the dimension of $\boldsymbol{y}$ is very large, since it needs to store all the intermediate variables in memory and extend the computational graph for back-propagation on back-propagation. Specifically, the big-$\mathcal{O}$ time and memory requirements for computing $D_{21}f_2$ is $\mathcal{O}(mn)$ and $\mathcal{O}(m+n)$.

Table 1: Comparisons of time and memory complexity.

Model	Time requirement	Memory requirement
Bi-level	$\mathcal{O}(mn)$	$\mathcal{O}(m+n)$
NSGame	$\mathcal{O}(dn)$	$\mathcal{O}(d+n)$

In contrast to vanilla model Eq. (1), our NSGame only needs to compute a small scale second-order gradients $D_{31}f_2 \in \mathbb{R}^{d\times n}(d \ll m)$ for $f_2(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2)$. The benefits of computing $D_{31}f_2 \in \mathbb{R}^{d\times n}$ over $D_{21}f_2 \in \mathbb{R}^{m\times n}$ for $f_2(\boldsymbol{x}, \boldsymbol{y})$are two-fold. On one hand, since $d \ll m$, the time requirement of $D_{31}f_2$ ($\mathcal{O}(dn)$) is much smaller than that of $D_{21}f_2$ ($\mathcal{O}(mn)$). On the other hand, $\boldsymbol{y}_2$ can be the parameters of one layer or a shallow subnetwork of a deep network, such as the FC layer of ResNet He et al. (2016), thus we do not need to store the intermediate variables and extend the corresponding computational graph. The memory requirements then reduce to $\mathcal{O}(d+n)$. Therefore, gradient based method for solving NSGame can be not only computational more efficient but also memory economic compare to the vanilla bi-level model. We summarize the time and memory requirements for solving bi-level model Eq. (1) vs our NSGame using gradient based methods in Table 1.

5 EXPERIMENTS

In this section, we first conduct experiments on toy example to verify the convergence of TTS-GD and the effectiveness of our proposed NSGame for solving the corresponding bi-level optimization problem. This numerical example serves as proof-of-concept problem and is easy to understand. We then apply NSGame for two realistic applications: class imbalance learning and noisy label learning. Two meta learning models MWN Shu et al. (2019) for class imbalance learning, MLC Wang et al. (2020) for noisy label learning are the base models for our NSGame. We provide a brief overview of each problem and present experimental results and analysis of NSGame compared to MWN and MLC. More detailed experimental settings are provided in the supplementary material (SM).

5.1 TOY EXAMPLE

To verify the convergence of TTS-GD to the optimal solution of the Stackelberg game, we consider

$$\begin{cases} \min_{\boldsymbol{x}\in\mathbb{R}^n} \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}_1\|_2^2 + \frac{1}{2}\|\boldsymbol{y}_2-\boldsymbol{e}\|_2^2 \\ (\boldsymbol{y}_1, \boldsymbol{y}_2) = \arg\min_{\boldsymbol{z}_1\in\mathbb{R}^n, \boldsymbol{z}_2\in\mathbb{R}^n} -\boldsymbol{x}^\top \boldsymbol{z}_1 + \frac{1}{2}\|\boldsymbol{z}_1\|_2^2 + \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{z}_2\|_2^2 \end{cases} \tag{9}$$

where $n = 5$ and $\boldsymbol{e} = (1, 1, 1, 1, 1)^\top$. $\boldsymbol{x}$ is the leader's variable, which aims to minimize the upper-level objective through itself and the follower's strategy $\boldsymbol{y}_1, \boldsymbol{y}_2$. One can easily obtained the optimal solution of Eq. (9) by simple calculation, that is $\boldsymbol{x}^* = \boldsymbol{y}_1^* = \boldsymbol{y}_2^* = (1, 1, 1, 1, 1)^\top$. The problem Eq. (9) satisfies the assumptions in Section 3.2. Then we construct the corresponding NSGame as:

$$(\boldsymbol{P}_1) \quad \min_{\boldsymbol{y}_1\in\mathbb{R}^n} -\boldsymbol{x}^\top \boldsymbol{y}_1 + \frac{1}{2}\|\boldsymbol{y}_1\|_2^2 \qquad (\boldsymbol{P}_2) \quad \begin{cases} \min_{\boldsymbol{x}\in\mathbb{R}^n} \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}_1\|_2^2 + \frac{1}{2}\|\boldsymbol{y}_2-\boldsymbol{e}\|_2^2 \\ s.t.\ \ \boldsymbol{y}_2 = \arg\min_{\boldsymbol{y}_2\in\mathbb{R}^n} \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}_2\|_2^2 \end{cases} \tag{10}$$

where $\boldsymbol{y}_1$ aims to solve the objective of $(\boldsymbol{P}_1)$ given $\boldsymbol{x}$, and $\boldsymbol{x}$ aims to solve the upper-level objective of $(\boldsymbol{P}_2)$ through itself and $\boldsymbol{y}_2$ given $\boldsymbol{y}_1$.

We employ GD to solve Eq. (9) and GD with two-time scale (TTS-GD) learning rate to solve Eq. (10), the experimental results are shown in Fig. 2. In the first row of Fig. 2, we set the learning rate for update of $\boldsymbol{x}$ as 0.005 and learning rate for update $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ as 0.5. We can see that TTS-GD for solving Eq. (10) converges to the optimal solution of Eq. (9) for different initialization, which indicates that one can solve Eq. (9) by optimizing Eq. (10). We reverse the learning rate in the second row of Fig. 2, that is $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ update slow, while $\boldsymbol{x}$ updates fast, one can see that gradient descent for solving NSGame does not converge to the optimal solution of Eq. (9), which indicates that time scale is very important for GD converges to our desired solution as shown in Theorem 4.1.

5.2 CLASS IMBALANCE LEARNING

Class imbalance learning Ren et al. (2018) deals with the problem where the number of samples for each class follows long-tailed distribution, therefore learning on such imbalanced dataset will seriously degenerate the performance of the learned model on test set. In this task, we work on long-tailed version of CIFAR datasets and apply NSGame based on the model MWN Shu et al. (2019) which is presented as a bi-level learning problem. More experimental settings about the data generation and analysis on the problem setting are provided in supplementary material Section B.

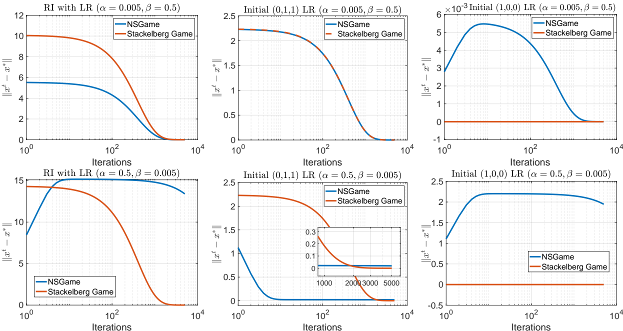

Figure 2: Illustration of the convergence of TTS-GD for solving NSGame Eq. (10) to the optimal solution of Stackelberg game Eq. (9).

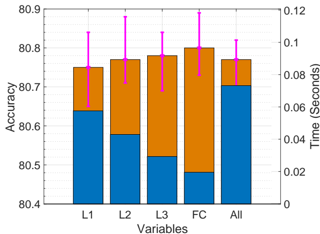

Figure 3: Comparison of different network layers as $\boldsymbol{y}_2$.

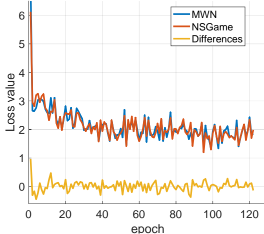

Figure 4: Comparisons of the meta loss.

Ablation study. We take ResNet 32 as our backbone network, to construct NSGame model, and split the network parameters ($\boldsymbol{y}$ in Eq. (1)) into different parts "Layer 1, Layer 2, Layer 3, FC layer", according to the network architecture. We conduct ablation study by setting the parameters of each parts as $\boldsymbol{y}_2$ in Eq. (3). Specifically, if $\boldsymbol{y}_2$ is the parameter of the "FC layer", then $\boldsymbol{y}_1$ is the parameters of the remaining layers. We show in Fig. 3 the performance (orange bar) and computational time (blue bar) of NSGame under these different settings. It can be seen from Fig. 3 that the performances are comparable in all cases, which means that we can take any parts of the network layers to be $\boldsymbol{y}_2$. While the computational time for taken $\boldsymbol{y}_2$ to be different layers are quite different. Since we need to compute second order derivative $D_{31}f_2$ according to Eq. (7), if $\boldsymbol{y}_2$ is the parameters of the "Layer 1", then we need to extend the entire computational graph, therefore costs time and memory. While taking parameters of the "FC layer" as $\boldsymbol{y}_2$ does not need to extend the computational graph too much, therefore it is more efficient than other settings [4].

To verify whether NSGame is a good approximation to the MWN, we plot the meta loss of MWN and NSGame in Fig. 4. It can be seen from Fig. 4 that the meta loss of NSGame is almost identical to MWN especially in the final stage of training, which indicates that our TTS-SGD for solving NSGame follows almost the same optimization path as SGD for MWN and converges to the optimal of MWN. The experimental results echoes the results presented by Theorem 4.1. More analysis are provided in Appendix B.3 and Appendix B.4.

Experiments on CIFAR. We then test NSGame on long-tailed CIFAR 10 and CIFAR 100 Cui et al. (2019) datasets. We follow the same experimental settings as MWN Shu et al. (2019) and compare our results with MWN and another approximated gradient based bi-level optimization method TBP Shaban et al. (2019). In Table 2, we report the top-1 mean accuracy (±std) of MWN, TBP and our

[4] In the follow up applications we take parameters of the "FC layer" as $\boldsymbol{y}_2$ in Eq. (3).

NSGame under class imbalance factor $20, 50, 100$ for 3 repeatitions. WMN[†] indicates that we report our running results using the original code provided by the authors. One can see that the NSGame achieves comparable and even better results than MWN and TBP for all cases in CIFAR 10 and CIFAR 100. Importantly, NSGame is computationally more efficient than MWN and TBP, it takes about 0.07 Seconds for one training iteration of NSGame under ResNet-32 backbone, while MWN costs about 0.15 Seconds and TBP takes about 0.17 Seconds for one training iteration. As for the memory costs, MWN requires 235MB and TBP needs 242MB extra memory for computing the second order derivative, while NSGame requires almost no extra memory for computing the second order derivative, these are consistent with theoretical results in Section 4.2. More experiments on large scale networks are provided in supplementary materials which demonstrate that the advantage of NSGame over MWN is more significant for deeper networks as ResNet-101 and ResNet-152.

5.3 Noisy label learning

We further consider a more practical problem where we learn a robust network from noisy labeled data. Two representative meta learning methods MWN Shu et al. (2019) and MLC Wang et al. (2020) are regarded as our base models, specifically, we apply our NSGame method to solve these two bi-level learning problem respectively. One can easily verify that the models of MWN and MLC for noisy label learning satisfy the assumption of our NSGame (Please refer to SM Section C for experimental settings and more analysis). We use the code provided by MWN and MLC and follow the same experimental settings, and test NSGame on uniform noise and flip noise under different noise ratio. Specifically, we use ResNet32 for MWN and NSGame$_{\mathrm{MWN}}$, and WRN-28-10 network for MLC and NSGame$_{\mathrm{MWN}}$. We also compare NSGame with an approximated gradient method TBP Shaban et al. (2019) for MWN.

Table 2: Test accuracy (%) and costs(s/MB) of ResNet 32 on long-tailed CIFAR-10 and CIFAR-100 datasets.

Dataset	Long-Tailed CIFAR-10			
Imbalance	Costs	100	50	20
MWN[†]	0.152/235	75.35$_{\pm 0.41}$	80.10$_{\pm 0.52}$	84.90$_{\pm 0.34}$
TBP	0.173/242	75.45$_{\pm 0.32}$	80.22$_{\pm 0.43}$	85.07$_{\pm 0.37}$
NSGame	0.068/0.056	75.42$_{\pm 0.22}$	80.30$_{\pm 0.34}$	85.21$_{\pm 0.38}$
Dataset	Long-Tailed CIFAR-100			
MWN[†]	0.164/235	42.01$_{\pm 0.39}$	46.0$_{\pm 0.45}$	53.55$_{\pm 0.33}$
TBP	0.188/241	42.08$_{\pm 0.41}$	46.17$_{\pm 0.36}$	53.6$_{\pm 0.44}$
NSGame	0.071/0.17	41.75$_{\pm 0.19}$	46.34$_{\pm 0.31}$	53.72$_{\pm 0.26}$

Table 3: Test accuracy (%) and computational costs(s/MB) on CIFAR-10 and CIFAR-100 datasets with varying noise rate under different noise type.

Datasets	Methods/Costs		Uniform noise			Flip noise		
	Methods	Costs	0%	20%	40%	0%	20%	40%
CIFAR-10	MWN	0.165/233	91.93$_{\pm 0.23}$	90.36$_{\pm 0.38}$	87.44$_{\pm 0.45}$	91.64$_{\pm 0.26}$	89.81$_{\pm 0.53}$	88.06$_{\pm 0.37}$
	TBP	0.192/242	92.11$_{\pm 0.27}$	90.19$_{\pm 0.33}$	87.15$_{\pm 0.38}$	92.15$_{\pm 0.24}$	90.05$_{\pm 0.46}$	88.12$_{\pm 0.39}$
	NSGame$_{\mathrm{MWN}}$	0.072/0.059	92.03$_{\pm 0.17}$	90.12$_{\pm 0.26}$	87.04$_{\pm 0.34}$	92.26$_{\pm 0.21}$	90.50$_{\pm 0.32}$	87.85$_{\pm 0.42}$
	MLC	0.215/388	90.42$_{\pm 0.36}$	84.91$_{\pm 0.27}$	80.60$_{\pm 0.42}$	90.45$_{\pm 0.33}$	86.62$_{\pm 0.41}$	80.13$_{\pm 0.35}$
	NSGame$_{\mathrm{MLC}}$	0.097/0.02	90.73$_{\pm 0.21}$	85.21$_{\pm 0.25}$	80.45$_{\pm 0.33}$	90.25$_{\pm 0.19}$	87.64$_{\pm 0.28}$	79.57$_{\pm 0.39}$
CIFAR-100	MWN	0.173/233	69.65$_{\pm 0.33}$	63.31$_{\pm 0.41}$	57.63$_{\pm 0.37}$	68.97$_{\pm 0.24}$	64.25$_{\pm 0.39}$	57.53$_{\pm 0.43}$
	TBP	0.201/242	69.77$_{\pm 0.22}$	63.14$_{\pm 0.37}$	58.26$_{\pm 0.39}$	69.12$_{\pm 0.28}$	64.11$_{\pm 0.37}$	57.82$_{\pm 0.44}$
	NSGame$_{\mathrm{MWN}}$	0.075/0.18	69.84$_{\pm 0.15}$	63.08$_{\pm 0.27}$	58.96$_{\pm 0.42}$	69.05$_{\pm 0.23}$	63.94$_{\pm 0.38}$	57.98$_{\pm 0.32}$
	MLC	0.221/388	72.18$_{\pm 0.33}$	66.17$_{\pm 0.39}$	59.34$_{\pm 0.44}$	71.53$_{\pm 0.36}$	65.97$_{\pm 0.29}$	52.26$_{\pm 0.45}$
	NSGame$_{\mathrm{MLC}}$	0.105/0.17	72.20$_{\pm 0.18}$	66.0$_{\pm 0.33}$	59.82$_{\pm 0.37}$	71.83$_{\pm 0.25}$	66.14$_{\pm 0.41}$	53.25$_{\pm 0.44}$

Table 3 reports the test accuracy and computational costs of our methods NSGame$_{\mathrm{MWN}}$ and NSGame$_{\mathrm{MLC}}$ compared with TBP, the base models MWN and MLC. We report the mean accuracy ($\pm$ std) after 3 repetitions. The experimental results show that NSGame is able to achieve comparable performance for both MWN, TBP and MLC, while NSGame is computationally more efficient. NSGame takes about 0.075 Seconds for one iteration to solve MWN, while the original MWN takes 0.17 Seconds and TBP takes 0.19 Seconds. Meanwhile, NSGame takes about 0.1 Seconds for one iteration to solve MLC, the original MLC takes 0.21 Seconds. For the memory costs, MWN, TBP and MLC require extra memory (233MB for MWN, 242MB for TBP and 388 for MLC) for computing the second order derivative, while our NSGame requires almost no extra memory.

6 Conclusion

This paper aims to alleviate the computation bottleneck of bi-level learning methods. Different to the current efforts which improve the computational efficiency of first-order algorithms for solving the bi-level model, we propose a novel NSGame as an alternative to the bi-level learning model. NSGame is the first to untie the hierarchical structure of the bi-level model, it is easy to solve and is guaranteed to be a good alternative to the original bi-level learning model both in theory and in practice. First-order two-time scale method can be readily used to solve NSGame, and it has been verified on representative meta learning applications that NSGame significantly improves the computational efficiency of current bi-level learning methods with same accuracy.

REFERENCES

Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations*, 2018.

Ondrej Bohdal, Yongxin Yang, and Timothy Hospedales. Evograd: Efficient gradient-based meta-learning and hyperparameter optimization. *Advances in Neural Information Processing Systems*, 34:22234–22246, 2021.

Vivek S Borkar. Stochastic approximation with two time scales. *Systems & Control Letters*, 29(5): 291–294, 1997.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *arXiv preprint arXiv:2103.12828*, 2021a.

Yinbo Chen, Zhuang Liu, Huijuan Xu, Trevor Darrell, and Xiaolong Wang. Meta-baseline: exploring simple meta-learning for few-shot learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9062–9071, 2021b.

Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9268–9277, 2019.

Stephan Dempe. *Bilevel optimization: theory, algorithms and applications*, volume 3. 2018.

Tanner Fiez, Benjamin Chasnov, and Lillian Ratliff. Implicit learning dynamics in stackelberg games: Equilibria characterization, convergence analysis, and empirical study. In *International Conference on Machine Learning*, pp. 3133–3144. PMLR, 2020.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 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*, pp. 1568–1577. PMLR, 2018.

Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Meta approach to data augmentation optimization. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2574–2583, 2022.

Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6629–6640, 2017.

Mingyi Hong, Hoi-To Wai, Zhaoran Wang, and Zhuoran Yang. A two-timescale framework for bilevel optimization: Complexity analysis and application to actor-critic. *arXiv preprint arXiv:2007.05170*, 2020.

TM Hospedales, A Antoniou, P Micaelli, and AJ Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Chi Jin, Praneeth Netrapalli, and Michael Jordan. What is local optimality in nonconvex-nonconcave minimax optimization? In *International conference on machine learning*, pp. 4880–4889. PMLR, 2020.

Hae Beom Lee, Taewook Nam, Eunho Yang, and Sung Ju Hwang. Meta dropout: Learning to perturb latent features for generalization. In *International Conference on Learning Representations*, 2019.

Risheng Liu, Jiaxin Gao, Jin Zhang, Deyu Meng, and Zhouchen Lin. Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pp. 1540–1552. PMLR, 2020.

Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. Scalable gradient-based tuning of continuous regularization hyperparameters. In *International conference on machine learning*, pp. 2952–2960. PMLR, 2016.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pp. 2113–2122. PMLR, 2015.

Eric Mazumdar, Lillian J Ratliff, and S Shankar Sastry. On gradient-based learning in continuous games. *SIAM Journal on Mathematics of Data Science*, 2(1):103–131, 2020.

Aditya Krishna Menon, Sadeep Jayasumana, Ankit Singh Rawat, Himanshu Jain, Andreas Veit, and Sanjiv Kumar. Long-tail learning via logit adjustment. In *International Conference on Learning Representations*, 2020.

Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. Making deep neural networks robust to label noise: A loss correction approach. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1944–1952, 2017.

HL Prasad, Prashanth LA, and Shalabh Bhatnagar. Two-timescale algorithms for learning nash equilibria in general-sum stochastic games. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pp. 1371–1379, 2015.

Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in neural information processing systems*, 32, 2019.

Lillian J Ratliff, Samuel A Burden, and S Shankar Sastry. On the characterization of local nash equilibria in continuous games. *IEEE transactions on automatic control*, 61(8):2301–2307, 2016.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *International conference on machine learning*, pp. 4334–4343. PMLR, 2018.

Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1723–1732. PMLR, 2019.

Albert Shaw, Wei Wei, Weiyang Liu, Le Song, and Bo Dai. Meta architecture search. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 11227–11237, 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 *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 1919–1930, 2019.

Jun Shu, Yanwen Zhu, Qian Zhao, Zongben Xu, and Deyu Meng. Mlr-snet: Transferable lr schedules for heterogeneous tasks. *arXiv preprint arXiv:2007.14546*, 2020.

Bradly Stadie, Lunjun Zhang, and Jimmy Ba. Learning intrinsic rewards as a bi-level optimization problem. In *Conference on Uncertainty in Artificial Intelligence*, pp. 111–120. PMLR, 2020.

Y Tian, L Shen, G Su, Z Li, and W Liu. Alphagan: Fully differentiable architecture search for generative adversarial networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

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

Shipeng Wang, Yan Yang, Jian Sun, and Zongben Xu. Variational hyperadam: A meta-learning approach to network training. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (01):1–1, 2021.

Zhen Wang, Guosheng Hu, and Qinghua Hu. Training noise-robust deep neural networks via meta-learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4524–4533, 2020.

Huaxiu Yao, Yu Wang, Ying Wei, Peilin Zhao, Mehrdad Mahdavi, Defu Lian, and Chelsea Finn. Meta-learning with an adaptive task scheduler. *Advances in Neural Information Processing Systems*, 34:7497–7509, 2021.

A Appendix – Proofs and Analysis

A.1 Proofs of the main results

In this part, we provide the proofs of all theoretical results in our main paper.

Proposition 3.1. *If the Stackelberg game Eq.* (1) *has SDSE, then its global optimal must be a SDSE. Furthermore, the Pareto optimal Nash equilibrium of the corresponding NSGame Eq.* (3) *is the global optimal of Eq.* (1)*.*

Proof. We use proof by contradiction. Let $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ be the global optimal of Eq. (1). If $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is not an SDSE, then it implies that either $D_1 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) \neq 0$, $D_2 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) \neq 0$, or the matrix $\begin{bmatrix} D_1^2 f_1 & D_{12} f_1 \\ D_{21} f_1 & D_2^2 f_1 \end{bmatrix}$ has negative eigenvalues. All these results suggests that one can find a direction $(-D_1 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*), -D_2 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*)$ or the vector corresponds to the negative eigenvalues) to decrease the objective function $f_1(\boldsymbol{x}^*, \boldsymbol{y}^*)$, which contradicts with that $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is a global optimal. □

Theorem 3.1. *Let $(\boldsymbol{x}^*, \boldsymbol{y}^*) \in X \times Y$ be a SDSE of Eq.* (1) *with $\boldsymbol{y}^* = (\boldsymbol{y}_1^*, \boldsymbol{y}_2^*) \in Y_1 \times Y_2$, then $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ is a DNE of Eq.* (3)*.*

Proof. According to the Definition 3.1, we have $D_1 f_1 = 0$ and $D\boldsymbol{y}_2^\top D_3 f_1 = 0$ for f_1 at $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$, thus $D_1 f_1 + D\boldsymbol{y}_2^\top D_3 f_1 = 0$. Meanwhile, we have $D_2 f_2(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*) = 0$ and $D_3 f_2(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*) = 0$, therefore $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ is an equilibrium of Eq. (3). According to Eq. (4), we have

$$\begin{bmatrix} D_1^2 f_1 & D_{12} f_1 & D_{13} f_1 \\ D_{21} f_1 & D_2^2 f_1 & D_{23} f_1 \\ D_{31} f_1 & D_{32} f_1 & D_3^2 f_1 \end{bmatrix} \succ 0 \tag{11}$$

for $f_1(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$, thus we can verify that $D_1^2 f_1 + D\boldsymbol{y}_2^\top D_{31} f_1 + D\boldsymbol{y}_2^\top D_{13} f_1 + (D\boldsymbol{y}_2^\top)^2 D_3^2 f_1 \succ 0$ at $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$. Together with $D_2^2 f_2(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*) \succ 0$ and $D_3^2 f_2(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*) \succ 0$, we prove that $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ is a DNE of of Eq. (3). □

Proposition 3.2. *If the Stackelberg game Eq.* (1) *has SDSE, then the pareto optimal Nash equilibrium of the corresponding NSGame Eq.* (3) *is the global optimal of Eq.* (1)*.*

Proof. It is straight forward to prove the results. According to Proposition 3.1 we know that if the Stackelberg game Eq. (1) has SDSE then the global optimal is a SDSE, then according to Theorem 3.1 we know the global optimal must be a Nash equilibrium of Eq. (3). Based on the definition of global optimal the Nash equilibrium that corresponds to the global optimal must be the pareto optimal Nash equilibrium. □

Theorem 3.2. *If the two functions $f_1(\boldsymbol{x}, \boldsymbol{y})$ and $f_2(\boldsymbol{x}, \boldsymbol{y})$ in Eq.* (1) *are coherent, then every DSE of Eq.* (1) *is a SDSE and it is also a DNE of Eq.* (3)*.*

Proof. Since $f_1(\boldsymbol{x}, \boldsymbol{y})$ and $f_2(\boldsymbol{x}, \boldsymbol{y})$ are coherent, then for differential Stackelberg equilibrium (DSE) $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ of Eq. (1), $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ must achieves a local minimum of f_1, which implies that $D_1 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0$, $D_2 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0$ and $\begin{bmatrix} D_1^2 f_1 & D_{12} f_1 \\ D_{21} f_1 & D_2^2 f_1 \end{bmatrix} \succ 0$ at $(\boldsymbol{x}^*, \boldsymbol{y}^*)$. Together with the definition of DSE, then $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is an SDSE. According to Theorem 3.1, we know the DSE of Eq. (1) is DNE of Eq. (3). □

Theorem 3.3. *Assume the above assumptions are satisfied by the Stackelberg game Eq.* (1). *Let $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ be a DNE of Eq.* (3). *If $\boldsymbol{y}^* = (\boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ is a function of $\boldsymbol{x}^*$, that is $\boldsymbol{y}^* = \phi(\boldsymbol{x}^*)$, and $D_2^2 f_2 \succ 0$ for $f_2(\boldsymbol{x}, \boldsymbol{y})$ at $(\boldsymbol{x}^*, \boldsymbol{y}^*)$, then $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ is a SDSE.*

Proof. With assumption 1, assumption 3 and that $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ is a DNE, we know that

$$D_1 f_1(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*) = 0, D_2 f_1(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*) = 0 \text{ and } D_3 f_1(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*) = 0 \tag{12}$$

which indicates that $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ is a critical point of $f_1(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{y}_2)$. The coherent of f_1 and f_2 also implies that

$$\begin{bmatrix} D_1^2 f_1 & D_{13} f_1 \\ D_{31} f_1 & D_3^2 f_1 \end{bmatrix} \succ 0 \tag{13}$$

Meanwhile, as $\boldsymbol{y}^* = \psi(\boldsymbol{x}^*)$ and $D_2^2 f_2(\boldsymbol{x}^*, \boldsymbol{y}^*) \succ 0$, which suggests $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ is a local optimal of $\min_{\boldsymbol{y}_1, \boldsymbol{y}_2} f_2(\boldsymbol{x}^*, \boldsymbol{y}_1, \boldsymbol{y}_2)$. According to assumption 2, we know

$$\begin{bmatrix} D_2^2 f_1 & D_{23} f_1 \\ D_{32} f_1 & D_3^2 f_1 \end{bmatrix} \succ 0 \tag{14}$$

together with Eq. (13) and the separability of f_1 in assumption 1, we have

$$\begin{bmatrix} D_1^2 f_1 & D_{12} f_1 & D_{13} f_1 \\ D_{21} f_1 & D_2^2 f_1 & D_{23} f_1 \\ D_{31} f_1 & D_{32} f_1 & D_3^2 f_1 \end{bmatrix} \succ 0. \tag{15}$$

Combine Eq. (12) with Eq. (12) and the fact $(\boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ is a local optimal of $\min_{\boldsymbol{y}_1, \boldsymbol{y}_2} f_2(\boldsymbol{x}^*, \boldsymbol{y}_1, \boldsymbol{y}_2)$, we know that $(\boldsymbol{x}^*, \boldsymbol{y}_1^*, \boldsymbol{y}_2^*)$ is a SDSE. □

The proof of the Theorem 4.1 is the same as the proof provided by Borkar (1997); Heusel et al. (2017).

A.2 Analysis on the main results

1) Reasons on the introduction of SDSE. The SDSE that we introduce in Section 3 is a special DSE of bi-level Stackelberg game Eq. (1), it is not only important for our main results but also very practical in use. For Eq. (1), its DSE $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ implies that

$$D_1 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0, D\boldsymbol{y}^\top D_2 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0. \tag{16}$$

For lots of real applications such as hyper-parameter learning Shu et al. (2019); Wang et al. (2020), $\boldsymbol{x}$ is only implicit in f_1 through $\boldsymbol{y}$ as

$$\begin{cases} \min\limits_{\boldsymbol{x} \in \mathbb{R}^n} f_1(\boldsymbol{y}) \\ s.t. \;\; \boldsymbol{y} = \arg \min\limits_{\boldsymbol{z} \in \mathbb{R}^m} f_2(\boldsymbol{x}, \boldsymbol{z}) \end{cases} \tag{17}$$

therefore, Eq. (16) implies

$$\frac{\partial f_1}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = 0$$

at $(\boldsymbol{x}^*, \boldsymbol{y}^*)$. While in practice, we aim to find $\boldsymbol{x}^*$ such that $f_1(\boldsymbol{y}^*)$ is minimal with the corresponding $\boldsymbol{y}^* = \arg\min_{\boldsymbol{z}} f_2(\boldsymbol{x}, \boldsymbol{z})$, that is $\boldsymbol{y}^*$ is also the optimal solution of $f_1(\boldsymbol{y}^*)$. Therefore, we are practically interested in $\frac{\partial f_1}{\partial \boldsymbol{y}} = 0$ and $\frac{\partial^2 f_1}{\partial^2 \boldsymbol{y}} \succ 0$, which is exactly why we introduce the definition of strong DSE for general bi-level learning problem Eq. (1).

To guarantee the existence of SDSE, we introduce coherence condition in **??**. The DSE $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ of the bi-level learning problem Eq. (1) implies

$$D_2 f_2(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0, D_2^2 f_2(\boldsymbol{x}^*, \boldsymbol{y}^*) \succ 0.$$

and

$$Df_1(\boldsymbol{x}^*, \boldsymbol{y}^*) = D_1 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) + D\boldsymbol{y}^\top D_2 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0.$$

$$D^2 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) = D_1^2 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) + D\boldsymbol{y}^\top D_{12} f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) + D\boldsymbol{y}^\top D_{21} f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) + (D\boldsymbol{y}^\top)^2 D_2^2 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) \succ 0.$$

Furthermore, if the two functions f_1 and f_2 are coherent and the $\boldsymbol{x}^*$ in the DSE $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is exactly the $\boldsymbol{x}^*$ in **??**. Then we have

$$D_1 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0, D_2 f_1(\boldsymbol{x}^*, \boldsymbol{y}^*) = 0$$

and

$$\begin{bmatrix} D_1^2 f_1 & D_{12} f_1 \\ D_{21} f_1 & D_2^2 f_1 \end{bmatrix} \succ 0,$$

which implies that the DSE of the bi-level learning problem Eq. (1) is SDSE.

2) Applications of SDSE in real bi-level learning problems. The introduction of the coherent condition is to assure that the bi-level learning problem that we consider is well-defined. That is to say the DSE of the bi-level learning problem is SDSE, and $\boldsymbol{y}^*$ obtained by $\boldsymbol{y}^* = \arg\min_{\boldsymbol{z}} f_2(\boldsymbol{x}, \boldsymbol{z})$ is also the optimal solution of $f_1(\boldsymbol{x}^*, \boldsymbol{y}^*)$. *In practice, coherent condition is also mild for a lot of applications, such as noise label learning and class imbalance learning.*

a. SDSE in noisy label learning. In noisy label learning such as MLC Wang et al. (2020), we aims to find the label corrector with which we can learning a network that is robust to the label noise. Specifically, as presented by Patrini et al. (2017), let $T \in [0,1]^{c \times c}$ be the noise transition matrix with its entries $T_{i,j}$ describing the probability of label i being flapped to label j, as $T_{i,j} = P(z = i | z = j)$.

The MLC aims to solve the following bi-level learning problem

$$\begin{cases} \min\limits_{T} -\dfrac{1}{M} \sum\limits_{i=1}^{M} y_i^{meta} \log\left(f(\boldsymbol{x}_i^{meta}; \boldsymbol{\theta})\right) \\ s.t. \quad \boldsymbol{\theta} = \arg\min\limits_{\boldsymbol{\theta}} -\dfrac{1}{N} \sum\limits_{i=1}^{N} y_i^{train} \log\left(Tf(\boldsymbol{x}_i^{train}; \boldsymbol{\theta})\right) \end{cases} \tag{18}$$

where $\boldsymbol{x}_i^{train}$ here is the i-th training sample with its label y_i^{train}, and $\boldsymbol{x}_i^{meta}$ is the i-th meta sample with its label y_i^{meta}. $\boldsymbol{\theta}$ is the parameter of the network. Let's denote by

$$\mathcal{L}^{meta}(\boldsymbol{\theta}) = -\frac{1}{M} \sum_{i=1}^{M} y_i^{meta} \log\left(f(\boldsymbol{x}_i^{meta}; \boldsymbol{\theta})\right)$$

and

$$\mathcal{L}^{train}(T, \boldsymbol{\theta}) = -\frac{1}{N} \sum_{i=1}^{N} y_i^{train} \log\left(f(T\boldsymbol{x}_i^{train}; \boldsymbol{\theta})\right)$$

One can easily find that $\mathcal{L}^{meta}(\boldsymbol{\theta})$ and $\mathcal{L}^{train}(T, \boldsymbol{\theta})$ are coherent according to the following theorem provided by Patrini et al. (2017).

Theorem A.1. *Patrini et al. (2017) Suppose that the noise matrix T is non-singular, given a proper composite loss $\mathcal{L}$, define the forward loss correction as:*

$$\mathcal{L}^{\rightarrow} = \mathcal{L}(Tf(\boldsymbol{x})). \tag{19}$$

Then, the minimizer of the corrected loss under the noise distribution is the same as the minimizer under the clean distribution:

$$\arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}, \tilde{y}} \mathcal{L}(Tf(\boldsymbol{x}; \boldsymbol{\theta})) = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}, y} \mathcal{L}(f(\boldsymbol{x}; \boldsymbol{\theta})) \tag{20}$$

where $\tilde{y}$ denotes that the label is noisy and y denotes the clean label, therefore

$$\mathcal{L}^{meta}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},y}\mathcal{L}(f(\boldsymbol{x};\boldsymbol{\theta}))$$

and

$$\mathcal{L}^{train}(T,\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},\tilde{y}}\mathcal{L}(Tf(\boldsymbol{x};\boldsymbol{\theta}))$$

The Theorem A.1 indicates that given the correct label transition matrix, the minimizer of the $\mathcal{L}^{train}(T,\boldsymbol{\theta})$ w.r.t $\boldsymbol{\theta}$ is exactly the minimizer of $\mathcal{L}^{meta}(\boldsymbol{\theta})$, thus the two objective functions $\mathcal{L}^{meta}(\boldsymbol{\theta})$ and $\mathcal{L}^{train}(T,\boldsymbol{\theta})$ are coherent. In consequence, the DSE $T^*, \boldsymbol{\theta}^*$ is a SDSE of Eq. (18).

b. SDSE in class imbalance learning. Class imbalance learning is very similar to noisy label learning. In class imbalance learning, the number of samples for each class are seriously different and the data follows long-tailed distribution. In the statistic viewpoint, let $P(y|\boldsymbol{x})$ be the probability of sample $\boldsymbol{x}$ with label y, then

$$P(y|\boldsymbol{x}) \propto P(y)\cdot P(\boldsymbol{x}|y). \tag{21}$$

Typically, one minimizes the cross-entropy loss from the training data to obtain a prediction model

$$\mathcal{L}(y, f(\boldsymbol{x},\boldsymbol{\theta})) = -\log y\cdot f(\boldsymbol{x},\boldsymbol{\theta})$$

where $P(\boldsymbol{x}|y) = f(\boldsymbol{x},\boldsymbol{\theta})$.

In class imbalance learning, only a highly skewed prior distribution $\hat{P}(y)$ is available Menon et al. (2020), such that the prediction $\hat{P}(y|\boldsymbol{x})$ is actually

$$\hat{P}(y|\boldsymbol{x}) \propto \hat{P}(y)\cdot P(\boldsymbol{x}|y). \tag{22}$$

Such a prediction highly bias towards the head classes. Assume that the distribution $P(\boldsymbol{x}|y)$ is balanced, the Eq. (22) can be obtained by

$$\hat{P}(y|\boldsymbol{x}) = P(y|\boldsymbol{x})\cdot\frac{\hat{P}(y)}{P(y)} \tag{23}$$

we assume $P(y)$ is the balanced prior distribution. The Eq. (23) tells us that the biased prediction $\hat{P}(y|\boldsymbol{x})$ which is due to the class imbalanced training data, can be estimated from unbiased prediction $P(y|\boldsymbol{x})$ if the unbalanced prior distribution $P(y)$ is available.

Existing method for class imbalance learning methods aim to estimate the distribution $P(y)$ by re-sampling He & Garcia (2009) or re-weighting Ren et al. (2018); Shu et al. (2019) strategies. The prediction $P(y|\boldsymbol{x})$ is generally represented by a deep neural network $f(\boldsymbol{x},\boldsymbol{\theta})$. The rationality of exiting methods is that if given an ideal balanced $P(y)$, one can learn a prediction $f(\boldsymbol{x},\boldsymbol{\theta})$ from the imbalanced dataset using either re-sampling or re-weighting, such that $f(\boldsymbol{x},\boldsymbol{\theta})$ can predict unbiased results. For meta weight net MWN Shu et al. (2019) as

$$\begin{cases} \min\limits_{\Theta} \dfrac{1}{M}\sum\limits_{i=1}^{M}\mathcal{L}_i^{meta}(\boldsymbol{w}^*(\Theta)) \\ s.t.\ \ \boldsymbol{w}^*(\Theta) = \arg\min\limits_{\boldsymbol{w}} \dfrac{1}{N}\sum\limits_{i=1}^{N}\mathcal{V}_i(\mathcal{L}_i^{train}(\boldsymbol{w}),\Theta)\mathcal{L}_i^{train}(\boldsymbol{w}) \end{cases} \tag{24}$$

where $\boldsymbol{w}$ is the parameter of a deep network. The above results indicate that there exits an ideal weighting function $\mathcal{V}_i(\mathcal{L}_i^{train}(\boldsymbol{w}),\Theta^*)$ which is related to $P(y)$, such that with $\mathcal{V}_i(\mathcal{L}_i^{train}(\boldsymbol{w}),\Theta^*)$ the optimal network parameter $\boldsymbol{w}^*(\Theta^*)$ not only minimizes the training loss, but also minimizes the meta loss. These result reveals that the two objectives in MWN are coherent.

B Experimental setup for class imbalance learning

B.1 Datasets

Our experiments on class imbalance learning are conducted on imbalanced CIFAR-10 and CIFAR-100 datasets Cui et al. (2019). The original CIFAR-10 and CIFAR-100 contains 50,000 training images and 10,000 validation images with 10 classes for CIFAR 10 and 100 classes for CIFAR 100. The image size is 32×32. To create class imbalanced version, we randomly remove the training samples by $n_i\mu^i$ for each class, where i is the class index, n_i is the original number of the training samples for the i-th class, and the parameter μ is set as $\mu \in (0,1)$.

Table 4: Comparisons of test accuracy for different learning rate β_t.

β	1e-3	1e-4	1e-5	1e-6
MWN	45.25	46.15	46.07	45.97
NSGame	34.91	43.30	46.21	46.17

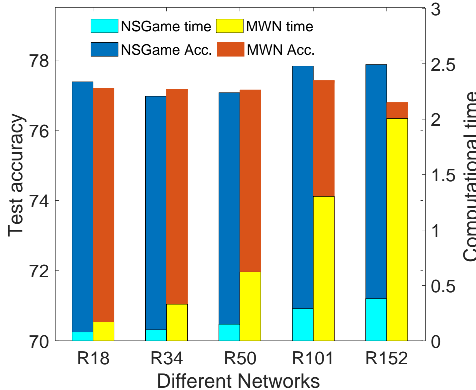

Figure 5: Comparison of the test accuracy and computational time for different networks.

B.2 EXPERIMENTAL SETUP

We conduct our experiments using Pytorch platform, training with one NVIDIA TITAN V GPU. Three imbalanced factors $\{20, 50, 100\}$ are applied on the imbalanced CIFAR-10 and CIFAR-100. We mainly compare our NSGame with MWN, for fair of comparison, all the experimental settings are the same as MWN, including the backbone network ResNet32, the learning rate schedule, total training epochs and the number of meta data. Consider the randomness, we repeat each experiment three times and report the averaged results in Table 2.

B.3 TWO-TIME SCALE ANALYSIS

Two-time scale learning rate schedule is very important for the convergence analysis of our method. In experimental setup, we set the learning rate α_t for the fast update variable to be $\alpha_0 = 1 \times 10^{-1}$ with learning rate schedule the same as MWN. Then based on the fast learning rate, we set the slow learning rate β_0 with different values as given in Table 4. It can be seen from Table 4 that when setting β_0 to be 1×10^{-5} with $\alpha_0 = 1 \times 10^{-1}$, our NSGame achieves the best test accuracy. The result is similar for $\beta_0 = 1 \times 10^{-6}$. While the test accuracy of NSGame degrades significantly as β_0 increases, especially when $\beta_0 = 1 \times 10^{-3}$, which is due to that increase the learning rate of the slow update variable will break the leader-follower relationship in our TTS-SGD algorithm, thus the proposed algorithm will not converges to the SDSE of the MWN model. In contrast to our NSGame, the MWN is robust the meta learning rate, as the leader-follower relationship is constructed by the bi-level model which is independent of the solving algorithm.

B.4 NSGAME FOR LARGE SCALE NETWORK

To further verify the efficiency of our NSGame for large scale model, we conduct experiments on very large scale deep networks such as ResNet-101 and ResNet-152 for class imbalance learning on CIFAR 100 with imbalance ratio 50. We show in Fig. 5 the test accuracy and computational time (per iteration) of NSGame for different networks, “R18” indicates that the backbone network is ResNet-18, similarly for “R34, R50, R101, R152”. It can be seen from Fig. 5 the test accuracy of NSGame and MWN are comparable for different networks, while the computational time of MWN is much longer than NSGame for all networks. Meanwhile, with the increase of the network layers, the computational advantage of NSGame is more significant than MWN. Specifically, for ResNet-18 the computational time of MWN is about 2 times that of NSGame, for ResNet-152 the computational time of MWN is about 5 times that of NSGame.

B.5 PYTORCH CODE FOR CLASS IMBALANCE LEARNING

We provide the pytorch code (main function) of NSGame for MWN, the entire code will be released upon acceptance.

```
optimizer_a = torch.optim.SGD(model.params(), 0.1,
                          momentum=args.momentum, nesterov=args.nesterov,
                          weight_decay=args.weight_decay)

vnet = VNet(1, 100, 1).cuda()

optimizer_c = torch.optim.SGD(vnet.params(), 1e-5,
                          momentum=args.momentum, nesterov=args.nesterov,
                          weight_decay=args.weight_decay)

def train(train_loader, validation_loader,model,
    vnet,optimizer_a,optimizer_c,epoch):
   """Train for one epoch on the training set"""
   model.train()

   for i, (input, target) in enumerate(train_loader):
      input_var = to_var(input, requires_grad=False)
      target_var = to_var(target, requires_grad=False)

      meta_model = build_model()
      meta_model.load_state_dict(model.state_dict())
      y_f_hat = meta_model(input_var)
      cost = F.cross_entropy(y_f_hat, target_var, reduce=False)
      cost_v = torch.reshape(cost, (len(cost), 1))

      v_lambda = vnet(cost_v)
      norm_c = torch.sum(v_lambda)
      if norm_c != 0:
         v_lambda_norm = v_lambda / norm_c
      else:
         v_lambda_norm = v_lambda

      l_f_meta = torch.sum(cost_v * v_lambda_norm)
      meta_model.linear.zero_grad()
      grads = torch.autograd.grad(l_f_meta,
          (meta_model.linear.params()), create_graph=True)
      meta_lr = args.lr * ((0.1 ** int(epoch >= 80)) * (0.1 ** int(epoch
          >= 90)))
      meta_model.linear.update_params(lr_inner=meta_lr,
          source_params=grads)
      del grads

      input_validation, target_validation = next(iter(validation_loader))
      input_validation_var = to_var(input_validation,
          requires_grad=False)
      target_validation_var = to_var(target_validation,
          requires_grad=False)
      y_g_hat = meta_model(input_validation_var)
      l_g_meta = F.cross_entropy(y_g_hat, target_validation_var)
      prec_meta = accuracy(y_g_hat.data, target_validation_var.data,
          topk=(1,))[0]

      optimizer_c.zero_grad()
      l_g_meta.backward()
      optimizer_c.step()

      y_f = model(input_var)
      cost_w = F.cross_entropy(y_f, target_var, reduce=False)
      cost_v = torch.reshape(cost_w, (len(cost_w), 1))
```

```
      prec_train = accuracy(y_f.data, target_var.data, topk=(1,))[0]

      with torch.no_grad():
         w_new = vnet(cost_v)
      norm_v = torch.sum(w_new)
      if norm_v != 0:
         w_v = w_new / norm_v
      else:
         w_v = w_new
      l_f = torch.sum(cost_v * w_v)

      optimizer_a.zero_grad()
      l_f.backward()
      optimizer_a.step()
```

C EXPERIMENTAL SETUP FOR NOISY LABEL LEARNING

C.1 EXPERIMENTAL SETUP

We test the proposed method for noisy label learning on CIFAR-10 and CIFAR-100 datasets with different noise types and noise levels. We conduct experiments on uniform noise and flip noise with noise level $\{0, 20, 40\}$. **(1) Uniform noise.** The label of each sample is transformed to random class with probability p. **(2) Flip noise.** The label of each sample is randomly flipped to similar classes with probability p, as specified in Shu et al. (2019), we select two classes as similar classes in our experiments.

We use ResNet-32 for NSGame$_{\mathrm{MWN}}$ and Shu et al. (2019) as specified in Shu et al. (2019), and use the SGD with momentum 0.9 and learning rate 1×10^{-5} for the meta objective, SGD with momentum 0.9 and initial learning rate 1×10^{-1} for the training objective.1000 images with clean labels in validation set are randomly selected as the meta-data set as in Shu et al. (2019) for MWN and NSGame$_{\mathrm{MWN}}$ For our NSGame$_{\mathrm{MWN}}$, all other settings are the same as Shu et al. (2019). Specifically, we train the entire model with 40 epochs (the learning rate is divided by 10 at the 36 and 38 epochs) for uniform noise and 60 epochs (the learning rate is divided by 10 after 40 and 50 epochs) for flip noise.

We use WRN-28-10 for MLC Wang et al. (2020) and our NSGame$_{\mathrm{MLC}}$. Both the meta objective and training objective of NSGame$_{\mathrm{MLC}}$ are optimized by SGD with learning rate 1×10^{-4} for meta objective and 1×10^{-1} for the training objective. As for the clean meta datasets, we randomly sample 50 clean images for each class for CIFAR-10 and 5 clean images for each class for CIFAR-100. All the training setups of MLC and NSGame$_{\mathrm{MLC}}$ are the same in our implements expect for the learning rate and optimizer setting, which is due to our two-time scale SGD algorithm. We provide the pytorch code of NSGame$_{\mathrm{MWN}}$ and NSGame$_{\mathrm{MLC}}$.

1) Pytorch code for NSGame$_{\mathrm{MWN}}$

```
optimizer_model = torch.optim.SGD(model.params(), args.lr,
                          momentum=args.momentum,
                              weight_decay=args.weight_decay)
optimizer_vnet = torch.optim.SGD(vnet.params(), 1e-5,
    momentum=args.momentum,
                      weight_decay=1e-4)

def train(train_loader,train_meta_loader,model,
    vnet,optimizer_model,optimizer_vnet,epoch):
   print('\nEpoch: %d' % epoch)

   train_loss = 0
   meta_loss = 0

   train_meta_loader_iter = iter(train_meta_loader)
   for batch_idx, (inputs, targets) in enumerate(train_loader):
      model.train()
```

```
        inputs, targets = inputs.to(device), targets.to(device)
        meta_model = build_model().cuda()
        meta_model.load_state_dict(model.state_dict())
        outputs = meta_model(inputs)

        cost = F.cross_entropy(outputs, targets, reduce=False)
        cost_v = torch.reshape(cost, (len(cost), 1))
        v_lambda = vnet(cost_v.data)
        l_f_meta = torch.sum(cost_v * v_lambda)/len(cost_v)
        meta_model.linear.zero_grad()
        grads = torch.autograd.grad(l_f_meta,
            (meta_model.linear.params()), create_graph=True)
        meta_lr = args.lr * ((0.1 ** int(epoch >= 40)) * (0.1 ** int(epoch
            >= 50)))
        meta_model.linear.update_params(lr_inner=meta_lr,
            source_params=grads)
        del grads

        try:
           inputs_val, targets_val = next(train_meta_loader_iter)
        except StopIteration:
           train_meta_loader_iter = iter(train_meta_loader)
           inputs_val, targets_val = next(train_meta_loader_iter)
        inputs_val, targets_val = inputs_val.to(device),
            targets_val.to(device)
        y_g_hat = meta_model(inputs_val)
        l_g_meta = F.cross_entropy(y_g_hat, targets_val)
        prec_meta = accuracy(y_g_hat.data, targets_val.data, topk=(1,))[0]


        optimizer_vnet.zero_grad()
        l_g_meta.backward()
        optimizer_vnet.step()

        outputs = model(inputs)
        cost_w = F.cross_entropy(outputs, targets, reduce=False)
        cost_v = torch.reshape(cost_w, (len(cost_w), 1))
        prec_train = accuracy(outputs.data, targets.data, topk=(1,))[0]

        with torch.no_grad():
           w_new = vnet(cost_v)

        loss = torch.sum(cost_v * w_new)/len(cost_v)

        optimizer_model.zero_grad()
        loss.backward()
        optimizer_model.step()
```

2) Pytorch code for NSGame$_{\text{MLC}}$

```
def train_MLC(Val_choose, train_datas, train_lables, test_datas,
    test_lables, type, ratio):
   main_model= build_model()
   optimizer = torch.optim.SGD(main_model.params(), lr=args.lr_2,
       momentum=args.momentum, weight_decay=args.weight_decay)

   if (type==1):
      type = 'uniform'
   elif (type==2):
      type = 'flip'
   else:
      type = 'flip_to_one'
   ########## load data ###########################################
```

```
print('noise ratio is '+ str(ratio)+'%')
train_data = Cifar10_Dataset(True, Val_choose, train_datas,
    train_lables, transform, target_transform, noise_type=type,
    noisy_ratio=ratio)
print('size of train_data:{}'.format(train_data.__len__()))
test_data = Cifar10_Dataset(False, Val_choose, test_datas,
    test_lables, transform, target_transform)
print('size of test_data:{}'.format(test_data.__len__()))
train_loader = Data.DataLoader(dataset=train_data,
    batch_size=args.batch_size, shuffle=True)
test_loader = Data.DataLoader(dataset=test_data,
    batch_size=args.batch_size, shuffle=True)

model_root = './Cifar10_warmup_CE_'+str(type)+str(ratio)+'.pth' #
    warmup model using pretrain

main_model.load_state_dict(torch.load(model_root))
## load small validation set

X_Val, Y_Val =Cifar10_Val_1(Val_choose, 50)
print('size of val_data:{}'.format(len(X_Val)))
Y_Val = Y_Val.int()
Y_Val = Y_Val.long()
X_Val = to_var(X_Val, requires_grad=False)
Y_Val = to_var(Y_Val, requires_grad=False)

plot_step = 100
for i in tqdm(range(50000)):
   main_model.train()
   image, labels = next(iter(train_loader))
   meta_net = Wide_ResNet(40, 2, 10)
   meta_net.load_state_dict(main_model.state_dict())
   if torch.cuda.is_available():
      meta_net.cuda()

   image = to_var(image, requires_grad=False)
   labels = to_var(labels, requires_grad=False)
   T = to_var(torch.eye(10, 10))
   y_f_hat = meta_net(image)
   pre2 = torch.mm(y_f_hat, T)
   l_f_meta = torch.sum(F.cross_entropy(pre2,labels, reduce=False))
   meta_net.linear.zero_grad()

   grads = torch.autograd.grad(l_f_meta, (meta_net.linear.params()),
       create_graph=True)
   meta_net.linear.update_params(1e-3, source_params=grads)
   y_g_hat = meta_net(X_Val)
   l_g_meta = F.cross_entropy(y_g_hat,Y_Val)
   grad_eps = torch.autograd.grad(l_g_meta, T, only_inputs=True)[0]
   T = torch.clamp(T-0.11*grad_eps,min=0)
   norm_c = torch.sum(T, 0)

   for j in range(10):
      if norm_c[j] != 0:
         T[:, j] /= norm_c[j]

   y_f_hat = main_model(image)
   pre2 = torch.mm(y_f_hat, T)

   l_f = torch.sum(F.cross_entropy(pre2,labels, reduce=False))

   optimizer.zero_grad()
   l_f.backward()
   optimizer.step()
```