Hierarchical Meta-Learning with Hyper-Tasks for Few-Shot Learning

Yunchuan Guan
hustgyc@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

Yu Liu*
liu_yu@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

Ke Zhou
zhke@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

Junyuan Huang
jyhuang@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

ABSTRACT

Meta-learning excels in few-shot learning by extracting shared knowledge from the observed tasks. However, it needs the tasks to adhere to the i.i.d. constraint, which is challenging to achieve due to complex task relationships between data content. Current methods that create tasks in a one-dimensional structure and use meta-learning to learn all tasks flatly struggle with extracting shared knowledge from tasks with overlapping concepts. To address this issue, we propose further constructing tasks from the same environment into hyper-tasks. Since the distributions of hyper-tasks and tasks in a hyper-task can both be approximated as i.i.d. due to further summarization, the meta-learning algorithm can capture shared knowledge more efficiently. Based on the hyper-task, we propose a hierarchical meta-learning paradigm to meta-learn the meta-learning algorithm. The paradigm builds a customized meta-learner for each hyper-task, which makes meta-learners more flexible and expressive. We apply the paradigm to three classic meta-learning algorithms and conduct extensive experiments on public datasets, which confirm the superiority of hierarchical meta-learning in the few-shot learning setting. The code is released at https://github.com/tuantuange/H-meta-learning.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning**.

KEYWORDS

Few-shot Learning; Hyper-Tasks; Meta-Learning

ACM Reference Format:
Yunchuan Guan, Yu Liu, Ke Zhou, and Junyuan Huang. 2023. Hierarchical Meta-Learning with Hyper-Tasks for Few-Shot Learning. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583780.3614911

*Corresponding author.

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3614911

Figure 1: The tasks and hyper-tasks constructed by the images in different domains.

1 INTRODUCTION

One of the main challenges in machine learning is ensuring the generalization performance of models. In the past decade, meta-learning has proven to be effective in improving generalization ability and enhancing few-shot learning. By learning to learn, current meta-learning algorithms extract shared knowledge from observed tasks, enabling good performance on target tasks in terms of validation performance, learning speed, and model robustness [5].

However, achieving good performance with meta-learning relies on the distribution of the tasks adhering to the independent and identically distributed (i.i.d.) assumption in an environment [7]. Recently, researchers begin to classify images in multiple environments, *i.e.*, in different domains or different periods [12, 18]. This background weakens the benefits of meta-learning since the complexity of the task relationships of multiple environments makes it difficult to construct i.i.d. tasks. For example, the six images can be constructed into tasks based on two domains, *i.e.*, realistic and cartoon images. For the tasks of classifying different species under a certain style, we show their construction in blue frames in Figure 1. Obviously, learning to classify species is not independent of learning to classify different image styles and the tasks in hyper-task 1 do not share the same distribution with those in hyper-task 2. However, ordinary meta-learning algorithms learn all six tasks by a shared meta-learner while ignoring the relationship among task 1, task 2, and task 3 and the difference between tasks in hyper-task 1 and hyper-task 2.

There are two kinds of methods for solving this problem. The first kind of method [1, 14] designs weights or schedulers for tasks, thus tuning the impact of different tasks on shared knowledge and improving the generalization capacity of the meta-model to unseen

tasks. The second kind of method [22] learns a neural network with a hierarchical structure to express dependencies between tasks and tailor the shared knowledge explicitly to different clusters of tasks. However, the relationship between the tasks may be implicit, making it difficult to express them explicitly by weighting tasks or constructing hierarchical networks. Therefore, we believe that a non-empirical solution that implicitly learns the shared knowledge of the non-i.i.d. task is still necessary.

According to the tasks shown in Figure 1, we believe that further constructing tasks from the same environment into hyper-tasks based on image style can make the tasks in each hyper-task to be i.i.d., and that further meta-learning the meta-learning algorithm on the hyper-task can enable environment-customized meta-learner. Within each hyper-task, the customized meta-learner can extract the shared knowledge implicitly because the tasks here are i.i.d. Note that we can approximately consider a hyper-task as an environment. The distinction between the two concepts is that the environment is an inherent attribute of the dataset and represents the distribution of a group of similar tasks, while the hyper-task is a collection of tasks constructed based on the environment.

Based on the idea of meta-learning the meta-learning algorithm, we propose a hierarchical meta-learning paradigm that enables two-order adaptation on hyper-tasks and tasks. First, we construct hyper-tasks based on tasks from the same environment. Then, we set up a hyper-meta-learner, meta-learners, and base-learners for hyper-tasks, tasks, and samples, respectively, and recursively learn these models. After all samples in a task are learned, the base-learner parameter update is completed, *i.e.*, the second-order adaptation is completed. After all tasks in a hyper-task are learned, the meta-learner parameter update is completed, *i.e.*, the first-order adaptation is completed. After a batch of hyper-tasks is learned, the hyper-meta-learner parameter update is completed, *i.e.*, an episode of training. Roughly speaking, the second-order adaptation is learning a task-customized base-learner and the first-order adaptation is learning an environment-customized meta-learner. We apply this hierarchical paradigm to three classic meta-learning algorithms and conduct extensive experiments on public datasets. The results confirm the superiority of hierarchical meta-learning in few-shot learning.

The main contributions of this paper are summarized as follows:

- We propose a method for constructing hierarchical structure tasks and achieving accurate knowledge extraction for non-i.i.d. tasks.
- We propose a new meta-learning paradigm that learns hierarchical structure tasks and provides environment-customized meta-learners.
- Extensive experiments on public datasets confirm that our approach not only outperforms classical meta-learning algorithms but also improves the performance of few-shot learning.

2 RELATED WORK

Meta Learning. Ordinary meta-learning algorithms [4, 5, 13] assume all tasks are drawn i.i.d. from the same environment. When the assumption does not hold, researchers deduce the consequences in terms of generalization error. Simeone *et al.* [7] conclude that the generalization error of the meta-learning algorithm is bounded by the factor ϵ which describes the average difference in the distribution of any two observed tasks. In addition, Fallah *et al.* [3] conclude that the generalization error of the meta-learning algorithm is bounded by term $D(p_{m+1}, \{p_i\}_{i=1}^m)$ which describes the average difference between the distribution of the target task and the observed tasks. To address this issue, Collins *et al.* [1] enable task-robust meta-learning by assigning trainable weights to unbalanced tasks. Jiang *et al.* [6] propose Conditional class-Aware Meta-Learning (CAML) to make explicit use of the label structure to inform the model to reshape the representation landscape in a manner that incorporates a global sense of class structure. Liu *et al.* [12] propose to leverage MAML [4] to learn a meta-representation and a meta-adapter to adapt to evolving tasks and overcome catastrophic forgetting. Inspired by the way human beings organize knowledge, Yao *et al.* [22] propose Hierarchically Structured Meta-learning (HSML) to learn the hierarchical relationships of different task features to achieve knowledge customization for different task clusters. These methods are modifications of the ordinary meta-learning approach, either through empirical or manual means, but do not fully utilize the properties of meta-learning itself. In this paper, we use the paradigm of meta-learning to complete few-shot learning, enabling fast adaptation to tasks from multiple environments.

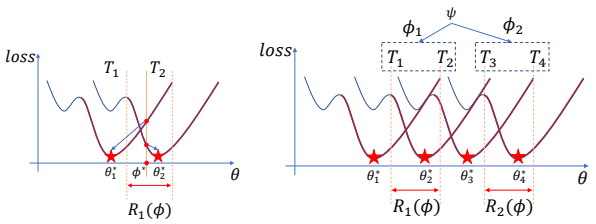
Multi-source domains adaptation. Multi-Source Domain Adaptation considers the domain adaptation problem when the source contains domains with a variety of styles. Yang *et al.* [21] pioneered this problem by adaptively picking the best among a set of hypotheses learned for different source domains. Several methods have been proposed after the introduction of deep learning. Some of these align domains pair-wise. Xu *et al.* [20] use a discriminator to align each source domain with the target domain, while Peng *et al.* [15] match moments across all pairs of source and target domains. These methods learn one classifier per domain and use their weighted combination to predict the class of target samples. Li *et al.* [11] propose a meta-learning technique to search the best initial conditions for multi-source domain adaptation. Unlike the scenario we assumed, the multi-source in multi-source domain adaptation refers to sample distributions rather than task distributions.

3 PRELIMINARY AND MOTIVATION

Single-Task Learning. Let $z_i = (x_i, y_i) \sim D$ be the sample in domain Z with $x_i \in X$ being the the input and $y_i \in Y$ being its corresponding label. We use loss function $l(\theta, z)$ to evaluate the performance of the learner parameterized by θ. In single-task learning, a learning algorithm $\mathcal{A}$ learns from the training set $D_{tr} = \{z_i\}_{i=1}^n$ drawn i.i.d. from D and selects the parameter $\vec{\theta} = \mathcal{A}(D_{tr})$ that minimize

$$loss = \sum_{z_i \in D_{tr}} l(\theta, z_i). \tag{1}$$

Meta Learning. In the context of meta-learning, the task is defined as a dataset and loss function, while the loss function is always shared for different tasks [5]. Therefore, we consider the task as a collection of samples. Let T_i be the task drawn i.i.d. from the environment τ. In the common setting, we further split the T_i into a support set S_i and a query set Q_i. For a specific task, we use the



(a) Two tasks (b) Four tasks

Figure 2: Landscape of two and four tasks. The horizontal coordinate represents the parameter space (in the case of one-dimensional parameters), and the vertical coordinate represents the population loss. The pentagrams in the figure represent the optimal parameters for each task. The red part of the curve is the feasible region where the optimal parameter can be reached by a few gradient descents. Interval $R_1(\phi)$ is the intersection of feasible regions of T_1 and T_2, while interval $R_2(\phi)$ is the intersection of feasible regions of T_3 and T_4.

loss function

$$\begin{aligned} \boldsymbol{L}(\phi, T_i) &= \boldsymbol{L}(\phi(S_i), Q_i) \\ &= \sum_{z_{ij} \in Q_i} l(\phi(S_i), z_{ij}) \end{aligned} \tag{2}$$

to evaluate the performance of the meta-learner parameterized by ϕ. Where $\phi(.)$[1] is an adaptation function that takes the support set S_i as input and outputs a task-customized base-learner θ_i. An ordinary meta-learning algorithm $\boldsymbol{\mathcal{A}}$ learns from the training task set $\boldsymbol{D_{tr}} = \{T_i\}_{i=1}^n$ (meta-training set) drawn i.i.d. from τ and selects the parameter $\hat{\phi} = \boldsymbol{\mathcal{A}}(\boldsymbol{D_{tr}})$ that minimize the empirical error

$$\boldsymbol{Loss} = \sum_{T_i \in \boldsymbol{D_{tr}}} \boldsymbol{L}(\phi, T_i). \tag{3}$$

Typically[2], we use the approach of gradient descent to optimize $\boldsymbol{Loss}$, thus the meta-learner ϕ is optimized by equation

$$\phi = \phi - \beta \nabla \boldsymbol{Loss}, \tag{4}$$

where β is the learning rate of the meta-learner.

In the single-task learning setting, we assume the observed samples are drawn i.i.d. from a sample distribution D, but this assumption does not hold in the meta-learning context, *i.e.*, the multi-task scenario. In the ordinary meta-learning setting, all the tasks T_i are drawn i.i.d. from an environment τ, but the samples in different tasks may be drawn from different sample distributions. As a result, as shown in Figure 2(a), a single-task learning algorithm cannot find a parameter θ^* which perform well on all tasks, cause $\{\theta_1{}^*\} \cap \{\theta_2{}^*\} = \emptyset$. In contrast, a meta-learning algorithm can learn a function that enables fast adaptation on all unseen tasks. Taking MAML as an example, it manages to find an initial parameter ϕ^* from feasible region R_1. With this initial parameter, the model can reach each task's optimal parameter θ_i^* by a few steps of gradient descent. Now, if the tasks are drawn from multiple environments, meta-learning algorithms would meet the same dilemma as single-task learning algorithms. As shown in Figure 2(b), let $T_1, T_2 \sim \tau_1$ and $T_3, T_4 \sim \tau_2$, which means some tasks are drawn from the same environment and some are not. In this scenario, MAML can find an initial parameter ϕ_1 for T_1, T_2 in the feasible region R_1 and an initial parameter ϕ_2 for T_3, T_4 in the feasible region R_2. However, since $R_1 \cap R_2 = \emptyset$, MAML cannot find a common initial parameter to achieve the best performance on all four tasks. Meta-learning addresses the i.i.d. constraint at the sample level through two hierarchical approaches. The first involves organizing samples into tasks. The second uses hierarchical learning, which involves bi-level optimization. We suggest that a more extensive hierarchical organization of tasks and meta-learning algorithms can help alleviate the i.i.d. constraint at the task level.

4 THE PROPOSE FRAMEWORK

4.1 Hyper-task

In this paper, we extend the concept of tasks. We believe that tasks can not only be composed of samples but also of tasks from the same environment. For an unseen task, meta-learning algorithms enable well performance on the query set Q with the support set S in the task. In the same way, for unseen environments, we aim to use hierarchical meta-learning algorithms to achieve good performance on the query task set T^Q, using the labeled support task set T^S in the hyper-task. The hyper-task HT consists of T^Q and T^S. The support task set T^S and query task set T^Q are composed of a series of tasks that are sampled from the same environment. As shown in the left part of Figure 3, a hyper-task has three layers, where each hyper-task is composed of multiple tasks and each task is composed of multiple samples.

4.2 Hierarchical meta-learning

To adapt to multi-environment scenarios, it's essential to construct environment-customized meta-learners capable of fast adaptation to a specific environment. However, current meta-learning algorithms construct task-customized base-learners by adapting to the support set S, while sharing the same meta-learner at the environment level. To address this issue, we propose to meta-learn the meta-learning algorithm. Specifically, we learn a meta-representation (*e.g.*, initialization parameter, optimization schedule, or embedding space) [5] for environments' meta-learners. This allows us to construct environment-customized meta-learners by adapting on the support task set T^S.

As shown in Figure 3, we leverage a hyper-meta-learner ψ to learn the meta-representation. ψ takes support task set T^S as input for adaptation and outputs environment-customized meta-learner

$$\phi_i = \psi(T^S). \tag{5}$$

Different meta-representation correspond to different adaptation function $\psi(T^S)$, we will detail the form of $\psi(T^S)$ in Section 4.3. Similar to the meta-learning paradigm, for a hyper-task, we use the

[1]For simplicity, we use $\phi(.)$ to refer to the function parameterized by the weight ϕ and the same for $\theta(.)$ and $\psi(.)$ mentioned in 4.

[2]We use bold letters $\boldsymbol{\mathcal{A}}, \boldsymbol{D_{tr}}, \boldsymbol{Loss}$ to denote the objects of meta-learning, and $\mathcal{A}, D_{tr}, loss$ to denote the objects of single-task learning.

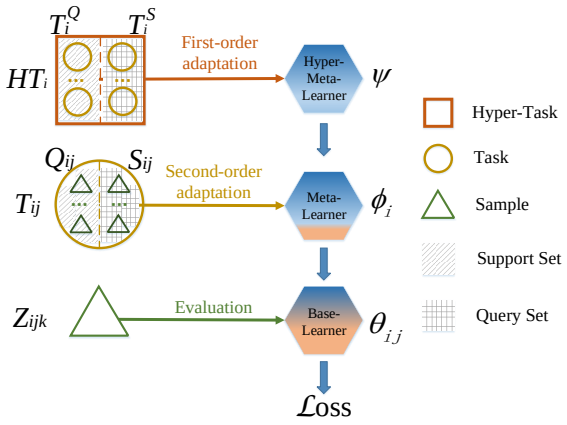


Figure 3: Overview of hierarchical meta-learning.

loss function

$$\mathcal{L}(\psi, HT_i) = \mathcal{L}(\psi(T_i^S), T_i^Q) = \sum_{T_{ij} \in T_i^Q} \boldsymbol{L}(\psi(T_i^S), T_{ij}) \tag{6}$$

to evaluate the performance of the hyper-meta-learner ψ.

Training phase. During the training phase, a hierarchical meta-learning algorithm learns from the training hyper-task set $\mathcal{D}_{tr} = \{HT_i\}_{i=1}^n$ and selects the parameter $\bar{\psi}$ that minimizes empirical error

$$\mathcal{L}oss = \sum_{HT_i \in \mathcal{D}_{tr}} \mathcal{L}(\psi, HT_i). \tag{7}$$

Similar to most meta-learning algorithms, we use a gradient descent-based approach to optimize $\mathcal{L}oss$. The hyper-meta-learner ψ is optimized by equation

$$\phi = \phi - \eta \nabla \mathcal{L}oss, \tag{8}$$

where η is the learning rate of the hyper-meta-learner. The full algorithm is outlined in Algorithm 1. Substitute Equation 6 into Equation 7 for $\mathcal{L}$, we acquire

$$\mathcal{L}oss = \sum_{\{T_i^S, T_i^Q\} \in \mathcal{D}_{tr}} \sum_{T_{ij} \in T_i^Q} \boldsymbol{L}(\psi(T_i^S), T_{ij}). \tag{9}$$

Note that we replace $\{T_i^S, T_i^Q\}$ with HT_i. Introduce index k to Equation 2 and substitute Equation 2 into Equation 9 for $\boldsymbol{L}$, we acquire

$$\mathcal{L}oss = \sum_{\{T_i^S, T_i^Q\} \in \mathcal{D}_{tr}} \sum_{\{S_{ij}, Q_{ij}\} \in T_i^Q} \sum_{z_{ijk} \in Q_{ij}} l(\psi(T_i^S)(S_{ij}), z_{ijk}). \tag{10}$$

Note that after reorganizing Equation 2 by introducing index k, we replace ϕ_i with $\psi(T_i^S)$ and $\{S_{ij}, Q_{ij}\}$ with T_{ij} in Equation 9.

From Equation 10, we can find that optimizing $\mathcal{L}oss$ essentially optimizes a two-order adaptation process $\psi(T_i^S)(S_{ij})$. As shown in Figure 3, the first-order adaptation $\psi(T_i^S)$ takes support task set T_i^S (within the hyper-task HT_i) as input and outputs an environment-customized meta-learner ϕ_i. By optimizing the loss of the first-order adaptation that evaluates on the hold-out query task set T_i^Q, the hyper-meta-learner ψ extracts environment-level shared knowledge. The second-order adaptation $\phi_i(S_{ij})$ takes support set S_{ij} (within the tasks from T_i^Q) as input and outputs a task-customized base-learner θ_{ij}. By optimizing the loss of the second-order adaptation that evaluates on the hold-out query set Q_{ij} (within the tasks from T_i^Q), the meta-learner ϕ_i extracts task-level shared knowledge from a specific environment. We will provide details of the two-order adaptation in Section 4.3.

Algorithm 1 Hierarchical meta-learning.

Require $E = \{\tau_1, \tau_2, ...\tau_k\}$: multiple environments
Require η : learning rate
Randomly initialize hyper-meta-learner ψ
while not done **do**
  Select candidate environments $\tau_i \in E$
  for all τ_i **do**
    Sample multiple tasks from τ_i and construct T_i^S and T_i^Q
    Construct hyper-task $HT_i = \{T_i^S, T_i^Q\} \in \mathcal{D}_{tr}$
  end for
  Construct batch of hyper-tasks $\{HT_1, HT_2, ...\}$
  Set $\mathcal{L}oss = 0$
  for all HT_i **do**
    Calculate $\mathcal{L}(\psi, HT_i)$
    $\mathcal{L}oss+ = \mathcal{L}(\psi, HT_i)$
  end for
  $\psi = \psi - \eta \nabla \mathcal{L}oss$
end while

Testing phase. The prediction process of hierarchical meta-learning is exactly the two-order adaptation. During the testing phase, when dealing with an unseen environment, the hierarchical meta-learning algorithm draws a few tasks (*i.e.*, support task set) from that environment for the first-order adaptation, and obtains an environmental-customized meta-learner ϕ. Then, when dealing with an unseen task in the unseen environment, it draws a few samples (*i.e.*, support set) from that task for the second-order adaptation, and obtains a task-customized base-learner θ. If the hyper-meta-learner ψ is well-trained, θ would be an unbiased predictor for that unseen task in the unseen environment.

4.3 Implementations

The three elements of a meta-learning algorithm are meta-objective, meta-optimization, and meta-representation [5]. According to Equation 10, to implement a hierarchical meta-learning algorithm is to define a function $l(\psi(T_i^S)(S_{ij}), z_{ijk})$ by choosing different meta-objectives, meta-optimizations, and meta-representations. Since hierarchical meta-learning involves three levels of learning, for efficiency reasons, this paper focuses on gradient-based meta-optimization. Also, since the scenario of this paper is few-shot learning, we set the meta-objective as fast adaptation. In this section, we choose MAML and ProtoNet as base meta-learning algorithms and set the meta-representation as initialization parameters. In addition, we explore the implementation of the hierarchical version ANIL (a variant of MAML) from the perspective of feature reuse [16].

4.3.1 Hierarchical MAML. According to the principle of learning initialization parameters for MAML and taking one-step gradient descent adaptation, for example, the first-order adaptation $\psi(T_i^S)$

Algorithm 2 Hierarchical MAML $\mathcal{L}oss$

1: **Require** A batch of hyper-tasks $\{HT_1, HT_2, ...\}$
2: **Require** Learning rates of first and second adaptations α, β
3: **Require** Hyper-meta-learner parameter by ψ
4: **for all** HT_i **do**
5: $\phi_i = \psi$
6: **for all** $\{S_{ij}, Q_{ij}\} \in T_i^S$ **do**
7: $\theta_{ij} = \phi_i$
8: $\theta_{ij} = \theta_{ij} - \alpha\nabla \sum_{z_{ijk} \in S_{ij}} l(\theta_{ij}, z_{ijk})$
9: Calculate loss on the query set: $loss_{ij} = \sum_{z_{ijk} \in Q_{ij}} l(\theta_{ij}, z_{ijk})$
10: **end for**
11: $\phi_i = \phi_i - \beta\nabla \sum loss_{ij}$
12: **for all** $\{S_{ij}, Q_{ij}\} \in T_i^Q$ **do**
13: $\theta_{ij} = \phi_i$
14: $\theta_{ij} = \theta_{ij} - \alpha\nabla \sum_{z_{ijk} \in S_{ij}} l(\theta_{ij}, z_{ijk})$
15: Calculate loss on the query set: $loss_{ij} = \sum_{z_{ijk} \in Q_{ij}} l(\theta_{ij}, z_{ijk})$
16: **end for**
17: Calculate loss on the query task set: $\boldsymbol{Loss_i} = \sum loss_{ij}$
18: **end for**
19: Calculate loss on hyper-task batch: $\mathcal{L}oss = \sum \boldsymbol{Loss_i}$

can be written as

$$\begin{aligned} &\phi_i = \psi, \\ &\phi_i = \phi_i - \beta\nabla \sum_{\{S_{ij}, Q_{ij}\} \in T_i^S} \boldsymbol{L}(\phi_i, T_{ij}), \end{aligned} \tag{11}$$

where the first equation involves the cloning[3] of hyper-meta-learner parameters to each meta-learner and β in the second equation is the learning rate of the first-order adaptation. According to the principle of MAML and taking one step gradient descent adaptation for example, the second-order adaptation $\phi_i(S_{ij})$, can be written as

$$\begin{aligned} &\theta_{ij} = \phi_i, \\ &\theta_{ij} = \theta_{ij} - \alpha\nabla \sum_{z_{ijk} \in S_{ij}} l(\theta_{ij}, z_{ijk}), \end{aligned} \tag{12}$$

where the first equation involves the cloning of meta-learner parameters to each base-learner and α in the second equation is the learning rate of the second-order adaptation. Taking regression problem as an example, loss function $l(.)$ can be written as

$$l(\theta_{ij}, z_{ijk}) = \|\theta_{ij}(x_{ijk}) - y_{ijk}\|_2^2, \tag{13}$$

where $z_{ijk} = (x_{ijk}, y_{ijk})$ and $\theta_{ij}(x_{ijk})$ represents the straightforward output of network θ_{ij}. The calculation process of Equation 10 is outlined in Algorithm 2.

4.3.2 Hierarchical ProtoNet. According to the principle of learning initialization parameters for ProtoNet, the first-order adaptation $\psi(T^S)$ is the same as Equation 11. According to the principle of ProtoNet, the second-order adaptation is expressed implicitly in the prototypes $c_{h_{ij}}$:

$$\begin{aligned} &\theta_i = \phi_i, \\ &c_{h_{ij}} = \frac{1}{N_C} \sum_{\{x_{ijk}, y_{ijk}\} \in S_{h_{ij}}} \theta_i(x_{ijk}), \end{aligned} \tag{14}$$

[3]A function in pytorh, see https://pytorch.org/docs/stable/generated/torch.clone.html.

where h represents the index of classes, N_C represents the number of classes, $c_{h_{ij}}$ and $S_{h_{ij}}$ represent the prototype and support set in the h-th class of the j-th task of i-th hyper-task. Note that $\phi_i(x_{ijk})$ is different from $\phi_i(S_{ij})$ used in MAML, its the straightforward output of network ϕ_i. Taking classification problem as an example, loss function $l(.)$ can be written as

$$\begin{aligned} l(\theta_i, c_{h_{ij}}, z_{ijk}) = \sum_{h=1}^{N_C} \Bigg(& d\big(\theta_i(x_{ijk}), c_{h_{ij}}\big) \\ & + \log \sum_{h'} \exp\Big(- d\big(\theta_i(x_{ijk}), c_{h'_{ij}}\big)\Big)\Bigg), \end{aligned} \tag{15}$$

where $z = (x, y)$ and $d(.)$ represents a distance metric function. Note that if x do not belong to the h-th class, $d(\phi(x), c_h) = 0$. The calculation process of Equation 10 is similar to Algorithm 2, the difference is that we replaced lines 8 and 14 with $c_{h_{ij}} = \frac{1}{N_C} \sum_{\{x_{ijk}, y_{ijk}\} \in S_{h_{ij}}} \phi_i(x_{ijk})$ and lines 9 and 15 with Equation 15.

4.3.3 Hierarchical ANIL. ANIL is a variant of MAML, which implements meta-learning from the perspective of feature reuse rather than fast adaptation. ANIL divides the network into a "head" and a "body", where the head is responsible for feature reuse, and the body is used for learning the shared knowledge. During training and testing, the body of ANIL does not participate in the inner optimization for meta-learning. Based on this principle, the first-order adaptation $\phi_{ij}(T_i^S)$ can be written as

$$\begin{aligned} &\phi_i = (\phi_i^{body}, \phi_i^{head}) = \psi, \\ &\phi_i^{head} = \phi_i^{head} - \beta\nabla \sum_{\{S_{ij}, Q_{ij}\} \in T_i^S} \boldsymbol{L}(\phi_i, T_{ij}), \end{aligned} \tag{16}$$

where β is the learning rate of the first-order adaptation. According to the principle of ANIL, the second-order adaptation can be written as

$$\begin{aligned} &\theta_{ij} = (\theta_{ij}^{body}, \theta_{ij}^{head}) = \phi_i, \\ &\theta_{ij}^{head} = \theta_{ij}^{head} - \alpha\nabla \sum_{z_{ijk} \in S_{ij}} l(\theta_{ij}, z_{ijk}), \end{aligned} \tag{17}$$

where α is the learning rate of the second-order adaptation. Taking the regression problem as an example, the loss function $l(.)$ is the same as hierarchical MAML. The calculation process of Equation 10 is similar to Algorithm 2, the difference is that we replaced line 5 with $\phi_i = (\phi_i^{body}, \phi_i^{head}) = \psi$, lines 7 and 13 with $\theta_{ij} = (\theta_{ij}^{body}, \theta_{ij}^{head}) = \phi_i$, and lines 8 and 14 with $\theta_{ij}^{head} = \theta_{ij}^{head} - \alpha\nabla \sum_{z_{ijk} \in S_{ij}} l(\theta_{ij}, z_{ijk})$.

4.3.4 Second-order Approximation. In Algorithm 2, the calculation of $\nabla\mathcal{L}oss$ may involve high-order derivations, potentially impacting both the computational overhead and the accuracy of some hierarchical meta-learning algorithms. In the implementation using pytorch, these higher-order derivations are the result of retaining the computational graph when performing gradient calculation operation on lines 8, 11, and 14 in Algorithm 2. Only retaining the computational graph in lines 8 and 14 leads to second-order derivations, while retaining the computational graph in lines 8, 11, and 14 leads to third-order derivations (original H-MAML). In

this paper, we will use second-order approximation to approximate hierarchical meta-learning algorithms, which will be detailed in Section 5.3.3.

5 EXPERIMENT

We perform experiments to answer:

- Do hierarchical meta-learning algorithms outperform their meta-learning counterparts?
- When do hierarchical meta-learning algorithms outperform their meta-learning counterparts?
- Why do hierarchical meta-learning algorithms outperform their meta-learning counterparts?
- Why do we need hierarchical meta-learning?

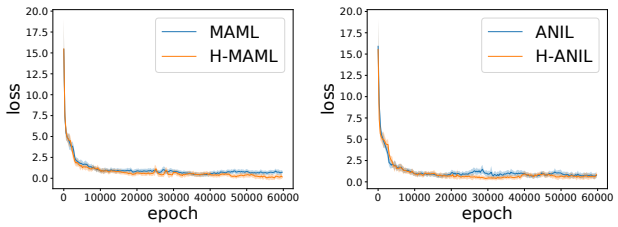
In the following experiments, we conduct five replicate experiments and show the mean value of each experiment result. Note that we do not use a meta-validation set since the main purpose of our experiment is not to produce state-of-the-art performance. Most of the hyperparameters are derived from existing papers and the other few hyperparameters are obtained on the training set.

5.1 Toy Regression Experiment

To preliminary explore hierarchical meta-learning algorithms and answer the first two questions, we conduct a toy regression experiment.

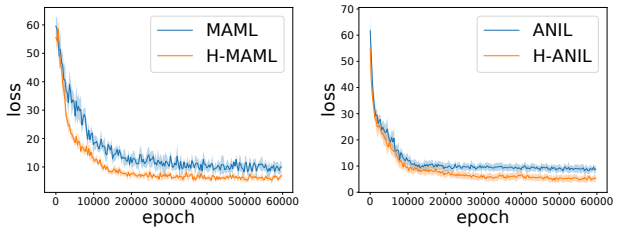
5.1.1 ***Dataset Setups.*** In the toy regression problem, we evaluate hierarchical meta-learning algorithms on the problem of 5-shot sine wave regression and compare them with their meta-learning counterparts. For a specific task, the ground truth function is a sine curve $y(x) = Asin(x + b)$, where the amplitude A and phase b follow the uniform distribution on intervals $D(A)$ and $D(b)$, respectively. The data point x in each task is drawn randomly from interval $[-5, 5]$. For meta-learning algorithm, we randomly select an amplitude A_i and a phase b_i from $D(A)$ and $D(b)$ to construct task $T_i = \{S_i, Q_i\}$, where $|S_i| = |Q_i| = 5$. For hierarchical meta-learning algorithms, we randomly select an amplitude sub-interval $D_s(A)$ and a phase sub-interval $D_s(b)$ from $D(A)$ and $D(b)$ to construct hyper-task $HT_i = \{T_i^S, T_i^Q\}$, where each task in T_i^S and T_i^Q shares the same structure with T_i, and $|T_i^S| = |T_i^Q| = 5$. The amplitude and phase of the tasks in T_i^S and T_i^Q are drawn randomly from the corresponding intervals $D_s(A)$ and $D_s(b)$ respectively. In other words, compared to the tasks used in meta-learning, the tasks within a hyper-task are more concentrated. The prediction loss is measured by mean squared error (MSE). To simulate unseen tasks, we separate the amplitude and phase intervals used for the training task and the testing task. Specifically, we set the first three quarters of $D(A)$ and $D(b)$ as the training intervals and the last quarter as the testing intervals. Based on the above settings, we evaluated the performance of our approach on small intervals $D(A) = [1, 5], D(b) = [0, \frac{\pi}{2}]$ and large intervals $D(A) = [1, 10]$ and $D(b) = [0, 2\pi]$.

5.1.2 ***Baselines and Our Approach.*** In this experiment, we use MAML and ANIL as baselines and compare them with their hierarchical meta-learning counterparts, *i.e.*, H-MAML and H-ANIL.
Network structure. To implement the four models, we follow the setup of [4], using a small neural network with an input layer of size 1, followed by 2 hidden layers of size 40 with ReLU nonlinearities, and then an output layer of size 1. All weight matrices use truncated normal initialization with a mean of 0 and a standard deviation of 0.01. For ANIL, we use the same setup as MAML and consider the last layer as its "head" and the other layers as its "body". For H-ANIL, we consider the last layer as its "head" for a specific task and consider the last two layers as its "head" for a specific hyper-task.
Learning rate. We set the learning rates of the base-learner and meta-learner as 0.001 for meta-learning algorithms while the learning rates of base-learner, meta-learner, and hyper-meta-learner as 0.001 for hierarchical meta-learning algorithms.
Adaptation step. For the adaptation step, meta-learning algorithms use two gradient descent steps (*i.e.*two rounds of inner loop) while hierarchical meta-learning algorithms use two gradient descent steps during first-order adaptation and two gradient descent steps during second-order adaptation.



(a) MAML with H-MAML. (b) ANIL with H-ANIL.

Figure 4: Comparison of hierarchical meta-learning algorithms with meta-learning algorithms on small intervals. The shaded area is one standard deviation interval of the mean value.



(a) MAML with H-MAML. (b) ANIL with H-ANIL.

Figure 5: Comparison of hierarchical meta-learning algorithms with meta-learning algorithms on large intervals.

5.1.3 ***Results.*** As shown in Figure 4, H-MAML and H-ANIL perform comparably with MAML and ANIL when we set $D(A) = [1, 5]$ and $D(b) = [0, \frac{\pi}{2}]$. However, as shown in Figure 5, when we increase the size of the intervals $D(A)$ and $D(b)$ to $[1, 10]$ and $[0, 2\pi]$ while fixing the lengths of $D_s(A)$ and $D_s(b)$, H-MAML and H-ANIL provide faster convergence and lower loss compared to their meta-learning counterparts. We believe that when the amplitude and phase intervals become too large, the distribution of the constructed tasks cannot be approximated as independent and identically distributed (i.i.d.). In such a scenario, ordinary meta-learning

algorithms learn tasks with bias. In contrast, hierarchical meta-learning algorithms learn from hyper-tasks. On the one hand, tasks within a hyper-task are constructed using similar amplitude and phase from small sub-intervals. On the other hand, the model's performance on unseen tasks can be improved through second-order adaptation, which provides a customized base-learner for each task, while first-order adaptation provides a customized meta-learner for each sub-interval.

In the toy regression experiment, we can draw the following conclusions:

- Hierarchical meta-learning algorithms outperform their meta-learning counterparts when the disparities between tasks increase.
- With the same training time, hierarchical meta-learning algorithms can still obtain better performance by second-order approximation (we will detail in Section 5.3.3).

5.2 Image Classification Experiment

To answer the second to fourth questions, we evaluate our approach on multiple few-shot image classification scenarios.

5.2.1 ***Dataset Setups.*** In the few-shot classification problem, we use Omniglot [9], MiniImageNet [2], ImageNet 2012 [8], and DomainNet [10] datasets. For meta-learning algorithms, the process of constructing tasks is the same as [4]. For the hierarchical meta-learning algorithm, we select tasks from the same environment to construct hyper-tasks. The environment and class information of the datasets can be found in Table 1.

Omniglot The Omniglot dataset consists of 1623 classes from 50 different alphabets and each class contains 20 samples. We divide the dataset into 40 alphabets for training and 10 alphabets for evaluation. In Omniglot datasets, one alphabet corresponds to an environment. When constructing the tasks in an environment, we randomly select N classes from the corresponding alphabet to construct a N-way task. We select 10 tasks from the same environment to construct a hyper-task, with 5 tasks as the support task set and 5 tasks as the query task set, *i.e.*, $|T^S| = |T^Q| = 5$. Note that we apply this hyper-task setting to all other datasets.

MiniImageNet and ImageNet 2012. The MiniImageNet dataset consists of 100 classes and each class contains 600 samples. We divide the dataset into 76 classes for training and 24 classes for evaluation. To explore the performance of hierarchical meta-learning algorithms with more classes, we use the ImageNet 2012 dataset consisting of 1000 classes. We divide the dataset into 760 classes for training and 240 classes for evaluation. For each class, we also use 600 samples. MiniImageNet and ImageNet 2012 datasets do not provide explicit environment criteria, thus we construct candidate class pools to simulate different environments. A candidate classes pool contains 40 randomly selected classes and corresponds to an environment. When constructing the task in an environment, we randomly select N classes from the candidate classes pool to construct a N-way task.

DomainNet. DomainNet is a multi-domain generalization dataset that consists of 345 classes from 6 domains. We divide the dataset into 5 domains for training and 1 domain for evaluation. In DomainNet datasets, one domain corresponds to an environment. When constructing the tasks in an environment, we randomly select N classes from the corresponding domain to construct a N-way task.

Table 1: Overview of the datasets.

	Omniglot	MiniImageNet	ImageNet 2012	DomainNet
Training classes	1271	76	760	345
Testing classes	352	24	240	345
Training environments	40	-	-	4
Testing environments	10	-	-	2

For all the datasets, we evaluate the performance of our approach on the setting of 5-way 1-shot, 5-way 5-shot, 20-way 5-shot, and 20-way 10-shot tasks.

5.2.2 ***Baselines and Our Approach.*** In this experiment, we use MAML [4], ANIL [16], ProtoNet [17] and HSML [22] as baselines and compare them with H-MAML, H-ANIL, and H-ProtoNet.

Network structure. For Omniglot, MiniImageNet, and ImageNet 2012 datasets, MAML, ANIL, and H-MAML use the same implementation with [4]. ANIL defines the last layer as its "head" and the other layers as its "body". For H-ANIL, we add a hidden layer to the last layer of ANIL. We consider the last layer as its "head" for a specific task and consider the last two layers as its "head" for a specific hyper-task. For ProtoNet, we replace a metric-based classifier with the last hidden layer of MAML. For H-ProtoNet, we use the same architecture as ProtoNet. Since the pictures in the DomainNet dataset are of high resolution, we change the backbone to ResNet9 while keeping the other settings unchanged. For HSML, we use the same implementation with [22].

Learning rate. For the meta-learning algorithms, we set the learning rates of the base-learner and the meta-learner as 0.01 and 0.001, respectively. For hierarchical meta-learning algorithms, we set the learning rates of the base-learner, the meta-learner, and the hyper-meta-learner as 0.005, 0.001, and 0.001, respectively.

Adaptation step. The setups of the adaptation step are the same as those in the toy regression experiment.

5.2.3 ***Results.*** We present the results in Table 2.

Omniglot. In the Omniglot dataset, compared to meta-learning algorithms, hierarchical meta-learning algorithms improve accuracy by 0.6% on average with the help of alphabet-customized meta-learners.

MiniImageNet and ImageNet 2012. The same result can be found in MiniImageNet and ImageNet 2012 datasets. Note that these two datasets do not provide explicit environment-level information (e.g. relationship between alphabet and classes), and we simulate the different environments by constructing candidate class pools. Meanwhile, we can find that hierarchical meta-learning algorithms improve more accuracy (10.8%) on the ImageNet 2012 dataset with more underlying classes compared to the 3.7% improvement on MiniImageNet. This is because ImageNet 2012 has more classes and the distribution of different tasks exhibits a larger gap. As a result, meta-learning algorithms learn more bias with these non-i.i.d. tasks. In contrast, hierarchical meta-learning algorithms learn from the hyper-tasks whose classes are drawn from narrower candidate class pools, thus tasks in the same hyper-task share more knowledge. If we consider ImageNet 2012 as a dataset with complex task distributions, then we can think of hierarchical meta-learning algorithms

Table 2: Accuracy of hierarchical meta-learning algorithms and the baselines obtain on four image classification datasets.

Method	5-way Accuracy								20-way Accuracy							
	Omniglot		MiniImageNet		ImageNet 2012		DomainNet		Omniglot		MiniImageNet		ImageNet 2012		DomainNet	
	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot	5-shot	10-shot	5-shot	10-shot	5-shot	10-shot	5-shot	10-shot
MAML	96.2	98.3	47.5	63.7	39.5	55.7	41.8	50.7	97.7	99.6	52.7	61.6	40.0	51.9	44.6	49.5
H-MAML	97.9	99.8	50.4	68.5	49.8	67.2	52.1	64.6	98.7	99.9	55.2	66.4	50.3	63.1	55.1	63.2
ANIL	97.0	99.4	47.1	62.6	38.7	55.2	42.4	51.3	98.5	99.8	51.5	60.4	39.7	51.7	45.6	50.1
H-ANIL	98.8	99.7	50.1	67.6	49.0	66.7	52.6	65.1	99.6	99.9	54.3	64.9	50.2	63.2	56.1	64.1
ProtoNet	97.4	99.5	48.7	64.5	40.6	56.8	42.8	51.6	98.7	99.7	53.8	62.5	40.9	52.6	45.8	50.7
H-ProtoNet	98.9	99.9	51.6	69.3	50.7	68.1	53.1	65.5	99.9	99.9	56.6	67.2	51.5	63.8	56.4	64.3
HSML	98.7	99.4	50.5	68.4	49.6	67.8	52.7	65.1	98.7	99.7	55.8	67.0	50.8	63.3	54.7	63.2

in such a way that, instead of leveraging explicit environment-level information, it improves prediction performance by dividing a complex task distribution into multiple simple task distributions for meta-learning (this process is the same as the improvement of meta-learning on single-task learning).

DomainNet. In the DomainNet dataset, all domains share the same class set. Meta-learning algorithms do not distinguish between images of the same class under different domains, which leads to poor performance. In contrast, hierarchical meta-learning algorithms are inherently suitable for multi-domain scenarios. It considers different domains as different environments and constructs the hyper-tasks for each domain. By learning from the hyper-tasks, the hyper-meta-learner of the hierarchical meta-learning algorithm captures domain-level shared knowledge and leverages tasks from the same domain for first-order adaptation to improve their performance on unseen domains. As a result, hierarchical meta-learning algorithms improve accuracy by 12.1% on average.

In fact, just as increasing the length of intervals $D(A)$ and $D(b)$ in the toy regression experiment, increasing the number of domains or increasing the number of classes in a dataset essentially increases the disparities between tasks. Therefore, in scenarios with more domains or more classes, hierarchical meta-learning algorithms are expected to make more improvements than their meta-learning counterparts. In addition, we can find that hierarchical meta-learning algorithms outperform HSML in most scenarios and H-ProtoNet outperform HSML in all scenario. Based on the above results, we can draw the following conclusions:

- Hierarchical meta-learning algorithms outperform their meta-learning counterparts in the few-shot classification scenario.
- Hierarchical meta-learning algorithms have advantages in the scenario with more classes.
- Hierarchical meta-learning algorithms have advantages in scenarios with multiple domains.
- In contrast to the variants of MAML such as HSML, hierarchical meta-learning can be equipped with most meta-learning algorithms. This means that hierarchical meta-learning is more likely to obtain better performance when better meta-learning algorithms are developed.

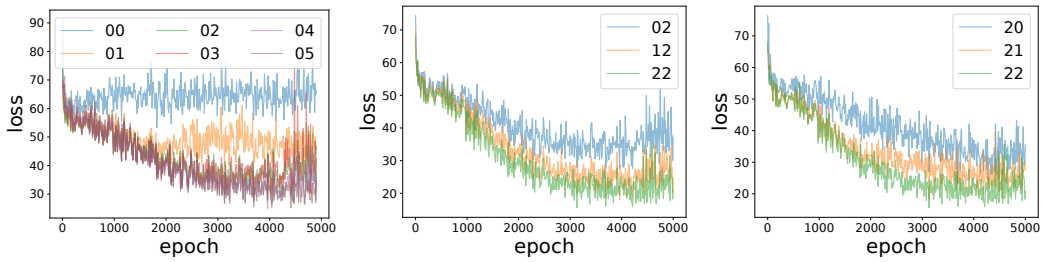
5.3 Ablation Experiment

Although previous experiments have demonstrated the superiority of hierarchical meta-learning, several suspicions related to the common issues in meta-learning remain. Based on the setups of toy regression and image classification experiments, we take MAML and H-MAML as examples to compare hierarchical meta-learning and meta-learning in-depth.

5.3.1 Adaptation Step. In the toy regression experiments, H-MAML adapts more steps compared to MAML, raising suspicion about short horizon [19]. For example, H-MAML takes two steps of first-order adaptation and each first-order adaptation consists of two steps of second-order adaptation. However, MAML has only one-order adaptation and adapts three steps in that order. To evade the effect of the short horizon problem caused by too few adaptation steps, we increase the number of adaptation steps of MAML and compare the regression loss of H-MAML and MAML after each adaptation step. Figure 6(a) shows the loss curve of MAML, Figure 6(b) shows the loss curve of H-MAML obtain by different first-order adaptation steps, and Figure 6(c) shows the loss curve of H-MAML obtain by different second-order adaptation steps after two first-order adaptation steps. As shown in Figure 6(a), we increase MAML's adaptation steps from 3 to 5. However, there is no significant reduction in regression loss for MAML after more than three adaptation steps. In contrast, as shown in Figure 6(b) and Figure 6(c), H-MAML optimizes the loss not only in second-order adaptation but also in first-order adaptation, which means that H-MAML learns not only task-level shared knowledge but also environment-level shared knowledge. Experimental results show that the performance improvement brought by H-MAML comes from learning environment-level shared knowledge, rather than more adaptation steps.

5.3.2 Feature Leakage. A direct comparison of H-MAML with MAML does not seem fair because the first-order adaptation of H-MAML uses a small number of tasks (*i.e.*, support task set) in the target environment and a small number of samples (*i.e.*, support set) in the target task, while MAML only uses the latter. This means that environment-related features are leaked to H-MAML but not to MAML. For fairness, we conduct two sets of image classification experiments with a 5-way 1-shot setup. For MAML, we fine-tune MAML with the additional tasks used by H-MAML, *i.e.*, we use these data to continue training MAML for several episodes[4]. For H-MAML, instead of using tasks from the target environment, we use tasks from the training set for first-order adaptation. As shown in Table 3, the additional tasks in the target environment cannot

[4]The fine-tuning mentioned here refers to the continued training of the meta-learner, rather than adaptation on the base learner.



(a) MAML (b) H-MAML first-order adaptation (c) H-MAML second-order adaptation

Figure 6: Regression loss of MAML and H-MAML after each adaptation step. Legend "12" in Figure 6(b) represents the loss curve after one first-order adaptation step and two second-order adaptation steps, and the same for the other legends. Note that since MAML has only second-order adaptation, the legend of the loss curve after two adaptation steps for MAML is "02". The curves "03", "04", and "05" in Figure 6(a) are almost overlapping, while the curve in the other Figures are separated, which means that more adaptation steps are not the reason why H-MAML outperforms MAML.

Table 3: Comparison of H-MAML and MAML on 5-way 1-shot image classification problem. MAML + denotes MAML fine-tuning with additional tasks. H-MAML - denotes H-MAML adapting with tasks from the training set.

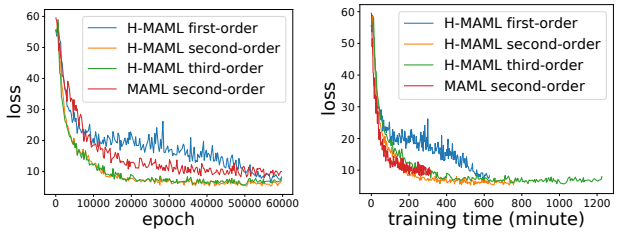
Method	Omniglot	MinImageNet	ImageNet 2012	DomainNet
MAML+	97.1	48.9	43.2	40.2
H-MAML-	97.3	49.6	45.9	39.8
MAML	96.2	47.5	39.5	41.8
H-MAML	97.9	50.4	49.8	52.1

help MAML outperforms H-MAML. Note that in the DomainNet dataset, the additional tasks degrade the performance of MAML. In addition, although using tasks from the training set leads H-MAML to adapt to biases, it outperforms MAML on most datasets.

5.3.3 High-order Derivation. To investigate the effect of higher-order derivations in H-MAML, we compare the overhead and regression loss of third-order H-MAML, second-order H-MAML, first-order H-MAML, and MAML based on the setup of toy regression experiments. As shown in Figure 7(a), when setting epoch as the horizontal coordinate, the loss curve of third-order H-MAML and second-order H-MAML perform comparably while first-order H-MAML obtains sub-optimal performance. However, as shown in Figure 7(b), when the horizontal coordinate is changed to training time, second-order H-MAML obtains the best performance because it obviates the computation of high-order derivation and has a lower training overhead. For simplicity, we use the second-order hierarchical meta-learning algorithm to approximate the hierarchical meta-learning algorithm. In addition, we can find that second-order H-MAML can provide better performance than MAML with the same training time.

6 CONCLUSION AND FUTURE WORKS

We propose a new paradigm of hierarchical meta-learning to learn the meta-learning algorithm. Our approach offers several benefits. Firstly, it provides a method for constructing hierarchically structured tasks and achieving accurate knowledge extraction for non-i.i.d. tasks using hierarchical meta-learning. Secondly, leveraging second-order approximation, it yields better few-shot learning performance than the traditional meta-learning paradigm, given the same training time constraint. Lastly, it is a flexible framework that can be equipped with most meta-learning algorithms. In this paper, we stack the meta-learning paradigm on top of the meta-learning paradigm. This kind of stacking can be performed multiple times. In future work, we plan to explore the limitations of the stacking tiers and further investigate the concept of learning to learn.



(a) Same training step (b) Same training time

Figure 7: Comparison H-MAML with its low-order derivation approximations and MAML. Figure 7(a) and Figure 7(a) depict the regression loss of the algorithms under the same training step and training time, respectively.

ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China (Grant No.62232007, No.61821003) and the Natural Science Foundation of Hubei Province No.2022CFB060.

REFERENCES

[1] Liam Collins, Aryan Mokhtari, and Sanjay Shakkottai. 2020. Task-Robust Model-Agnostic Meta-Learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.*

[2] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. 2014. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In *Proceedings of the 31th International Conference on*

Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014, Vol. 32. JMLR.org, 647–655.

[3] Alireza Fallah, Aryan Mokhtari, and Asuman E. Ozdaglar. 2021. Generalization of Model-Agnostic Meta-Learning Algorithms: Recurring and Unseen Tasks. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. 5469–5480.

[4] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, Vol. 70. PMLR, 1126–1135.

[5] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. 2021. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence* 44, 9 (2021), 5149–5169.

[6] Xiang Jiang, Mohammad Havaei, Farshid Varno, Gabriel Chartrand, Nicolas Chapados, and Stan Matwin. 2019. Learning to Learn with Conditional Class Dependencies. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[7] Sharu Theresa Jose and Osvaldo Simeone. 2021. An Information-Theoretic Analysis of the Impact of Task Similarity on Meta-Learning. In *IEEE International Symposium on Information Theory, ISIT 2021, Melbourne, Australia, July 12-20, 2021*. IEEE, 1534–1539.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. 1106–1114.

[9] Brenden M. Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B. Tenenbaum. 2011. One shot learning of simple visual concepts. In *Proceedings of the 33th Annual Meeting of the Cognitive Science Society, CogSci 2011, Boston, Massachusetts, USA, July 20-23, 2011*. cognitivesciencesociety.org.

[10] Aristotelis Leventidis, Laura Di Rocco, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2021. DomainNet: Homograph Detection for Data Lake Disambiguation. In *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*. OpenProceedings.org, 13–24.

[11] Da Li and Timothy M. Hospedales. 2020. Online Meta-learning for Multi-source and Semi-supervised Domain Adaptation. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XVI*, Vol. 12361. Springer, 382–403.

[12] Hong Liu, Mingsheng Long, Jianmin Wang, and Yu Wang. 2020. Learning to Adapt to Evolving Domains. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

[13] Alex Nichol, Joshua Achiam, and John Schulman. 2018. On First-Order Meta-Learning Algorithms. *CoRR* abs/1803.02999.

[14] Boris N. Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. 2018. TADAM: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. 719–729.

[15] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. 2019. Moment Matching for Multi-Source Domain Adaptation. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 1406–1415.

[16] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. 2020. Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[17] Jake Snell, Kevin Swersky, and Richard S. Zemel. 2017. Prototypical Networks for Few-shot Learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 4077–4087.

[18] Jun Wu and Jingrui He. 2022. A Unified Meta-Learning Framework for Dynamic Transfer Learning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*. International Joint Conference on Artificial Intelligence, 3573–3579.

[19] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger B. Grosse. 2018. Understanding Short-Horizon Bias in Stochastic Meta-Optimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[20] Ruijia Xu, Ziliang Chen, Wangmeng Zuo, Junjie Yan, and Liang Lin. 2018. Deep Cocktail Network: Multi-Source Unsupervised Domain Adaptation With Category Shift. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 3964–3973.

[21] Brandon Yang, Gabriel Bender, Quoc V. Le, and Jiquan Ngiam. 2019. CondConv: Conditionally Parameterized Convolutions for Efficient Inference. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 1305–1316.

[22] Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. 2019. Hierarchically Structured Meta-learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 7045–7054.