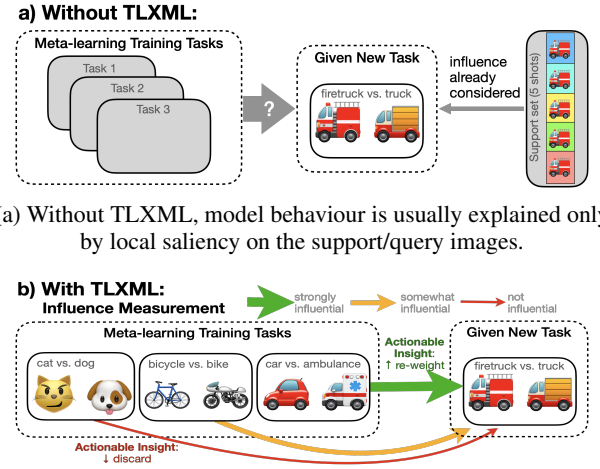
# A Task-Level Explanation Framework for Meta-Learning Algorithms

**Yoshihiro Mitsuka**[1], **Shadan Golestan**[2], **Zahin Sufiyan**[3], **Shotaro Miwa**[4] **Osmar Zaiane**[2, 3]
[1] Information Technology R&D Center, Mitsubishi Electric
[2] Alberta Machine Intelligence Institute
[3] Department of Computing Science, University of Alberta
[4] Advanced Technology R&D Center, Mitsubishi Electric
{Mitsuka.Yoshihiro@bp, Miwa.Shotaro@bc}.MitsubishiElectric.co.jp
{golestan, zsufiyan, zaiane}@ualberta.ca

## Abstract

Meta-learning enables models to rapidly adapt to new tasks by leveraging prior experience, but its adaptation mechanisms remain opaque, especially regarding how past training tasks influence future predictions. We introduce TLXML (Task-Level eXplanation of Meta-Learning), a novel framework that extends influence functions to meta-learning settings, enabling task-level explanations of adaptation and inference. By reformulating influence functions for bi-level optimization, TLXML quantifies the contribution of each meta-training task to the adapted model's behaviour. To ensure scalability, we propose a Gauss-Newton-based approximation that significantly reduces computational complexity from $O(pq^2)$ to $O(pq)$, where p and q denote model and meta parameters, respectively. Empirical results demonstrate that TLXML effectively ranks training tasks by their influence on downstream performance, offering concise and intuitive explanations aligned with user-level abstraction. This work provides a critical step toward interpretable and trustworthy meta-learning systems.




(a) Without TLXML, model behaviour is usually explained only by local saliency on the support/query images.

(b) TLXML ranks meta-training tasks by their influence on the new task, providing actionable insights.

Figure 1: Key insights of TLXML.

## 1 Introduction

Meta-learning, or "learning to learn," equips models with the ability to rapidly adapt to unseen tasks, addressing limitations in generalization caused by data scarcity during training (Song and Jeong 2024; Li et al. 2018; Shu et al. 2021; Lu et al. 2021) and distribution shifts in deployment environments (Mann et al. 2021; Mouli, Alam, and Ribeiro 2024; Lin et al. 2020). Despite its growing success and fast adaptation scenarios, meta-learning remains largely opaque. Current approaches offer limited insight into which training tasks influence the adaptation process and final predictions, creating significant barriers for transparency, trust, and safe deployment in real-world applications (Zhang et al. 2020; Khattar et al. 2024; Wen et al. 2022; Yao et al. 2024). Consequently, robust explanation methods are required to enhance transparency and ensure the safe and reliable deployment of autonomous systems.

Most explanation methods in machine learning focus on local interpretability, aiming to understand model behaviour around individual input examples (Figure 1a). While such methods can be adapted to explain meta-learners' post-adaptation, they fall short of capturing the unique characteristics of meta-learning. Notably, the final model behaviour in meta-learning is shaped not only by the test-time support set but also by the collection of prior training tasks. This results in high adaptation sensitivity, as highlighted by Agarwal et al. (Agarwal, Yurochkin, and Sun 2021) , who showed that accuracy on CIFAR-FS can range from $4\%$ to $95\%$ depending solely on the choice of support images—even when the model remains unchanged.

Furthermore, local explanations often require technical expertise to interpret (Adebayo et al. 2022) , and may not be helpful for end-users seeking actionable or intuitive insights. In contrast, task-level explanations—those that attribute predictions to previously encountered tasks—align more naturally with how meta-learning models are trained and how humans reason about prior experience (Figure 1b). Understanding how individual tasks contribute to model behaviour is especially important as meta-learning systems increasingly operate across heterogeneous, multi-domain datasets. Task-level insights can improve both the interpretability and safety of such systems by identifying which training con-

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

texts are most responsible for specific behaviours, making it possible to re-weight helpful influences and discard harmful ones, ultimately improving the performance and reliability of deployed systems.

Influence functions (Hampel 1974; Cook and Weisberg 1980) provide a powerful tool for tracing model predictions back to training data, and have been successfully used in standard supervised learning to identify influential data points (Koh and Liang 2017; Koh et al. 2019), assess robustness (Cohen, Sapiro, and Giryes 2020), detect bias (Han and Tsvetkov 2020), and improve interpretability (Chhabra et al. 2024). However, influence functions have not yet been extended to meta-learning, where the training examples are tasks, not data points.

In this paper, we propose TLXML, a framework for Task-Level eXplanation of Meta Learning via influence functions. TLXML quantifies the impact of individual meta-training tasks on the model's adaptation and inference behaviour. We reformulate influence functions to accommodate the bi-level optimization structure inherent in meta-learning algorithms. Our approach enables users to understand how prior tasks shape the learning process in a principled and interpretable way.

**Contributions.** Our main contributions are as follows: 1) Task-Level Explanations for Meta-Learning: We introduce TLXML, a principled method for quantifying the influence of meta-training tasks on adaptation and prediction in meta-learning. It offers concise, interpretable explanations aligned with user abstraction levels. 2) Scalable Influence Computation: We analyze the computational cost of TLXML, showing that the exact method scales poorly with $O(pq^2)$, where p and q are the number of model and meta-parameters. We propose a Gauss-Newton-based approximation that reduces the cost to $O(pq)$, making TLXML feasible for practical use. 3) Empirical Validation: We empirically demonstrate that TLXML can meaningfully rank meta-training tasks by their influence on adaptation and performance, successfully identifying helpful versus unhelpful training tasks across several benchmarks, and it can be utilized for improving the adaptation ability of meta-trained models.

## 2 Related work

**Influence functions for machine learning**. The primary focus of existing research is the use of influence functions in supervised learning initiated by Koh and Liang (2017). Influence functions have been successfully used for multiple purposes, such as explaining model behavior with respect to training data in various tasks (Barshan, Brunet, and Dziugaite 2020; Koh and Liang 2017; Han, Wallace, and Tsvetkov 2020), quantifying model uncertainty (Alaa and Van Der Schaar 2020), crafting/detecting adversarial training examples (Cohen, Sapiro, and Giryes 2020). These approaches focus on data-level explanations, rendering them of limited practical value in meta-learning settings. TLXML leverages influence functions for task-level explanations, offering more effective insights into how training tasks influence the model's behavior. However, as noted in Alaa and Van Der Schaar (2020), computing the Hessian matrix is expensive. To scale the methods of influence functions to more complex networks and larger datasets, TLXML uses an approximation method for faster computation of the matrix.

**Explainable AI (XAI) for meta-learning.** Naturally, XAI methods that are agnostic to the learning process are applicable for explaining inference in meta-learning. Although this area is still in its early stages, several research examples exist (Woźnica and Biecek 2021; Sijben et al. 2024; Shao et al. 2023). The closest work to ours is by Woźnica and Biecek (2021) who quantified the importance of meta-features, i.e., high-level characteristics of a dataset such as size, number of features and number of classes. However, we emphasize that the goal of meta-learning is to train models that generalize across *tasks*, and understanding the impact of training tasks is crucial for evaluating a model's adaptability.

**Impact of training data.** The robustness of machine learning models is another area where the impact of training data is frequently discussed (Khanna et al. 2019; Ribeiro, Singh, and Guestrin 2016). This topic has also been explored in the context of meta-learning such as creating training-time adversarial attacks via meta-learning (Zügner and Günnemann 2019; Xu et al. 2021), training robust meta-learning models by exposing models to adversarial attacks during the query step of meta-learning (Goldblum, Fowl, and Goldstein 2020), and data augmentation for enhancing the performance of meta-learning algorithms (Ni et al. 2021a). However, influence functions have not been utilized for meta-learning to date. Similar to Khanna et al. (2019); Barshan, Brunet, and Dziugaite (2020) in the case of supervised learning, TLXML also leverages Fisher information metrics, which provides a valuable geometric viewpoint for analyzing the model's parameter space.

**Novelty of this work.** We frame task-level explanation in meta-learning as estimating the influence of each meta-learning training task exerts on adaptation and inference. We introduce TLXML, a task-level influence framework that assigns the highest influence to training tasks most similar to the test task and, elevates the scores of tasks from the same sub-distribution, separating useful from less useful meta-learning training tasks, and ultimately improving the performance of downstream classification tasks.

## 3 Preliminaries

**Influence functions.** Koh and Liang (2017) proposed influence functions for measuring the impact of training data on the outcomes of supervised learning, under the assumption that the trained weights $\hat{\theta}$ minimize the empirical risk:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, L\left(D^{\text{train}}, f_\theta\right) = \underset{\theta}{\operatorname{argmin}}\, \frac{1}{n}\sum_{i=1}^{n} l\left(z_i, f_\theta\right)$$

where $f_\theta$ is the model to be trained, $D^{\text{train}}=\{z_i\}_{i=1}^{n}$ is the training dataset consisting of $n$ pairs of an input $x_i$ and a label $y_i$, i.e., $z_i=(x_i, y_i)$, and the total loss $L$ is the sum of the losses $l$ of each data point. The influence functions are

defined with a perturbation $\epsilon$ for each data point $z_j$:

$$\hat{\theta}_{\epsilon,j} = \underset{\theta}{\operatorname{argmin}} L_{\epsilon,j}\left(D^{\text{train}}, f_\theta\right) = \underset{\theta}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} l\left(z_i, f_\theta\right) + \epsilon l\left(z_j, f_\theta\right),$$

which is considered as a shift of the probability that $z_j$ occurs in the data distribution. The influence of the data $z_j$ on the model parameter $\hat{\theta}$ is defined as its increase rate with respect to this perturbation:

$$I^{\text{param}}(j) \stackrel{\text{def}}{=} \left.\frac{d\hat{\theta}_{\epsilon,j}}{d\epsilon}\right|_{\epsilon=0} = -H^{-1}\left.\frac{\partial l\left(z_j, f_\theta\right)}{\partial\theta}\right|_{\theta=\hat{\theta}} \quad (1)$$

where the Hessian is given by $H = \partial_\theta\partial_\theta L|_{\theta=\hat{\theta}}$. The influence on a differentiable function of $\theta$ is defined through the chain rule. For example, the influence on the loss of a test data $z_{\text{test}}$ is calculated to be

$$I^{\text{perf}}(z_{\text{test}}, j) \stackrel{\text{def}}{=} \left.\frac{dl\left(z_{\text{test}}, f_{\hat{\theta}_{\epsilon,z_j}}\right)}{d\epsilon}\right|_{\epsilon=0} = \left.\frac{dl\left(z_{\text{test}}, f_\theta\right)}{d\theta}\right|_{\theta=\hat{\theta}} \cdot I^{\text{param}}(j).$$

See Supplement A.2 for the derivation of Eq. 1. An underlying assumption is that the Hessian matrix is invertible, which is not always true. Typically, in over-parameterized networks, the loss function often has non-unique minima with flat directions around them. In this paper, we examine how the definition of influence functions extends to cases with a non-invertible Hessian matrix.

**Supervised meta-learning.** In supervised meta-learning (See Hospedales et al. (2021) for a review), a task $\mathcal{T}$ is defined as a pair $\left(\mathcal{D}^{\mathcal{T}}, \mathcal{L}^{\mathcal{T}}\right)$ where $\mathcal{D}^{\mathcal{T}}$ represents the dataset and $\mathcal{L}^{\mathcal{T}}$ is the associated loss function for the learning task. The occurrence of each task follows a distribution $\mathcal{T}\sim p\left(\mathcal{T}\right)$. An adaptation algorithm $\mathcal{A}$ takes as input a task $\mathcal{T}$ and meta-parameters $\omega$, and outputs the weights $\hat{\theta}$ of the model $f_\theta$. The learning objective is stated as the optimization of $\omega$ with respect to the test loss averaged over the task distribution:

$$\hat{\omega} = \underset{\omega}{\operatorname{argmin}} \underset{\mathcal{T}\sim p(T)}{E}\left[\mathcal{L}^{\mathcal{T}}(\mathcal{D}^{\mathcal{T},\text{test}}, f_{\hat{\theta}^{\mathcal{T}}})\right] \quad \text{with } \hat{\theta}^{\mathcal{T}} = \mathcal{A}(\mathcal{T}, \omega)$$

The formulation of empirical risk minimization uses sampled tasks as building blocks, which are divided into a taskset for training (source taskset) $D^{\text{src}}=\{\mathcal{T}^{\text{src(i)}}\}_{i=1}^{M}$ and a taskset for testing (target taskset) $D^{\text{trg}}=\{\mathcal{T}^{\text{trg(i)}}\}_{i=1}^{M'}$ The learning objective is stated as:

$$\hat{\omega} = \underset{\omega}{\operatorname{argmin}} \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}^{\text{src(i)}}(\mathcal{D}^{\text{src}(i)\text{test}}, f_{\hat{\theta}^i}) \quad (2)$$
$$\text{with } \hat{\theta}^i = \mathcal{A}(\mathcal{D}^{\text{src}(i)\text{train}}, \mathcal{L}^{\text{src(i)}}, \omega)$$

One performance metric in meta-testing is the test loss $\mathcal{L}^{\text{trg(i)}}(\mathcal{D}^{\text{trg}(i)\text{test}}, f_{\hat{\theta}^i})$ where $\hat{\theta}^i{=}\mathcal{A}(\mathcal{D}^{\text{trg}(i)\text{train}}, \mathcal{L}^{\text{trg(i)}}, \hat{\omega})$. We experiment with MAML (Finn, Abbeel, and Levine 2017), and Prototypical Network (Protonet) (Snell, Swersky, and Zemel 2017), two widely used meta-learning methods. Supplement A.1 provides the explicit forms of $\mathcal{A}$ for them.

# 4 Proposed Method

## 4.1 Task-level Influence Functions

We now describe our method. TLXML measures the influence of training tasks on the adaptation and inference processes in meta-learning. To measure the influence of a training task $\mathcal{T}^j$ on the model's behaviors, we consider the task-level perturbation of the empirical risk defined in Eq. 2:

$$\hat{\omega}_\epsilon^j = \arg\min_{\omega} \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}^i(\mathcal{D}^{(i)\text{test}}, f_{\hat{\theta}^i}) + \epsilon\mathcal{L}^j(\mathcal{D}^{(j)\text{test}}, f_{\hat{\theta}^j}) \quad (3)$$

with $\hat{\theta}^i{=}\mathcal{A}(\mathcal{D}^{(i)\text{train}}, \mathcal{L}^i, \omega)$. The influence on $\hat{\omega}$ is given by:

$$I^{\text{meta}}(j) \stackrel{\text{def}}{=} \left.\frac{d\hat{\omega}_\epsilon^j}{d\epsilon}\right|_{\epsilon=0} = -\,H^{-1}\left.\frac{\partial\mathcal{L}^j(\mathcal{D}^{(j)\text{test}}, f_{\hat{\theta}^j})}{\partial\omega}\right|_{\omega=\hat{\omega}} \quad (4)$$

where the Hessian matrix $H$ is defined as follows:

$$H = \frac{1}{M}\sum_{i=1}^{M}\left.\frac{\partial^2\mathcal{L}^i(\mathcal{D}^{(i)\text{test}}, f_{\hat{\theta}^i})}{\partial\omega\partial\omega}\right|_{\omega=\hat{\omega}}. \quad (5)$$

See Supplement A.2 for the derivation of Eq. 4. The model's behavior is affected by the perturbation through the adapted parameters $\hat{\theta}_\epsilon^{ij} \equiv \mathcal{A}(\mathcal{D}^{(i)\text{train}}, \mathcal{L}^i, \hat{\omega}_\epsilon^j)$. The influence of the training task $\mathcal{T}^j$ on model parameters $\hat{\theta}^i = \mathcal{A}(\mathcal{D}^{(i)\text{train}}, \mathcal{L}^i, \hat{\omega})$ is measured by:

$$I^{\text{adpt}}(i,j) \stackrel{\text{def}}{=} \left.\frac{d\hat{\theta}_\epsilon^{ij}}{d\epsilon}\right|_{\epsilon=0} \quad (6)$$
$$= \left.\frac{\partial\mathcal{A}\left(\mathcal{D}^{(i)\text{train}}, \mathcal{L}^i, \omega\right)}{\partial\omega}\right|_{\omega=\hat{\omega}} I^{\text{meta}}(j) \quad (7)$$

The influence of the training task $\mathcal{T}^j$ on the loss of a test task $\mathcal{T}^i$ is measured by:

$$I^{\text{perf}}(i,j) \stackrel{\text{def}}{=} \left.\frac{d\mathcal{L}^i\left(\mathcal{D}^{(i)\text{test}}, f_{\hat{\theta}_\epsilon^{ij}}\right)}{d\epsilon}\right|_{\epsilon=0} \quad (8)$$
$$= \left.\frac{\partial\mathcal{L}^i\left(\mathcal{D}^{(i)\text{test}}, f_\theta\right)}{\partial\theta}\right|_{\theta=\hat{\theta}^{(i)}} I^{\text{adpt}}(i,j) \quad (9)$$

The training tasks are used only for evaluating $I^{\text{meta}}$. This means that we can obtain other explanation data without requiring access to the original raw data. By retaining only the calculated $I^{\text{meta}}$ from the meta-learning process, we can

mitigate storage concerns, making this approach suitable for devices with limited storage capacity. We note that the above method can be extended to a higher level of abstraction than task-level for situations where task-level explanations are insufficient, a direction left for future work. See Supplement A.3 for details.

### 4.2 Approximation via Gauss-Newton matrix

TLXML faces computational barriers when applied to large models, due to the cost of handling the Hessian in Eq. 5. Although the Hessian is defined as the second-order tensor of the meta parameters, the bi-level structure of meta-learning raises a third-order tensor in the form $\partial_\omega \partial_\omega \theta$ appears during its computation, resulting in a computational cost of at least $\mathcal{O}(pq^2)$ for a model with $p$ weights and $q$ meta-parameters. (see A.4 for details). Furthermore, as is common in matrix inversion issues, inverting the Hessian incurs a computational cost of $\mathcal{O}(q^3)$, which one of the subjects addressed by Koh and Liang (2017) for supervised learning.

We use the Gauss-Newton matrix (GN matrix) to approximate the Hessian matrix. For specific loss functions, e.g., mean squared error and cross-entropy, the Hessian can be decomposed into a sum of outer products of two vectors along with terms containing second-order derivatives. In this work, we only focus on the case of cross-entropy with the softmax function, which leads to:

$$\frac{\partial^2 L}{\partial\omega\partial\omega} = \sum_{njk} \sigma_k\left(\boldsymbol{y}_n\right)\left(\delta_{kj} - \sigma_j\left(\boldsymbol{y}_n\right)\right)\frac{\partial y_{nk}}{\partial\omega}\frac{\partial y_{nj}}{\partial\omega} - \sum_{njk} t_{nj}\left(\delta_{jk} - \sigma_k\left(\boldsymbol{y}_n\right)\right)\frac{\partial^2 y_{nk}}{\partial\omega\partial\omega} \tag{10}$$

where $y$ is the output of the last layer, $\sigma$ is the softmax function, $t$ is the one-hot vector of the target label, $j, k$ are class indices, and $n$ is the index specifying tasks and input tensors in them. The first term of Eq. 11 is the GN matrix or the Fisher information metric. Since the second-order derivatives in the second term give rise to third-order tensors, we focus on cases where these derivatives are uncorrelated with their coefficients, allowing the Hessian to be approximated by the first term alone.

Approximating the Hessian using the GN matrix is well-established in supervised learning (see for example Martens (2020)). In supervised learning, both $L$ and $\mathbf{y}$ are functions of the model's weights $\theta$; in our setting, they are functions of the meta-parameters $\omega$. This difference does not affect the argument. Using basic facts regarding cross-entropy (see Supplement A.5), we show that when the training taskset closely approximates a distribution $P\left(X|\omega^*\right)$ and the learned $\hat{\omega}$ is close to $\omega^*$, the first term in Eq. 10 dominates. As the term is positive-semidefinite, we use its factorized form:

$$\frac{\partial^2 L}{\partial\omega_\mu\partial\omega_\nu} \sim \sum_{nj} \left(\mathbf{V}\right)_{\mu(nj)}\left(\mathbf{V}^{\mathrm{T}}\right)_{(nj)\nu} \tag{11}$$

where the parameter index $\mu$ as a row index and $(nj)$ as a column index. In the case of $q$ meta-parameters, $M$ tasks, $n$ data points per task, and $c$ target classes, the shape of $V$ is $q \times cnM$, meaning that, if $q > cnM$, the approximated Hessian has zero eigenvalues. This also happens for smaller $q$ if the columns of $\mathbf{V}$ are not independent of each other.

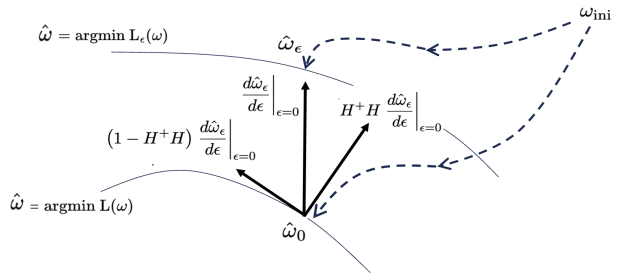



Figure 2: Diagram of the projected influence function, which measures the influence of a training task on the meta-parameters with the Hessian flat directions projected out.

### 4.3 Influence Functions with Flat Directions

Figure 2 depicts a generic case where flat directions of the Hessian appear in the parameter space. When the number of parameters is sufficiently large, the points that satisfy the minimization condition, $\hat{\omega} = \mathrm{argmin}\ \mathrm{L}(\omega)$, form a hypersurface, resulting in flat directions of the Hessian. The same holds for the perturbed loss $L_\epsilon(\omega)$ used for defining the influence functions. The position along the flat directions resulting from minimization depends on the initial conditions and the learning algorithm. We do not explore this dependency in our paper. Instead, we employ a geometric definition of influence functions: we take the partial inverse $H^+$ of $H$ in the subspace perpendicular to the flat directions, known as the pseudo-inverse matrix. With this approach, we modify the definitions of influence functions as:

$$I^{\mathrm{meta}}(j) \stackrel{\mathrm{def}}{=} H^+H\left.\frac{d\hat{\omega}_\epsilon^j}{d\epsilon}\right|_{\epsilon=0} = -H^+\left.\frac{\partial\mathcal{L}^j(D^{(j)\mathrm{test}}, f_{\hat{\theta}^j})}{\partial\omega}\right|_{\omega=\hat{\omega}}. \tag{12}$$

See Supplement A.2 for the derivation of the second equation. $H^+H$ represents the projection that drops the flat directions. When the Fisher information metric approximates the Hessian, those are viewed as the direction in which the data distribution remains unchanged. The influence function, projected by $H^+H$, expresses the sensitivity of $\hat{\omega}$ in the steepest direction of the distribution change.

Note that $H^+$ can be computed without diagonalizing $H = VV^T$. Instead, this can be achieved by first finding an orthogonal matrix $O$ that diagonalizes $V^TV$, i.e., $V^TV{=}O\Lambda O^T$. Then, $VV^T$ is implemented as a sequence of unnormalized projections in the direction of the column vectors of $VO = [v_1, v_2, \cdots]$, i.e., $VV^T = \sum v_i v_i^T$. Since $v_i$s are orthogonal to each other, the pseudo-inverse of $VV^T$ is given by adjusting the norms of the projections associated with non-vanishing vectors, i.e., $v_i v_i^T \to v_i v_i^T/|v_i|^4$ for $|v_i| > 0$. See Supplement B.1 for implementation details.

### 4.4 One-step Update via TLXML

We explore how TLXML can be used to enhance the performance of meta-learners. For utilizing influence functions for supervised learning, Koh and Liang (2017) employed a leave-out approach, in which the network is retrained after removing training data with low influence scores. Applying this approach to improve meta-learning seems reasonable, but it digresses from our theoretical argument. Since the influence functions are defined based solely on the local structure around the convergence point, it is not guaranteed that tasks or data with low scores exert negative influences throughout the entire training process.

Recalling that $I^{\text{meta}}$ represents the derivative of the meta-parameters with respect to the perturbation $\epsilon$ in the training task distribution, we can regard it as a linear approximation of the parameter shifts caused by that distribution change:

$$\delta\omega \sim \xi \times \left.\frac{d\hat{\omega}_\epsilon^j}{d\epsilon}\right|_{\epsilon=0} = \xi \times I^{\text{meta}}(j) \tag{13}$$

where we rename the non-zero perturbation parameter to $\xi$ to avoid confusion with the differential variable. The case of $\xi = -1$ corresponds to removing task $j$ during leave-out retraining. This equation allows us to adjust the parameters $\omega$ to account for the distribution change without rerunning the training. This approach can be used not only to block the influence of low-scored tasks with negative $\xi$ but also to amplify the influence of high-scored tasks with positive $\xi$.

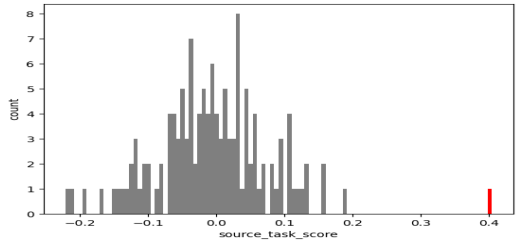
## 5 Experiments

In this section, we first examine whether TLXML provides adequate explanations that attribute the model's behavior to the influence of meta-learning training tasks; and second, we investigate whether incorporating TLXML into the training process can improve the performance of meta-learners.

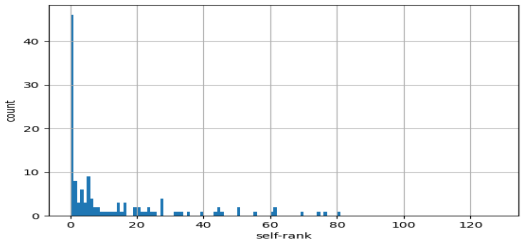
### 5.1 Validation of TLXML

We empirically investigate whether the proposed method satisfy two fundamental properties: **1)** If the network memorizes a training task, its influence on a test task with similar characteristics should be scored higher than the influence of other training tasks; and **2)** if the network encodes generalizable information about the task distribution, training tasks that belong to a subpopulation sharing salient features with the test task, should receive higher influence scores than tasks outside that subpopulation. Both properties are natural requirements for an explanation method to be regarded as a method based on past experiences, as training tasks that resemble the test task generally boost test performance.

**Setup** We employ MAML and Protonet as meta-learning algorithms and conduct experiments with few-shot learning problems taken from the MiniImagenet (Vinyals et al. 2016) and Ominiglot (Lake, Salakhutdinov, and Tenenbaum 2015) datasets. In addition to them, we use two datasets created for our purposes. Unless otherwise stated, experiments use a 5-way 5-shot configuration. The implementation builds on the *lean2learn* meta-learning library (Arnold et al. 2020) and PyTorch's automatic-differentiation framework. We train the meta-parameters with Adam, employing a meta-batch size of 32, that is, gradients are accumulated over 32 randomly sampled tasks before each update.




(a) Example of training task score distribution in a single test. Highlighted is the training task identical to the test task.




(b) Histogram of the self-ranks. The self-rank is defined as the rank of the training task identical to each test task.

Figure 3: Test with training tasks with a two-layer fully connected network (1,285 parameters) overfitted to 128 training tasks in MiniImagenet with MAML.

**Distinction of Tasks** To validate property 1, we generate pairs of similar training and test tasks by making each test task identical to one of the training tasks. We then apply Eq. 9 to the trained network and examine whether it assigns a higher influence score to the training task identical to each test task than to any other training task.

Figure 3 presents the results for a two-layer, fully connected network (1,285 parameters) trained with MAML and overfitted on 128 MiniImageNet training tasks. Figure 3a illustrates a successful case in which the training task identical to the test task (highlighted in red) is clearly distinguished from all other training tasks. Figure 3b shows the distribution of the ranks assigned to the training task identical to the test task across 128 trials (we call them *self-ranks*).

Although the self-task often appears near the top of the ranking, it is not always placed first(it is $12.6 \pm 18.9$). This is likely because of the non-convexity of the training loss. In our case, many of the 1285 Hessian eigenvalues are close to zero, and 92 are negative, violating the underlying assumption of Eq. 4 (See B.3 for details). The extended influence function in Eq. 12, which replaces the inverse Hessian with its pseudo-inverse, circumvents this instability. Pruning the 92 negative directions from the Hessian (keeping 1193 positive eigenvalues, setting the others to zero, and inverting only the positive ones) drives the self-rank to a perfect $0.0 \pm 0.0$. The rank stays perfect as we aggressively truncate

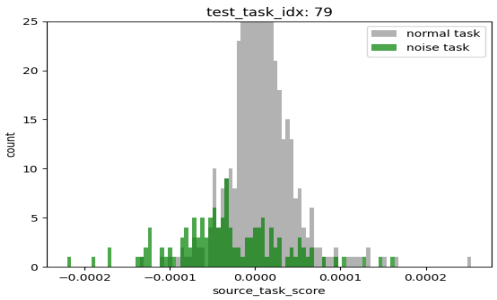



Figure 4: Example of training task score distribution (synthetic dataset).

the spectrum down to 1024, 512, 256, 128, and 64. Only when we keep 32, 16 or 8 eigenvalues does the ranking degrade ($0.0 \pm 0.2$, $2.0 \pm 3.2$ and $8.6 \pm 9.1$, respectively). Thus, we observe that discarding too many eigenvalues degrades the self-rank, underscoring the need for an appropriate estimate of the Hessian's eigenspace to maintain reliable task discrimination. Also, as a sanity check, we see that reducing the similarity of training and test tasks leads to a degradation of the self-ranks (See Supplement B.3 for details).

## Distinction of Normal and Noise Task Distributions

To validate property 2, we prepare three training tasksets that include a subpopulation with greater or less similarity to the test tasks than the rest of the training tasks. In each case, we enforce the desired similarity by replacing a subset of training tasks with tasks consisting of noise data.

**Classification of Synthetic Data** To examine the method with the exact form of Eq. 4, we employ a lightweight network and tasksets that the network can learn easily. We create tasksets of clustered data points in a two-dimensional space. Using Gaussian distributions, we first sample cluster centers around the origin of the plane and then the members of each cluster around its center and assign a unique label to each cluster. Noise tasks consist of data points sampled around the origin and assigned random labels.

Figure 4 shows the results of a 3-layer fully connected network (63 parameters) trained with 1024 training tasks that include 128 noise tasks in the 3-ways-5-shots problem setting. We observe that the normal and noise tasks follow different distributions. To formalize the intuitive similarity between two distributions, we define *proper order* as an ordering that preserves that similarity (e.g., subpopulations sharing the same subclass are more similar). Tests that result in such a relationship are referred to as *tests with the proper order* or *proper tests*. We score the training tasks using the influence function evaluated over 128 test losses and observe that 113 tests result in the proper order of the training task distributions in terms of their mean values. According to the binomial test, this count exceeds the average count ($= 64$) of the binomial distribution by $8.7\sigma$, satisfying property 2 statistically. See Supplement B.4 for details.

| learning method | dataset | proper tests [count] | proper tests [$\sigma$] |
|---|---|---|---|
| MAML | MI | 85.6±11.1 | 3.8±2.0 $\sigma$ |
| MAML | OM | 94.0±6.0 | 5.3±1.1 $\sigma$ |
| Protonet | MI | 95.6±5.3 | 5.6±0.9 $\sigma$ |
| Protonet | OM | 81.2±5.5 | 3.0±1.0 $\sigma$ |

Table 1: Experiment of distinguishing noise tasks and normal task distributions. MI and OM represent MiniImagenet and Omniglot respectively. The standard deviation $\sigma$ under the hypothesis of the random ordering is $\sqrt{0.5 \times 0.5 \times 128} \sim 5.66$. $\pm$ means the average and standard deviation across 5 runs of training.

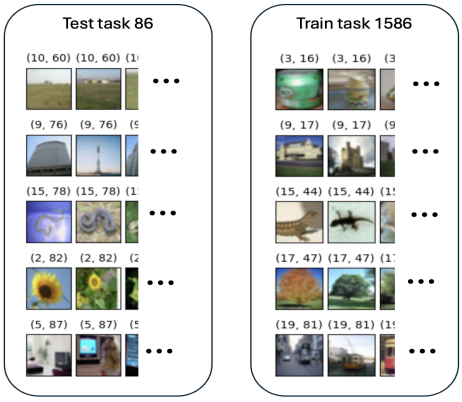



Figure 5: Examples of FC60 tasks. Left: A test task (accuracy=0.96). Right: the training tasks ranked 1st by TLXML. The pair of labels on each image represents the semantic labels (superclass, subclass) obtained from CIFAR100.

**MiniImagenet and Omniglot.** We validate property 2 on realistic tasks by using a large network with sufficient capacity to generalize. Given the added computational complexity, we adopt the approximation method in Eq. 11. For each dataset, using 8192 training tasks, We train a network with 3-conv layers using MAML (∼20k parameters) and a 4-conv-layer feature extractor using Protonet (∼28k parameters). For each training taskset, we replace the image tensors of 1024 tasks with uniform noise tensors of the same shape (noise images). For each training setup, we evaluate the influence of the training tasks on 128 test loss values using Eq. 9 with the projected influence on the meta-parameters (Eq. 12) and the GN matrix approximation (Eq. 11).

Table 1 shows the number of test tasks in which the proper ordering holds; A two-sided binomial test rejects the null hypothesis of random ordering, suggesting that our scoring method satisfies property 2 statistically. See Supplement B.4 for the results with different training settings.

## Consistency with Semantic Similarity

We also validate TLXML in terms of semantic similarities between training and test tasks. We define a new image dataset, *the FC60 dataset*, in which each image is assigned hierarchical labels specifying the super- and sub-

| learning method | overlap | train tasks filtered | proper tests |
|---|---|---|---|
| MAML | 1 | 7522±314 | 618 (+6.6$\sigma$) |
| MAML | 2 | 4781±718 | 676 (+10.3$\sigma$) |
| MAML | 3 | 1508±422 | 688 (+11.0$\sigma$) |
| MAML | 4 | 171±69 | 637 (+7.8$\sigma$) |
| Protonet | 1 | 7522±314 | 658 (+9.1$\sigma$) |
| Protonet | 2 | 4781±718 | 672 (+10.0$\sigma$) |
| Protonet | 3 | 1508±422 | 703 (+11.9$\sigma$) |
| Protonet | 4 | 171±69 | 644 (+8.3$\sigma$) |

Table 2: Experiment of distinguishing training tasks with superclasses shared with test tasks from other training tasks. $\pm$ means the average and standard deviation across 1024 tasks. The standard deviation $\sigma$ under the hypothesis of the random ordering is $\sqrt{0.5 \times 0.5 \times 1024} = 16$.

| **Dataset (train→test)** | **# tasks** | Method | Accuracy |
|---|---|---|---|
| FC60→FC60 | 2048 | $\xi = -8$ | 0.659±0.009 |
| ($0.663 \pm 0.004$) | 2048 | $\xi = -4$ | **0.674±0.007** |
| | 2048 | $\xi = -2$ | **0.672±0.007** |
| | 2048 | $\xi = -1$ | 0.669±0.004 |
| | 2048 | leave-out | 0.667±0.01 |

Table 3: Test accuracies after a single TLXML-guided update to a MAML model with blocked tasks. $\pm$ means the average and standard deviation across 5 runs of MAML training. Bold values outperform the MAML baseline (shown in the bracket) with a unpaired Welch two-sample $t$-test, $p < 0.05$.

class of the image. We use the FC100 dataset (Oreshkin, Rodríguez López, and Lacoste 2018), which curates meta-learning task sets from CIFAR-100 (Krizhevsky 2009). The tasks in FC100 are grouped based on their superclass, and the train, validation and test splits are constructed so that no superclass appears in more than one split. We split the FC100 training taskset by dividing the subclasses of each 12 superclasses into separate training and test splits, resulting in train/test splits that have 60 subclasses in total and share 12 superclasses. Figure 5 shows examples of a test and training task in FC60 dataset.

We train a network with 3-conv layers using MAML and a 4-conv-layer feature extractor using Protonet, using 8192 training tasks from FC60. Table 2 shows the results of counting the proper tests when the training subpopulation is defined by how many superclasses it shares with the test task, illustrating that TLXML distinguishes training tasks that share semantic properties with test tasks from the other tasks. See Supplement B.5 for details.

### 5.2 Post-Convergence One-Step Update

We examine the effect of the one-step update Eq. 13 from the convergence point of meta-learning. We utilize 8192 training tasks from FC60 and train a network with 3-conv layers using MAML. Each training task is scored with the influence function for the average test loss across 1024 test tasks.

Table 3 shows the effects of blocking training tasks with one-step updates (and leave-out for comparison) of the trained parameters on the test accuracies. We observe that the test accuracies are improved by one-step update with suitable shift values ($\xi$), whereas leave-out retraining fails to yield a statistically significant gain.

| **Dataset (train→test)** | **# tasks** | $\xi = -4$ **(block)** |
|---|---|---|
| FC60→FC60 ($0.663 \pm 0.004$) | 2048 | **0.674±0.007** |
| FC60→FC100 ($0.429 \pm 0.005$) | 2048 | **0.449±0.011** |
| **Dataset (train→test)** | **# tasks** | $\xi = 4$ **(enhance)** |
| FC60→FC60 ($0.663 \pm 0.004$) | 2048 | **0.676±0.008** |
| FC60→FC100 ($0.429 \pm 0.05$) | 2048 | 0.429±0.010 |

Table 4: Test accuracies after a single TLXML-guided update to a MAML model with blocked and enhanced tasks. See Table 3 for notation.

Table 4 shows the effects of both blocking and enhancing training tasks. The table evaluates the impact on the test accuracies in two cases: test datasets with and without shared superclasses (FC60 testset and FC100 testset, respectively). We observe the improvement in those combinations of blocked/enhanced and the two testsets, except the case of enhancing the training tasks for the FC100 testset. Even in that case, other one-step update parameters boost performance (see Supplement B.6 for details).

## 6 Discussion and Conclusion

This paper introduced TLXML, a method for quantifying the impact of meta-learning tasks. We reduced its computational cost using the GN matrix approximation and handled flat directions around the convergence point using the pseudo-inverse Hessian. Experiments suggest that TLXML offers explanations at the task level properly, leading to actionable insights and improved performance in downstream tasks.

Extending TLXML beyond classification tasks, e.g., regression and Reinforcement Learning (RL), is expected to be straightforward, as its formulation through gradient is almost agnostic to the differences in the learning objectives. For example, it is plausible that an RL policy can be improved using a similar technique. Moreover, because TLXML possesses an aspect of task similarity, future work should explore its relation to general notions, such as task embedding space, out-of-distribution awareness, or domain adaptations, not just the usage as a similarity measure between training and test tasks in each single dataset.

One limitation is the assumption of a local minimum of the loss function in the definitions of the influence functions. Future work could aim to design influence functions compatible with early stopping techniques. We also note that the current evaluation is a purely quantitative one conducted with standard benchmark datasets; investigating how non-experts would leverage TLXML is also a worthwhile future direction.

## 7 Acknowledgments

The authors thank Sheila Schoepp for helpful discussions at the early stages of this work.

Thank you for reading these instructions carefully. We look forward to receiving your electronic files!

## References


Adebayo, J.; Muelly, M.; Abelson, H.; and Kim, B. 2022. Post hoc explanations may be ineffective for detecting unknown spurious correlation. In *International conference on learning representations*.

Agarwal, M.; Yurochkin, M.; and Sun, Y. 2021. On sensitivity of meta-learning to support data. In Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*.

Alaa, A.; and Van Der Schaar, M. 2020. Discriminative jackknife: Quantifying uncertainty in deep learning via higher-order influence functions. In *International Conference on Machine Learning*, 165–174. PMLR.

Arnold, S. M. R.; Mahajan, P.; Datta, D.; Bunner, I.; and Zarkias, K. S. 2020. learn2learn: A Library for Meta-Learning Research.

Barshan, E.; Brunet, M.-E.; and Dziugaite, G. K. 2020. RelatIF: Identifying Explanatory Training Samples via Relative Influence. In Chiappa, S.; and Calandra, R., eds., *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, 1899–1909. PMLR.

Chhabra, A.; Li, P.; Mohapatra, P.; and Liu, H. 2024. " What Data Benefits My Classifier?" Enhancing Model Performance and Interpretability through Influence-Based Data Selection. In *The Twelfth International Conference on Learning Representations*.

Cohen, G.; Sapiro, G.; and Giryes, R. 2020. Detecting adversarial samples using influence functions and nearest neighbors. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 14453–14462.

Cook, R. D.; and Weisberg, S. 1980. Characterizations of an Empirical Influence Function for Detecting Influential Cases in Regression. *Technometrics: a journal of statistics for the physical, chemical, and engineering sciences*, 22.

Csurka, G.; Dance, C. R.; Fan, L.; Willamowski, J.; and Bray, C. 2004. Visual categorization with bags of keypoints. https://people.eecs.berkeley.edu/~efros/courses/AP06/Papers/csurka-eccv-04.pdf. Accessed: 2024-11-26.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In Precup, D.; and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 1126–1135. PMLR.

Goldblum, M.; Fowl, L.; and Goldstein, T. 2020. Adversarially Robust Few-Shot Learning: A Meta-Learning Approach. *Advances in Neural Information Processing Systems*, 17886–17895.

Hampel, F. R. 1974. The influence curve and its role in robust estimation. *Journal of the american statistical association*, 69(346): 383–393.

Han, X.; and Tsvetkov, Y. 2020. Fortifying Toxic Speech Detectors Against Veiled Toxicity. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 7732–7739.

Han, X.; Wallace, B. C.; and Tsvetkov, Y. 2020. Explaining Black Box Predictions and Unveiling Data Artifacts through Influence Functions. In Jurafsky, D.; Chai, J.; Schluter, N.; and Tetreault, J. R., eds., *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, 5553–5563. Association for Computational Linguistics.

Hospedales, T. M.; Antoniou, A.; Micaelli, P.; and Storkey, A. J. 2021. Meta-Learning in Neural Networks: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.*

Khanna, R.; Kim, B.; Ghosh, J.; and Koyejo, S. 2019. Interpreting Black Box Predictions using Fisher Kernels. In Chaudhuri, K.; and Sugiyama, M., eds., *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, 3382–3390. PMLR.

Khattar, V.; Ding, Y.; Sel, B.; Lavaei, J.; and Jin, M. 2024. A CMDP-within-online framework for meta-safe reinforcement learning. *arXiv preprint arXiv:2405.16601*.

Koh, P. W.; and Liang, P. 2017. Understanding Black-box Predictions via Influence Functions. *Proceedings of the 34th International Conference on Machine Learning*, 70: 1885–1894.

Koh, P. W. W.; Ang, K.-S.; Teo, H.; and Liang, P. S. 2019. On the accuracy of influence functions for measuring group effects. *Advances in neural information processing systems*, 32.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images.

Lake, B. M.; Salakhutdinov, R.; and Tenenbaum, J. B. 2015. Human-level concept learning through probabilistic program induction. *Science*, 350(6266): 1332–1338.

Li, D.; Yang, Y.; Song, Y.-Z.; and Hospedales, T. 2018. Learning to generalize: Meta-learning for domain generalization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Lin, Z.; Thomas, G.; Yang, G.; and Ma, T. 2020. Model-based adversarial meta-reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 10161–10173.

Lowe, D. G. 1999. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, 1150–1157 vol.2. IEEE.

Lowe, D. G. 2004. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*.

Lu, C.; Wu, Y.; Hernández-Lobato, J. M.; and Schölkopf, B. 2021. Invariant causal representation learning for out-of-distribution generalization. In *International Conference on Learning Representations*.

Mann, K. S.; Schneider, S.; Chiappa, A.; Lee, J. H.; Bethge, M.; Mathis, A.; and Mathis, M. W. 2021. Out-of-distribution

generalization of internal models is correlated with reward. In *Self-Supervision for Reinforcement Learning Workshop-ICLR*, volume 2021.

Martens, J. 2020. New Insights and Perspectives on the Natural Gradient Method. *J. Mach. Learn. Res.*, 21(146): 1–76.

Mouli, S. C.; Alam, M.; and Ribeiro, B. 2024. MetaPhysiCa: Improving OOD Robustness in Physics-informed Machine Learning. In *The Twelfth International Conference on Learning Representations*.

Ni, R.; Goldblum, M.; Sharaf, A.; Kong, K.; and Goldstein, T. 2021a. Data augmentation for meta-learning. In *International Conference on Machine Learning*, 8152–8161. PMLR.

Ni, R.; Goldblum, M.; Sharaf, A.; Kong, K.; and Goldstein, T. 2021b. Data Augmentation for Meta-Learning. In Meila, M.; and Zhang, T., eds., *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, 8152–8161. PMLR.

Oreshkin, B.; Rodríguez López, P.; and Lacoste, A. 2018. TADAM: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 1135–1144.

Shao, X.; Wang, H.; Zhu, X.; Xiong, F.; Mu, T.; and Zhang, Y. 2023. EFFECT: Explainable framework for meta-learning in automatic classification algorithm selection. *Information Sciences*, 622: 211–234.

Shu, Y.; Cao, Z.; Wang, C.; Wang, J.; and Long, M. 2021. Open domain generalization with domain-augmented meta-learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 9624–9633.

Sijben, E.; Jansen, J.; Bosman, P.; and Alderliesten, T. 2024. Function Class Learning with Genetic Programming: Towards Explainable Meta Learning for Tumor Growth Functionals. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1354–1362.

Snell, J.; Swersky, K.; and Zemel, R. 2017. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30.

Song, Y.; and Jeong, H. 2024. Towards cross domain generalization of Hamiltonian representation via meta learning. In *ICLR 2024, The Twelfth International Conference on Learning Representations*, 12319–12338. ICLR.

Vinyals, O.; Blundell, C.; Lillicrap, T.; Kavukcuoglu, K.; and Wierstra, D. 2016. Matching networks for one shot learning. *Adv. Neural Inf. Process. Syst.*, 3630–3638.

Wen, L.; Zhang, S.; Tseng, H. E.; Singh, B.; Filev, D.; and Peng, H. 2022. Improved Robustness and Safety for Pre-Adaptation of Meta Reinforcement Learning with Prior Regularization. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 8987–8994. IEEE.

Woźnica, K.; and Biecek, P. 2021. Towards Explainable Meta-learning. In *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 505–520. Springer International Publishing.

Xu, H.; Li, Y.; Liu, X.; Liu, H.; and Tang, J. 2021. Yet Meta Learning Can Adapt Fast, it Can Also Break Easily. In *Proceedings of the 2021 SIAM International Conference on Data Mining*.

Yao, Y.; Liu, Z.; Cen, Z.; Zhu, J.; Yu, W.; Zhang, T.; and Zhao, D. 2024. Constraint-conditioned policy optimization for versatile safe reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Zhang, J.; Cheung, B.; Finn, C.; Levine, S.; and Jayaraman, D. 2020. Cautious adaptation for reinforcement learning in safety-critical settings. In *International Conference on Machine Learning*, 11055–11065. PMLR.

Zügner, D.; and Günnemann, S. 2019. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *International Conference on Learning Representations*.

# Technical appendices

## A Method details

### A.1 Meta-learning Algorithms

We present the explicit forms of the adaptation algorithm $\mathcal{A}$ used for MAML and Protonet.

In MAML, the initial values $\theta_0$ of the network weights serve as the meta-parameters, and $\mathcal{A}$ represents a one-step gradient descent update of the weights with a fixed learning rate $\alpha$:

$$\mathcal{A}(\mathcal{D}^{(i)\text{train}}, \mathcal{L}^i, \theta_0) = \theta_0 - \alpha\, \partial_\theta \mathcal{L}^i(\mathcal{D}^{(i)\text{train}}, f_\theta)\Big|_{\theta=\theta_0}.$$

In Protonet, the meta-parameters are the weights of a feature extractor $f_\theta$ and the adaptation $\mathcal{A}$ does not involve the loss function. It passes the weights $\theta$ without any modification and calculates the feature centroid $c_k$ for each class $k$ based on the support set $S_k \in \mathcal{D}$:

$$\theta = \mathcal{A}_\theta(\mathcal{D}^{(i)\text{train}}, \mathcal{L}^i, \theta),$$
$$c_k = \mathcal{A}_k(\mathcal{D}^{(i)\text{train}}, \mathcal{L}^i, \theta) = \frac{1}{|S_k^{(i)}|} \sum_{(x,y)\in S_k^{(i)}} f_\theta(x).$$

The class prediction for each data point $x$ is given by:

$$P_\theta(i|x) = \frac{\exp d(f_\theta(x), c_i)}{\sum_k \exp d(f_\theta(x), c_k)},$$

where $d$ is a distance measure(e.g. Euclidean distance).

### A.2 Implicit differentiation

The second equation in each of Eq. 1, Eq. 4, and Eq. 12 are derived immediately from the following property.

*Property 1*: If a vector parameter $\hat{\theta}_\epsilon$ is parametrized by a scalar parameter $\epsilon$ in a way that the local maximum or the local minimum condition of a second-order differentiable

function $L(\theta, \epsilon)$ is satisfied for each value of $\epsilon$, then the derivative of $\hat{\theta}_\epsilon$ with respect to $\epsilon$ satisfies:

$$\frac{\partial^2 L(\theta,\epsilon)}{\partial\theta\partial\theta}\bigg|_{\theta=\hat{\theta}_\epsilon} \frac{d\hat{\theta}_\epsilon}{d\epsilon} = -\frac{\partial L\left(\theta,\epsilon\right)}{\partial\theta\partial\epsilon}\bigg|_{\theta=\hat{\theta}_\epsilon}. \tag{14}$$

The proof is done almost trivially by differentiating the local maximum or minimum condition

$$\frac{\partial L\left(\theta,\epsilon\right)}{\partial\theta}\bigg|_{\theta=\hat{\theta}_\epsilon} = 0 \tag{15}$$

with respect to $\epsilon$ and apply the chain rule. Note that if the matrix $\partial\partial L$ in Eq. 14 is invertible, we can solve the equation to obtain the $\epsilon$-derivative of $\hat{\theta}_\epsilon$. Note also that we do not assume $\hat{\theta}_\epsilon$ to be the unique solution of Eq. 15 and Eq. 14 is true for any parametrization of $\theta$ with $\epsilon$ that satisfies Eq. 15.

### A.3 Task Grouping

In some cases, the abstraction of task-level explanations is insufficient, and explanations based on task groups are more suitable. This requirement occurs when the tasks used in the training are similar to each other for human intuition. For example, when an image recognition model is trained with task augmentation(see Ni et al. (2021b) for terminologies of data augmentation for meta-learning), e.g., flipping, rotating, or distorting the images in original tasks, the influence of each deformed task is not of interest; rather, the influence of the task group generated from each original task is of interest.

We extend the definition of influence functions to the task-group level by considering a common perturbation $\epsilon$ in the losses of tasks within a task group $\mathcal{G}^J{=}\{\mathcal{T}^{j_0}, \mathcal{T}^{j_1}, \cdots\}$:

$$\hat{\omega}_\epsilon^J = \arg\min_\omega \frac{1}{M}\Bigg[ \sum_{i=1}^{M} \mathcal{L}^i(\mathcal{D}^{(i)\text{test}}, f_{\hat{\theta}^i}) + \epsilon \sum_{\mathcal{T}^j\in\mathcal{G}^J} \mathcal{L}^j(\mathcal{D}^{(j)\text{test}}, f_{\hat{\theta}^j}) \Bigg] \tag{16}$$

with $\hat{\theta}^i{=}\mathcal{A}(\mathcal{D}^{(i)\text{train}}, \mathcal{L}^i, \omega)$,

which modifies the influence function in Eq. 4 as:

$$I^{\text{meta}}(J) \stackrel{\text{def}}{=} \frac{d\hat{\omega}_\epsilon^J}{d\epsilon}\bigg|_{\epsilon=0} = \sum_{\mathcal{T}^j\in\mathcal{G}^J} I^{\text{meta}}(j). \tag{17}$$

Eq. 7 and Eq. 9 are only affected by replacing the task index $j$ with the task-group index $J$. The derivation of Eq. 17 is done by directly applying the argument in A.2

### A.4 Third-order tensors in influence functions

Here, we explain how the computational cost of $\mathcal{O}(pq^2)$ arises in evaluating the influence function in Eq. 4. This is due to the third-order tensors which appear in the intermediate process of evaluating the Hessian:

$$H = \frac{1}{M}\sum_{i=1}^{M} \frac{\partial^2 \mathcal{L}^i(\mathcal{D}^{(i)\text{test}}, f_{\hat{\theta}^i(\omega)})}{\partial\omega\partial\omega} \tag{18}$$

$$= \frac{1}{M}\sum_{i=1}^{M}\Bigg[ \left(\frac{\partial\hat{\theta}^i(\omega)}{\partial\omega}\right)^{\mathrm{T}} \frac{\partial^2\mathcal{L}^i(\mathcal{D}^{(i)\text{test}}, f_\theta)}{\partial\theta\partial\theta}\bigg|_{\theta=\hat{\theta}^i(\omega)} \frac{\partial\hat{\theta}^i(\omega)}{\partial\omega} + \frac{\partial\mathcal{L}^i(\mathcal{D}^{(i)\text{test}}, f_\theta)}{\partial\theta}\bigg|_{\theta=\hat{\theta}^i(\omega)} \frac{\partial^2\hat{\theta}^i(\omega)}{\partial\omega\partial\omega}\Bigg] \tag{19}$$

where $\hat{\theta}^i = \mathcal{A}(\mathcal{L}^{(i)\text{train}}, \mathcal{L}^i, \omega)$. Because $\hat{\theta}^i$ and $\omega$ are $p$-dimensional and $q$-dimensional respectively, the second-order derivative $\partial\partial\hat{\theta}^i$ in the last term is the third-order tensor of $pq^2$ elements. This tensor also appears in the evaluation of the second-order derivative of the network output with respect to $\omega$. In the case of MAML, this tensor is in the form of a third-order derivative:

$$\frac{\partial^2\hat{\theta}^i(\theta_0)}{\partial\theta_0\partial\theta_0} = \frac{\partial^2}{\partial\theta_0\partial\theta_0}\mathcal{A}(\mathcal{D}^{(i)\text{train}}, \mathcal{L}^i, \theta_0) = -\alpha\frac{\partial^3}{\partial\theta_0\partial\theta_0\partial\theta_0}\mathcal{L}^i\left(\mathcal{D}^{(i)\text{train}}, f_{\theta_0}\right).$$

### A.5 Relations Among KL-divergence, Cross-Entropy, and Fisher Information Metric

For the reader's convenience, we present basic facts related to the approximation method argued in Section 4.2.

**Variant expressions of Hessian** The cross-entropy $L$ between two probability distributions $P\left(X|\omega^*\right)$, $P\left(X|\omega\right)$ parametrized by $\omega^*$ and $\omega$, is equivalent to the Kullback–Leibler (KL) divergence up to a $\omega$-independent term:

$$D_{KL}\left(P\left(X|\omega^*\right), P\left(X|\omega\right)\right) = L\left(P\left(X|\omega^*\right), P\left(X|\omega\right)\right) + \int P(X|\omega^*)\log P(X|\omega^*)dX.$$

Therefore, the second-order derivatives of them with respect to $\omega$ are identical:

$$\frac{\partial^2}{\partial\omega\partial\omega}D_{KL} = \frac{\partial^2}{\partial\omega\partial\omega}L$$

Furthermore, considering the Taylor expansion of $D_{KL}$ with $\Delta\omega{=}\omega{-}\omega^*$

$$
\begin{aligned}
&D_{KL}\left(P\left(X|\omega^*\right), P\left(X|\omega\right)\right) \\
&\sim -\sum_{\mu} \Delta\omega^{\mu} \left[\int \partial_{\mu} P\left(X|\omega^*\right) dX\right] \\
&+ \frac{1}{2} \sum_{\mu\nu} \Delta\omega^{\mu}\Delta\omega^{\nu} \left[ \right. \\
&\left. \int -\partial_{\mu}\partial_{\nu} P\left(X|\omega^*\right) + \frac{\partial_{\mu} P\left(X|\omega^*\right) \partial_{\nu} P\left(X|\omega^*\right)^2}{P\left(X|\omega^*\right)} dX\right] \\
&= \frac{1}{2} \sum_{\mu\nu} \Delta\omega^{\mu}\Delta\omega^{\nu} \\
&\times \mathrm{E}_{X\sim P(X|\omega^*)} \left[\partial_{\mu} \log P\left(X|\omega^*\right) \partial_{\nu} \log P\left(X|\omega^*\right)\right] \\
&\equiv \frac{1}{2} \sum_{\mu\nu} g_{\mu\nu}\left(\omega^*\right) \Delta\omega^{\mu}\Delta\omega^{\nu},
\end{aligned}
$$

we see that

$$
\left.\frac{\partial^2}{\partial\omega_{\mu}\partial\omega_{\nu}} D_{KL}\right|_{\omega=\omega^*} = \left.\frac{\partial^2}{\partial\omega_{\mu}\partial\omega_{\nu}} L\right|_{\omega=\omega^*} = g_{\mu\nu}\left(\omega^*\right) \tag{20}
$$

**approximations by empirical sums** Let us consider the case that $X = (x, c)$ is the pair of a network input $x_n$ and a class label $c_n$ and $P$ is the composition of soft-max function $\sigma$ and the network output $\boldsymbol{y}_n \equiv f_{\omega}(x_n)$. Assuming that sampled data accurately approximate the distributions, we obtain the expressions of the Fisher information metric in the form of an empirical sum:

$$
\begin{aligned}
&g_{\mu\nu}\left(\omega^*\right) \\
&= \mathrm{E}_{X\sim P(X|\omega^*)} \left[\partial_{\mu} \log P\left(X|\omega^*\right) \partial_{\nu} \log P\left(X|\omega^*\right)\right] \\
&= \mathrm{E}_{(c,x)\sim P_{\omega^*}(c|x)P(x)} \left[ \right. \\
&\quad \left. \partial_{\mu} \log\left(P_{\omega}\left(c|x\right) P(x)\right) \partial_{\nu} \log\left(P_{\omega}\left(c|x\right) P(x)\right)\right]|_{\omega=\omega^*} \\
&= \mathrm{E}_{(c,x)\sim P_{\omega^*}(c|x)P(x)} \left[ \right. \\
&\quad \left. \partial_{\mu} \log\left(P_{\omega}\left(c|x\right)\right) \partial_{\nu} \log\left(P_{\omega}\left(c|x\right)\right)\right]|_{\omega=\omega^*} \\
&\sim \left.\sum_{ni} \sigma_i\left(\boldsymbol{y}_n\right) \left[\partial_{\mu} \log\left(\sigma_i\left(\boldsymbol{y}_n\right)\right) \partial_{\nu} \log\left(\sigma_i\left(\boldsymbol{y}_n\right)\right)\right]\right|_{\omega=\omega^*} \\
&= \sum_{nijk} \sigma_i\left(\boldsymbol{y}_n\right) \left(\delta_{ik} - \sigma_k\left(\boldsymbol{y}_n\right)\right) \left(\delta_{ij} - \sigma_j\left(\boldsymbol{y}_n\right)\right) \\
&\quad \left.\times \frac{\partial y_{nk}}{\partial\omega_{\mu}} \frac{\partial y_{nj}}{\partial\omega_{\nu}}\right|_{\omega=\omega^*} \\
&= \left.\sum_{nkj} \sigma_k\left(\boldsymbol{y}_n\right) \left(\delta_{kj} - \sigma_j\left(\boldsymbol{y}_n\right)\right) \frac{\partial y_{nk}}{\partial\omega_{\mu}} \frac{\partial y_{nj}}{\partial\omega_{\nu}}\right|_{\omega=\omega^*}.
\end{aligned}
$$

To express the Hessian in a similar way, we denote the target vector of a sample $x_n$ as $t_{ni}$. Then:

$$
\begin{aligned}
&\frac{\partial^2}{\partial\omega\partial\omega} L \qquad (21) \\
&= -\frac{\partial^2}{\partial\omega\partial\omega} \sum_{c} \int P_{\omega^*}\left(c|x\right) P(x) \log\left[P_{\omega}\left(c|x\right) P(x)\right] dx \\
&\sim -\frac{\partial^2}{\partial\omega\partial\omega} \sum_{ni} t_{ni} \log\left[P_{\omega}\left(i|x_n\right) P(x_n)\right] \\
&= -\frac{\partial^2}{\partial\omega\partial\omega} \sum_{ni} t_{ni} \log \sigma_i(\boldsymbol{y_n}) \\
&= -\frac{\partial}{\partial\omega} \sum_{nik} t_{ni} \left[\delta_{ik} - \sigma_k(\boldsymbol{y_n})\right] \frac{\partial y_{nk}}{\partial\omega} \\
&= \sum_{nijk} t_{ni}\sigma_k(\boldsymbol{y_n}) \left(\delta_{kj} - \sigma_j(\boldsymbol{y_n})\right) \frac{\partial y_{nk}}{\partial\omega} \frac{\partial y_{nj}}{\partial\omega} \qquad (22) \\
&\quad - \sum_{nik} t_{ni} \left[\delta_{ik} - \sigma_i(\boldsymbol{y_n})\right] \frac{\partial^2 y_{nk}}{\partial\omega\partial\omega} \\
&= \sum_{njk} \sigma_k(\boldsymbol{y_n}) \left(\delta_{kj} - \sigma_j(\boldsymbol{y_n})\right) \frac{\partial y_{nk}}{\partial\omega} \frac{\partial y_{nj}}{\partial\omega} \qquad (23) \\
&\quad - \sum_{nik} t_{ni} \left[\delta_{ik} - \sigma_i(\boldsymbol{y_n})\right] \frac{\partial^2 y_{nk}}{\partial\omega\partial\omega} \\
&= g(\omega) - \sum_{nik} t_{ni} \left[\delta_{ik} - \sigma_i(\boldsymbol{y_n})\right] \frac{\partial^2 y_{nk}}{\partial\omega\partial\omega}. \qquad (24)
\end{aligned}
$$

The second-to-last equation proves Eq. 10. From the last equation, we see that if the distributions $P(X|\omega^*)$ and $P(X|\omega)$ are accurately approximated by the training data samples and the model's outputs, respectively, the first term in Eq. 10 becomes equivalent to the Fisher information metric evaluated at $\omega$. By comparing Eq. 20 and Eq. 24, we see that if $\omega$ is near to $\omega^*$, the second term of Eq. 24 drops, and the Hessian is well approximated by the Fisher information metric.

# B Experimental details

## B.1 Implementation

We comment on the implementation for computing the pseudo-inverse $H^+$ in Eq. 12 using the approximated Hessian (Eq. 11):

$$
\begin{aligned}
H &\sim \sum_{nj} \left(\mathbf{V}\right)_{\mu(nj)} \left(\mathbf{V}^{\mathrm{T}}\right)_{(nj)\nu} \\
&= VV^T
\end{aligned} \tag{25}
$$

where we define the matrix $V$ as the horizontal concatnation of $\mathbf{V}$s, and each block $\mathbf{V}$ satifies

$$
\begin{aligned}
&\sum_{j} \left(\mathbf{V}\right)_{\mu(nj)} \left(\mathbf{V}^{\mathrm{T}}\right)_{(nj)\nu} \\
&= \sum_{jk} \sigma_k\left(\boldsymbol{y}_n\right) \left(\delta_{kj} - \sigma_j\left(\boldsymbol{y}_n\right)\right) \frac{\partial y_{nk}}{\partial\omega} \frac{\partial y_{nj}}{\partial\omega} \quad \text{for } \forall n.
\end{aligned}
$$

As explained in the main paper, there exists an orthogonal system of vectors $\{v_i\}$ such that this matrix can be written as the summation of unnormalized projections associated with them: $VV^T = \sum v_i v_i^T$, and the pseudo-inverse of this summation is computed by adjusting the normalizations of the elements satisfying $|v_i| > 0$.

Note that this method is feasible if the number of independent columns in $V$ is small. Although the Gauss-Newton matrix approximation avoids the $\mathcal{O}(pq^2)$ computational cost, the large size of $V$ leads to storage/memory cost. [1] Fortunately, the number of independent columns in $V$ is expected to be small, and most columns are dropped as zero vectors. This number corresponds to the non-flat directions, representing the constraints imposed by the loss minimization condition, and typically, it is at most the number of training tasks. Therefore, we can implement the accumulation of the vector product elements in Eq. 25 by preparing a buffer and sequentially adding the elements $v_i v_i^T$ from each training task to it, orthogonalizing the elements inside it, and dropping zero elements.

In the experiments of Section 5.1, and 5.2 , in accumulation of the vector product elements, 8192 independent vectors are collected, and at each time when the number of vectors exceeds $8192 \times 2$ the columns of $V$ are orthogonalized, the vectors with the largest 8192 norms are kept, and the other vectors are dropped.

After that, to calculate the pseudo-inverse of the Hessian, we choose non-zero vector elements. To determine the number of vectors treated as non-zero, we calculate the pseudo-inverses under different assumptions on the number of positive eigenvalues. We assume that the Hessian has 64, 128, 256, 512, 1024, 2048, 4096, and 8192 positive eigenvalues. For each of them, we retain the assumed numbers of eigenvalues in descending order, treat the other elements as zero, compute the self-ranks, and choose the assumption that gives the smallest self-rank. If different numbers provide the same self-rank, the smallest one is chosen.

## B.2 Dataset

In the experiments below, we use the MiniImagenet and Omniglot datasets. Both are commonly used datasets for measuring model performance in the few-shot learning problem. In addition, to fill their shortcomings, we define the following two datasets.

**Synthetic dataset.** To validate the proposed method with the exact form of Eq. 12, we employ a lightweight network and a taskset that the network can learn easily. We create tasksets of clustered data points in a two-dimensional space. Using Gaussian distributions, we first sample cluster centers around the origin of the plane with a standard deviation of 1 and then the members of each cluster around its center with a standard deviation of 0.1, and assign a unique label to each cluster. We also use noise tasks consisting of data points sampled around the origin with a standard deviation of 1 and assigned random labels.

Fig. 6 shows examples of a normal task and a noise task in the 3-way-5-shot problem.

**FC60 dataset.** To investigate the relation between influence scores and semantic similarity, we need a dataset in which training and test tasks share semantic properties. We define a new image dataset, *the FC60 dataset*, in which each image is assigned hierarchical labels specifying the super- and subclass of the image. This is achieved by utilizing the FC100 dataset (Oreshkin, Rodríguez López, and Lacoste 2018), which curates few-shot-learning tasks from CIFAR-100 (Krizhevsky 2009). The tasks in FC100 are grouped based on the superclasses defined in CIFAR100, and the train, validation, and test splits are constructed in a way that no superclass appears in more than one split. We split the FC100 training taskset by dividing the subclasses of each 12 superclasses into training and test splits, resulting in train/test splits that have 60 subclasses in total and share 12 superclasses. This dataset is implemented based on the FC100 dataset implemented in learn2learn. It is achieved by applying filters on the superclasses at each acquisition of data.

Table 5 shows the superclasses and subclasses used in FC60. Fig. 7 shows examples of a test task and a training task.

## B.3 Distinction of Tasks

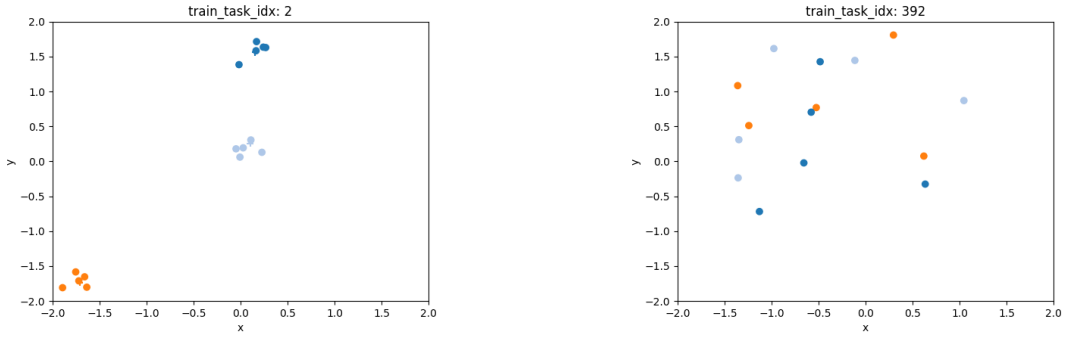
**Setup** We train a two-layer fully connected network with widths of 32 and 5, ReLU activations, and batch normalization layers(1,285 parameters) using 1000 meta-batches sampled from 128 training tasks using MAML. For the Mini-Imagenet dataset, the model's input is a 32-dimensional feature vector extracted from the image using the Bag-of-Visual-Words (Csurka et al. 2004) with SIFT descriptors (Lowe 1999, 2004) and k-means clustering. For the Omniglot dataset, it is a 36-dimensional feature vector extracted from the image by applying 2-dimensional FFT and clipping the $6\times6$ image at the center.

**MiniImagenet** In the experiments in Section 5.1 with MiniImagenet, we encountered negative eigenvalues of the Hessian. We provide some details here. Figure 8 shows 1285 eigenvalues arranged from the largest to the smallest. We can see that the positive eigenvalues are almost restricted to the first several hundred elements. We can also see negative eigenvalues in the tail.

In addition to the result mentioned in the main paper, as a sanity check, we checked that reducing the similarity of training and test tasks leads to a degradation of the self-ranks. We degraded the test tasks by darkening some of the images in the task and examined whether the ranks and scores of the originally identical training tasks get worse as the similarity decreases.

Figure 9 shows examples of the results with MiniImagenet. We observe that the ranks and scores tend to get worse as the darkness or the number of dark images in the test task increases. These are examples in the case of a small amount of Hessian pruning. We investigate the effects of pruning on

[1] For example, if the network has 100k parameters, the training taskset has 1000 tasks, and each task is a 5-way-5-shot problem, then the number of matrix elements is $p\times cnM = 10^5\times(5\times25\times1000)\sim10^{10}$, which makes it challenging to use floating-point numbers of 16, 32, or 64-bit precision.




(a) A normal task

(b) A noise task

Figure 6: Examples of synthetic tasks generated from Gaussian distributions.

| superclass | subclass | |
|---|---|---|
| | train | test |
| fish | [aquarium fish, flatfish, ray] | [shark, trout] |
| flowers | [orchids, poppies, roses] | [sunflowers, tulips] |
| food containers | [bottles, bowls, cans] | [cups, plates] |
| fruit and vegetables | [apples, mushrooms, oranges] | [pears, sweet peppers] |
| household electrical devices | [clock, keyboard, lamp] | [telephone, television] |
| household furniture | [bed, chair, couch] | [table, wardrobe] |
| large man-made outdoor things | [bridge, castle, house] | [road, skyscraper] |
| large natural outdoor scenes | [cloud, forest, mountain] | [plain, sea] |
| reptiles | [crocodile, dinosaur, lizard] | [snake, turtle] |
| trees | [maple, oak, palm] | [pine, willow] |
| vehicles 1 | [bicycle, bus, motorcycle] | [pickup truck, train] |
| vehicles 2 | [lawn-mower, rocket, streetcar] | [tank, tractor] |

Table 5: Superclasses and subclasses of FC60 dataset

the relation between degradation and self-rank. The third to sixth columns of Table 6 show the correlation coefficients between the degradation parameters and the self-rank and score of the originally identical training tasks. We observe that pruning increases the correlation values.

**Omniglot** We also conducted experiments with the Omniglot dataset. In this case, we encountered 428 negative eigenvalues of the Hessian. The effects of pruning on the self-rank are shown in the first column of Table 7. Tests with the degradation of those test tasks were also conducted. Again, we observe that the values of correlation coefficients are increased by pruning to some extent. However, those correlations are weaker than in the cases of MiniImagenet (Table 6). A possible reason for the weak correlation with $\alpha$ is that the images of handwritten digits in Omniglot have a small range of darkness, and the trained model is less sensitive to the darkness. A possible reason for the weak correlation with the ratio is that the tasks in Omniglot are easier to adapt to than in MiniImagenet.

### B.4 Distinction of Normal and Noise Task Distributions

**Setup.** For the networks used in the experiments below, we employ ReLu activation and batch normalization. When we use a CNN, we choose a network such that its number of parameters is around $\sim$20,000, and the test accuracy does not decrease continuously during the training with 8192 tasks from the dataset of each setting.

When we calculate an approximated Hessian (Eq. 11) after the training, we accumulate vector products over the training tasks with the buffer size set to 8192, and then truncate them before evaluating the influence scores. To determine the truncated number, we calculate the average self-rank over the training tasks with different assumptions on the number of elements(64, 128, 256, 512, 1024, 2048, 4096, and 8192). We select the assumption that gives the best self-rank. When different values provide the same self-rank (like $0 \pm 0$), we selected the assumption of the smallest number.

**Synthetic dataset** To examine the method with the exact form of Eq. 4, we employed a lightweight network and tasksets that the network can learn easily. Using 1024 training tasks of the 3-ways-5-shots problem, including 128 noise




(a) Test task




(b) 1st training task

Figure 7: Examples of FC60 tasks. Left: A test task (accuracy=0.96). Right: the training tasks ranked 1st by TLXML. The pair of labels on each image represents the semantic labels (superclass, subclass) obtained from CIFAR100. The superclass 9 (large man-made outdoor things) and 15 (reptiles) are shared by them.

| # eigenvalues | selfrank(avg±std) | correlation with degradation(avg±std) | | | |
|---|---|---|---|---|---|
| | | alpha/rank | alpha/score | ratio/rank | ratio/score |
| 1285 | 12.6±18.9 | 0.51±0.32 | -0.41±0.29 | 0.36±0.32 | -0.11±0.31 |
| 1193 | 0.0±0.0 | 0.69±0.21 | -0.69±0.22 | 0.46±0.30 | -0.15±0.34 |
| 512 | 0.0±0.0 | 0.71±0.12 | -0.96±0.06 | 0.63±0.09 | -0.89±0.13 |
| 256 | 0.0±0.0 | 0.71±0.11 | -0.95±0.08 | 0.62±0.11 | -0.88±0.14 |
| 128 | 0.0±0.0 | 0.72±0.10 | -0.94±0.04 | 0.63±0.10 | -0.86±0.15 |
| 64 | 0.0±0.0 | 0.71±0.12 | -0.92±0.06 | 0.66±0.12 | -0.77±0.20 |
| 32 | 0.0±0.2 | 0.72±0.16 | -0.85±0.12 | 0.68±0.14 | -0.68±0.22 |
| 16 | 2.0±3.2 | 0.67±0.22 | -0.69±0.20 | 0.61±0.22 | -0.46±0.27 |
| 8 | 8.6 ±9.1 | 0.55±0.26 | -0.53±0.23 | 0.47±0.27 | -0.29±0.31 |

Table 6: Effect of Hessian pruning(MiniImagenet). "# eigenvalues" denotes the number of eigenvalues treated as positive in computing the pseudo-inverse (the 1st row is the original Hessian). The 2nd row corresponds to the pruned Hessian, where all 92 negative eigenvalues are set to zero. The 2nd column shows the self-ranks in the tests without degradation. and the remaining columns show correlation coefficients between the two degradation parameters and the self-ranks or self-scores. Values are reported as means and standard deviations across the 128 tasks.

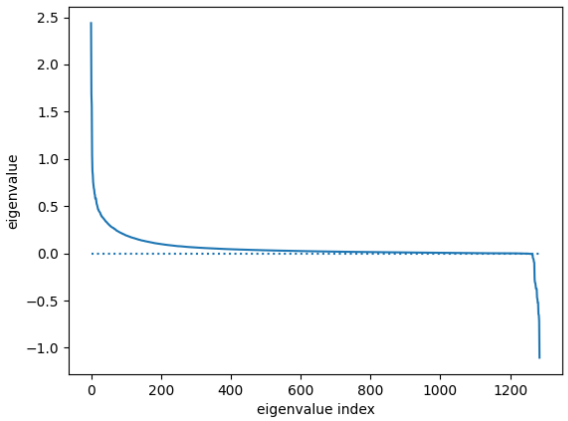


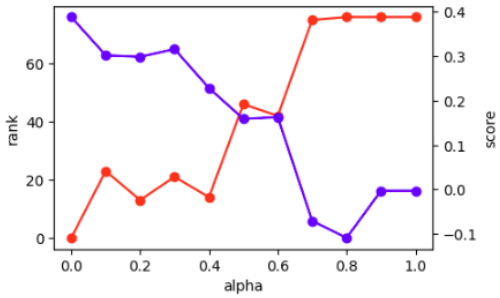

Figure 8: Eigenvalues of the Hessian before the pruning in B.3

tasks, in the synthetic dataset described above, we trained a 3-layer fully connected network with 2-dimensional input, 4-4 hidden dimensions, 3-dimensional output layers (63 parameters) with 30,000 meta-batches, and then calculated $I^{\mathrm{meta}}$. To determine the number of positive eigenvalues of an exact Hessian, the average self-ranks over the training tasks were calculated with different assumptions on the number of positive eigenvalues(4, 8, 16, 32, and 63). 32 was chosen as the one giving the best average self-rank. The result is described in the main paper.

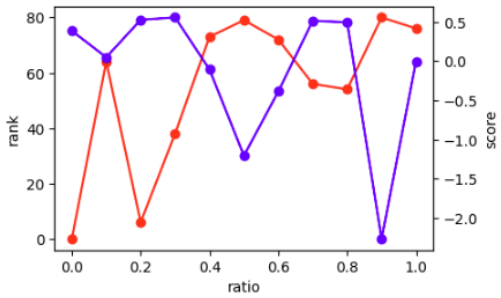
In addition, we calculated the scores with an approximated Hessian. To obtain the Hessian, we accumulated vector products over the training tasks with the buffer size set to 32. We obtained a result similar to the one for the exact Hessian and observed that 124 tests resulted in the proper order of the training task distributions in terms of their mean values. This count exceeds the average count ($= 64$) of the binomial distribution by $10.6\sigma$ under the hypothesis of random ordering. We also checked the correlations between

| # eigenvalues | selfrank(avg±std) | correlation with degradation(avg±std) | | | |
|---|---|---|---|---|---|
| | | alpha/rank | alpha/score | ratio/rank | ratio/score |
| 1413 | 66.0 ±36.9 | 0.15 ±0.48(21) | 0.06 ±0.50 | 0.02 ±0.34 | 0.03 ±0.23 |
| 985 | 7.8 ±10.0 | 0.48 ±0.12(2) | -0.37 ±0.33 | 0.25 ±0.28 | 0.01 ±0.23 |
| 512 | 0.8 ±5.0 | 0.50 ±0.00(1) | -0.50 ±0.00 | 0.35 ±0.26 | -0.15 ±0.23 |
| 256 | 1.6 ±6.3 | 0.50 ±0.00(1) | -0.50 ±0.00 | 0.34 ±0.27 | -0.12 ±0.25 |
| 128 | 2.9 ±7.4 | 0.50 ±0.00(1) | -0.50 ±0.00 | 0.36 ±0.27 | -0.11 ±0.24 |
| 64 | 4.3 ±8.0 | 0.50 ±0.00(1) | -0.50 ±0.00 | 0.36 ±0.26 | -0.13 ±0.23 |
| 32 | 6.4 ±8.8 | 0.50 ±0.00(1) | -0.49 ±0.09 | 0.33 ±0.26 | -0.08 ±0.24 |
| 16 | 8.4 ±10.0 | 0.50 ±0.00(1) | -0.50 ±0.00 | 0.32 ±0.30 | -0.08 ±0.25 |
| 8 | 11.8 ±12.5 | 0.50 ±0.00(4) | -0.50 ±0.00 | 0.29 ±0.29 | -0.06 ±0.24 |

Table 7: Effect of pruning the Hessian(Omniglot dataset). See table 6 for notations. The second row is the case that we only prune the 428 negative eigenvalues. There are cases in which increasing $\alpha$ does not change the ranks of the original training tasks. The numbers of those cases are shown in the brackets, and they are removed from the statistics because the correlation coefficients can not be defined for them.




(a) Example of the effect of increasing $\alpha$. $\alpha$=1 means all the images in the test task are dark.




(b) Example of the effect of increasing the ratio. ratio=1 means all the images in the test task are dark.

Figure 9: Test with degraded training tasks(MiniImagenet). The parameters $\alpha$ and *ratio* specify the darkness of images, and the number of dark images in each degraded task, respectively. The red and blue lines represent ranks and scores, respectively. Both examples were performed with the Hessian pruned to retain only the 1193 most significant eigenvalues.

the scores calculated with the exact and approximated Hessians. The Pearson correlation of the influence scores of the 8192 training tasks calculated for each of the 128 tests was 0.715±0.092.

**MiniImagenet and Omniglot.** We trained 3-conv×32-filter + 1-fully-connected layers networks (21,029 parameters for MiniImagenet, 19,178 parameters for Omniglot) using MAML and 4-conv ×32-filter feature extractors (28,896 parameters for MiniImagenet, 28,320 parameters for Omniglot) using Protonet. We used 128, 256, 512, 1024, 2048, 4096, and 8192 training tasks for each dataset, and combined them with 16, 32, 64, 128, 256, 512, and 1024 noise tasks, respectively, We created noise tasks by replacing the image tensors in each training task with uniform noise tensors of the same shape(noise images). In MAML training, the networks were trained with 80,000 meta-batches and 40,000 meta-batches for MiniImagenet and Omniglot, respectively. In Protonet training, the networks were trained with 20,000 meta-batches and 10,000 meta-batches for MiniImagenet and Omniglot, respectively. We conducted 5 runs of training with seeds 0, 1, 2, 3, and 42. We evaluated the influence scores of the training tasks on 128 test tasks, and counted the number of proper tests based on Eq. 7 , and 9 with the projected influence on the meta-parameters (Eq. 12) and the Gauss-Newton matrix approximation of the Hessian (Eq. 11).

Table 8, 9, 10, and 11 show the results of the experiments. We observe that the number of proper tests with a sufficient number of training tasks is large, which rejects the hypothesis of random ordering, suggesting that our scoring method satisfies property 2 statistically. Noting that the statistical significance is weak in the results of a small number of training tasks, we can see that Property 2 arises as an effect of generalization. Fig.10 demonstrates this as the relation between the test accuracy and the number of proper tests in the case of MAML and MiniImagenet. In that case, in the region of 128 and 256 training tasks, the number of proper tests is exceeded by the number of tests with the opposite order. This can be interpreted as reflecting the tendency that when overfitting occurs, only a few training tasks similar to the test task provide helpful information, and most of the regular training tasks become detrimental.

### B.5 Consistency with Semantics

Using 8192 training tasks in the FC60 dataset, we trained 3-conv×16-filter + 1-fully-connected layers networks (6469

| learning method | dataset | training tasks | | accuracy | | proper tests | |
|---|---|---|---|---|---|---|---|
| | | total | noise | test | train | [count] | [$\sigma$] |
| MAML | MI | 128 | 16 | 0.314±0.003 | 1.000±0.000 | 16.4±12.9 | -8.4±2.3 $\sigma$ |
| MAML | MI | 256 | 32 | 0.323±0.014 | 1.000±0.000 | 14.4±7.6 | -8.8±1.3 $\sigma$ |
| MAML | MI | 512 | 64 | 0.345±0.006 | 0.995±0.005 | 51.2±16.5 | -2.3±2.9 $\sigma$ |
| MAML | MI | 1024 | 128 | 0.368±0.007 | 0.902±0.049 | 99.6±20.5 | 6.3±3.6 $\sigma$ |
| MAML | MI | 2048 | 256 | 0.435±0.006 | 0.828±0.014 | 73.4±6.0 | 1.7±1.1 $\sigma$ |
| MAML | MI | 4096 | 512 | 0.520±0.012 | 0.762±0.008 | 89.2±10.3 | 4.5±1.8 $\sigma$ |
| MAML | MI | 8192 | 1024 | 0.552±0.012 | 0.717±0.007 | 85.6±11.1 | 3.8±2.0 $\sigma$ |

Table 8: Experiments with a CNN trained by MAML(MiniImagenet dataset, combined with noise image tasks). 128 test tasks were selected from the test taskset of pure MiniImagenet. The numbers of proper tests are shown in the second-to-last column for each training setting. The standard deviation $\sigma$ under the null-hypothesis of random ordering is $\sqrt{128 \times 0.5^2} \sim 5.66$. $\pm$ means the average and standard deviation across 5 runs of training.

| learning method | dataset | training tasks | | accuracy | | proper tests | |
|---|---|---|---|---|---|---|---|
| | | total | noise | test | train | [count] | [$\sigma$] |
| MAML | OM | 128 | 16 | 0.668±0.017 | 1.000±0.000 | 68.6±6.0 | 0.8±1.1 $\sigma$ |
| MAML | OM | 256 | 32 | 0.759±0.009 | 1.000±0.000 | 77.0±7.9 | 2.3±1.4 $\sigma$ |
| MAML | OM | 512 | 64 | 0.824±0.010 | 0.984±0.032 | 118.4±4.4 | 9.6±0.8 $\sigma$ |
| MAML | OM | 1024 | 128 | 0.882±0.032 | 0.999±0.003 | 114.8±7.2 | 9.0±1.3 $\sigma$ |
| MAML | OM | 2048 | 256 | 0.892±0.005 | 0.987±0.004 | 90.0±3.9 | 4.6±0.7 $\sigma$ |
| MAML | OM | 4096 | 512 | 0.933±0.006 | 0.940±0.005 | 82.2±5.9 | 3.2±1.0 $\sigma$ |
| MAML | OM | 8192 | 1024 | 0.960±0.001 | 0.906±0.003 | 94.0±6.0 | 5.3±1.1 $\sigma$ |

Table 9: Experiments with a CNN trained by MAML(Omniglot dataset, combined with noise image tasks). See Table 8 for notations.

parameters) with 40,000 meta-batches using MAML and 4-conv×32-filter feature extractors (28,896 parameters) with 40,000 meta-batches using Protonet.

Table 12 shows the results of counting the proper tests when the training task subpopulation is defined by how many superclasses it shares with the test task, illustrating that TLXML distinguishes the subpopulations that share semantic properties with test tasks from the other training tasks.

We also considered classifying the labels in a test task and defining the training subpopulation for each of the classified groups of labels. Table 13, and 14 show the results of counting the proper tests when the training subpopulation is defined with the test labels classified by their recall values achieved in the tests. Although the classification leads to a small number of statistics for each condition, the results show that TLXML distinguishes training tasks that share semantic properties with test tasks from the other tasks. Furthermore, we can observe the tendency that the labels of low recall values define subpopulations with comparatively small significance, which implies correlation between the recalls and influence scores(i.e., when a label has a good recall, training tasks with that label are scored high).

## B.6 One-Step Update Using TLXML

We examined the effect of the one-step update Eq. 13 from the convergence point of meta-learning. We utilized 8192 training tasks from FC60 and trained a 3-conv×16-filter network(6469 parameters) with 40,000 meta-bathces using MAML. We conducted 5 runs of training with seeds 0, 1, 2, 3, and 42. Each training task was scored with the influence function for the average test loss across 1024 test tasks. For each setting on the number of blocked and enhanced training tasks, those tasks were determined by selecting the worst and best training tasks from the scoring results.

Table 15 shows the effects of both blocking and enhancing training tasks, and the impact on the test accuracies in two cases: test datasets with and without shared superclasses (FC60 testset and FC100 testset, respectively). For each of the combinations of blocked/enhanced and the two testsets, we observe cases of improved test accuracy. The improvement is small in the region of a small absolute value of the shift parameter $\xi$ and a small number of blocked/enhanced tasks, because that is near the cases of doing nothing. The improvement is also small in the region of a large absolute value of the shift parameter $\xi$ and a large number of blocked/enhanced tasks, which is considered to be caused by the deterioration of the linear approximation by the influence function $I^{\mathrm{meta}}$.

The table also shows the results of leave-out retraining. For each set of blocked tasks, the network was trained from scratch with those tasks removed from the dataset. We observe that the leave-out trainings fail to yield a statistically significant gain.

| learning method | dataset | training tasks total | training tasks noise | accuracy test | accuracy train | proper tests [count] | proper tests [$\sigma$] |
|---|---|---|---|---|---|---|---|
| Protonet | MI | 128 | 16 | 0.397±0.016 | 0.997±0.001 | 63.0±15.5 | -0.2±2.7 $\sigma$ |
| Protonet | MI | 256 | 32 | 0.435±0.018 | 0.778±0.010 | 108.2±9.2 | 7.8±1.6 $\sigma$ |
| Protonet | MI | 512 | 64 | 0.496±0.009 | 0.608±0.037 | 94.4±8.4 | 5.4±1.5 $\sigma$ |
| Protonet | MI | 1024 | 128 | 0.503±0.015 | 0.570±0.007 | 81.8±7.5 | 3.1±1.3 $\sigma$ |
| Protonet | MI | 2048 | 256 | 0.498±0.007 | 0.550±0.008 | 84.2±7.6 | 3.6±1.3 $\sigma$ |
| Protonet | MI | 4096 | 512 | 0.497±0.016 | 0.537±0.007 | 96.4±9.9 | 5.7±1.7 $\sigma$ |
| Protonet | MI | 8192 | 1024 | 0.509±0.007 | 0.535±0.008 | 95.6±5.3 | 5.6±0.9 $\sigma$ |

Table 10: Experiments with a CNN trained by Prototypical Network(MiniImagenet dataset, combined with noise image tasks). See Table 8 for notations.

| learning method | dataset | training tasks total | training tasks noise | accuracy test | accuracy train | proper tests [count] | proper tests [$\sigma$] |
|---|---|---|---|---|---|---|---|
| Protonet | OM | 128 | 16 | 0.937±0.007 | 1.000±0.000 | 77.4±4.7 | 2.4±0.8 $\sigma$ |
| Protonet | OM | 256 | 32 | 0.958±0.003 | 1.000±0.000 | 61.0±3.4 | -0.5±0.6 $\sigma$ |
| Protonet | OM | 512 | 64 | 0.963±0.004 | 0.986±0.004 | 112.6±3.9 | 8.6±0.7 $\sigma$ |
| Protonet | OM | 1024 | 128 | 0.973±0.002 | 0.928±0.002 | 84.0±4.1 | 3.5±0.7 $\sigma$ |
| Protonet | OM | 2048 | 256 | 0.978±0.002 | 0.901±0.001 | 81.0±4.1 | 3.0±0.7 $\sigma$ |
| Protonet | OM | 4096 | 512 | 0.978±0.003 | 0.891±0.001 | 84.0±4.6 | 3.5±0.8 $\sigma$ |
| Protonet | OM | 8192 | 1024 | 0.979±0.001 | 0.884±0.008 | 81.2±5.5 | 3.0±1.0 $\sigma$ |

Table 11: Experiments with a CNN trained by Prototypical Network(Omniglot dataset, combined with noise image tasks). See Table 8 for notations.

### B.7 Computational Resources

The experiments in this paper are carried out in multiple computing environments. In a typical environment, the machine is equipped with an Intel Core i7-7567U CPU with a 3.50GHz clock, 32GB RAM, and an NVIDIA GeForce RTX 2070 GPU, and the operating system is Ubuntu 24.04 LTS.

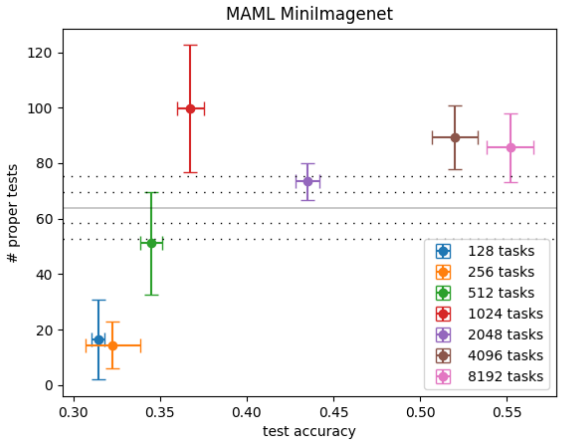



Figure 10: Effect of generalization with a CNN trained by MAML(MiniImagenet dataset, combined with noise image tasks). The solid horizontal line at y=64 represents the mean value under the hypothesis of random ordering. The dashed horizontal lines represent the mean value $\pm 1\sigma$ and $\pm 2\sigma$ under the hypothesis of random ordering.

| learning method | filter label overlap | train tasks total | train tasks filtered | test tasks total | test tasks proper | $\sigma$ |
|---|---|---|---|---|---|---|
| MAML | 1 | 8192 | 7522±314 | 1024 | 618 (+6.6$\sigma$) | 16 |
| MAML | 2 | 8192 | 4781±718 | 1024 | 676 (+10.3$\sigma$) | 16 |
| MAML | 3 | 8192 | 1508±422 | 1024 | 688 (+11.0$\sigma$) | 16 |
| MAML | 4 | 8192 | 171±69 | 1024 | 637 (+7.8$\sigma$) | 16 |
| Protonet | 1 | 8192 | 7522±314 | 1024 | 658 (+9.1$\sigma$) | 16 |
| Protonet | 2 | 8192 | 4781±718 | 1024 | 672 (+10.0$\sigma$) | 16 |
| Protonet | 3 | 8192 | 1508±422 | 1024 | 703 (+11.9$\sigma$) | 16 |
| Protonet | 4 | 8192 | 171±69 | 1024 | 644 (+8.3$\sigma$) | 16 |

Table 12: Experiments of distinguishing a training task subpopulation with superclasses shared with the test task from other training tasks. The 2nd column specifies the least number of superclasses shared with each test task, which defines the training tasks 'similar' to the test task. The 4th column shows the number of training tasks satisfying the condition. The 6th column shows the counts of proper tests and their statistical significance. ± means the average and standard deviations across 1024 test tasks.

| learning method | filter recall | label overlap | train tasks total | train tasks filtered | test tasks total | test tasks proper | $\sigma$ |
|---|---|---|---|---|---|---|---|
| MAML | 1.0 | 1 | 8192 | 2013±1145 | 828 | 440 (+1.8$\sigma$) | 14.4 |
| MAML | 1.0 | 2 | 8192 | 785±325 | 377 | 205 (+1.7$\sigma$) | 9.7 |
| MAML | 1.0 | 3 | 8192 | 245±53 | 97 | 57 (+1.7$\sigma$) | 4.9 |
| MAML | 0.8 | 1 | 8192 | 1833±1184 | 819 | 488 (+5.5$\sigma$) | 14.3 |
| MAML | 0.8 | 2 | 8192 | 728±358 | 427 | 255 (+4.0$\sigma$) | 10.3 |
| MAML | 0.8 | 3 | 8192 | 221±81 | 136 | 90 (+3.8$\sigma$) | 5.8 |
| MAML | 0.8 | 4 | 8192 | 43±17 | 20 | 17 (+3.1$\sigma$) | 2.2 |
| MAML | 0.6 | 1 | 8192 | 2187±1105 | 706 | 431 (+5.9$\sigma$) | 13.3 |
| MAML | 0.6 | 2 | 8192 | 793±319 | 267 | 170 (+4.5$\sigma$) | 8.2 |
| MAML | 0.6 | 3 | 8192 | 217±81 | 55 | 37 (+2.6$\sigma$) | 3.7 |
| MAML | 0.4 | 1 | 8192 | 2485±960 | 549 | 319 (+3.8$\sigma$) | 11.7 |
| MAML | 0.4 | 2 | 8192 | 266±848 | 138 | 87 (+3.1$\sigma$) | 5.9 |
| MAML | 0.4 | 3 | 8192 | 247±41 | 16 | 11 (+1.5$\sigma$) | 2.0 |
| MAML | 0.2 | 1 | 8192 | 2707±764 | 380 | 223 (+3.4$\sigma$) | 9.7 |
| MAML | 0.2 | 2 | 8192 | 906±226 | 57 | 35 (+1.7$\sigma$) | 3.8 |
| MAML | 0.0 | 1 | 8192 | 2888±87 | 220 | 133 (+3.1$\sigma$) | 7.4 |

Table 13: Experiments of distinguishing a training task subpopulation with superclasses shared with the test task (classified by recall values) from other training tasks(MAML). The 2nd column specifies the recall value of each label in the test tasks, and the 3rd column specifies the least number of labels with that recall value. The 6th column shows the number of test tasks the labels of which satisfy the conditions. The 5th column shows the number of training tasks 'similar' to the test task, meaning that the superlabels in each of them include those in the test task specified by the conditions. The 7th column shows the counts of proper tests and their statistical significance. ± means the average and standard deviations across the test tasks.

| learning method | filter recall | label overlap | train tasks total | train tasks filtered | test tasks total | test tasks proper | $\sigma$ |
|---|---|---|---|---|---|---|---|
| Protonet | 1.0 | 1 | 8192 | 2225±1095 | 681 | 396 (+4.3$\sigma$) | 13.0 |
| Protonet | 1.0 | 2 | 8192 | 804±326 | 248 | 151 (+3.4$\sigma$) | 7.9 |
| Protonet | 1.0 | 3 | 8192 | 218±82 | 56 | 37 (+2.4$\sigma$) | 3.7 |
| Protonet | 0.8 | 1 | 8192 | 1652±1197 | 838 | 479 (+4.1$\sigma$) | 14.5 |
| Protonet | 0.8 | 2 | 8192 | 679±378 | 492 | 290 (+4.0$\sigma$) | 11.1 |
| Protonet | 0.8 | 3 | 8192 | 207±90 | 184 | 109 (+2.5$\sigma$) | 6.8 |
| Protonet | 0.8 | 4 | 8192 | 45±15 | 36 | 25 (+2.3$\sigma$) | 3.0 |
| Protonet | 0.6 | 1 | 8192 | 1874±1198 | 790 | 478 (+5.9$\sigma$) | 14.1 |
| Protonet | 0.6 | 2 | 8192 | 697±373 | 392 | 233 (+3.7$\sigma$) | 9.9 |
| Protonet | 0.6 | 3 | 8192 | 194±92 | 127 | 68 (+0.8$\sigma$) | 5.6 |
| Protonet | 0.6 | 4 | 8192 | 33±18 | 20 | 12 (+0.9$\sigma$) | 2.2 |
| Protonet | 0.4 | 1 | 8192 | 2311±1075 | 589 | 341 (+3.8$\sigma$) | 12.1 |
| Protonet | 0.4 | 2 | 8192 | 763±344 | 187 | 118 (+3.6$\sigma$) | 6.8 |
| Protonet | 0.4 | 3 | 8192 | 162±88 | 37 | 21 (+0.8$\sigma$) | 3.0 |
| Protonet | 0.2 | 1 | 8192 | 2763±718 | 312 | 174 (+2.0$\sigma$) | 8.8 |
| Protonet | 0.2 | 2 | 8192 | 855±244 | 38 | 19 (+0.0$\sigma$) | 3.1 |
| Protonet | 0.0 | 1 | 8192 | 2999±243 | 74 | 44 (+1.6$\sigma$) | 4.3 |

Table 14: Experiments of distinguishing a training task subpopulation with superclasses shared with the test task (classified by recall values) from other training tasks(Protonet). See Table 13 for notations

| Op. | Dataset (train→test) | # tasks | $\xi = -8$ | $\xi = -4$ | $\xi = -2$ | $\xi = -1$ | leave-out |
|---|---|---|---|---|---|---|---|
| block | FC60→FC60 | 128 | 0.670±0.005 | 0.667±0.004 | 0.665±0.004 | 0.664±0.004 | 0.662±0.006† |
| | (MAML: $0.663 \pm 0.004$) | 256 | **0.672±0.005** | 0.670±0.004 | 0.666±0.004 | 0.665±0.004 | 0.660±0.007† |
| | | 512 | **0.675±0.006** | **0.672±0.005** | 0.669±0.004 | 0.666±0.004 | 0.663±0.008 |
| | | 1024 | 0.673±0.008 | **0.676±0.005** | **0.671±0.004** | 0.668±0.004 | 0.661±0.006† |
| | | 2048 | 0.659±0.009 | **0.674±0.007** | **0.672±0.004** | 0.669±0.004 | 0.666±0.010 |
| | FC60→FC100 | 128 | **0.439±0.007** | 0.436±0.005 | 0.433±0.005 | 0.431±0.005 | – |
| | (MAML: $0.429 \pm 0.005$) | 256 | **0.443±0.008** | **0.438±0.006** | 0.435±0.006 | 0.432±0.005 | – |
| | | 512 | **0.448±0.011** | **0.443±0.007** | **0.438±0.005** | 0.434±0.005 | – |
| | | 1024 | **0.452±0.011** | **0.448±0.009** | **0.441±0.007** | 0.437±0.006 | – |
| | | 2048 | 0.447±0.013 | **0.449±0.011** | **0.444±0.008** | 0.439±0.007 | – |
| **Op.** | **Dataset (train→test)** | **# tasks** | $\xi = 1$ | $\xi = 2$ | $\xi = 4$ | $\xi = 8$ | |
| enhance | FC60→FC60 | 128 | 0.664±0.004 | 0.666±0.005 | 0.668±0.005 | **0.672±0.005** | – |
| | (MAML: $0.663 \pm 0.004$) | 256 | 0.666±0.005 | 0.668±0.005 | **0.672±0.005** | **0.676±0.005** | – |
| | | 512 | 0.668±0.005 | **0.672±0.005** | **0.677±0.006** | **0.677±0.007** | – |
| | | 1024 | **0.670±0.005** | **0.676±0.005** | **0.680±0.006** | 0.662±0.010† | – |
| | | 2048 | **0.675±0.005** | **0.680±0.006** | **0.676±0.008** | **0.595±0.026**† | – |
| | FC60→FC100 | 128 | 0.432±0.005 | 0.433±0.004 | 0.436±0.005 | 0.437±0.007 | – |
| | (MAML: $0.429 \pm 0.005$) | 256 | 0.434±0.005 | 0.436±0.005 | 0.438±0.007 | 0.436±0.012 | – |
| | | 512 | 0.435±0.005 | 0.438±0.007 | 0.439±0.010 | 0.430±0.012 | – |
| | | 1024 | **0.438±0.005** | **0.441±0.008** | 0.437±0.011 | **0.413±0.008**† | – |
| | | 2048 | **0.440±0.007** | **0.442±0.009** | 0.429±0.010 | **0.390±0.020**† | – |

Table 15: Test accuracies after a single TLXML-guided update to a MAML model with blocked tasks. $\pm$ means the average and standard deviation across 5 runs of MAML training. Bold values outperform the MAML baseline (shown in the bracket) with a unpaired Welch two-sample $t$-test, $p < 0.05$. † means the average is below the baseline value.