A THESIS SUBMITTED FOR THE DEGREE OF Ph.D.

EXPLOITING GRADIENT INFORMATION FOR MODERN MACHINE LEARNING PROBLEMS

CHEN YIZHOU

(B.S. (Hons.), Nanyang Technological University)

A0174476U

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2022

Supervisor:

Associate Professor Bryan Kian Hsiang Low

Examiners:

Professor Ng See Kiong & Dr. Harold Soh Soon Hong

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

CHEN YIZHOU

29 April 2022

ACKNOWLEDGEMENT

I would like to say a special thank you to my supervisor, Bryan. His support, guidance and overall insights in this field have made my Ph.D. journey an inspiring experience for me. His insightful feedback pushed me to sharpen my thinking and brought my work to a higher level. I would also like to thank all of the people who participated in the collaboration with me, for their valuable support throughout our studies. Finally, I would like to thank my family, friend, and lover for supporting me during the compilation of this dissertation. Their wise counsel and sympathetic ear gave me the courage to complete this dissertation.

Contents

Summary

Many *deep learning* achievements are attributed to the *back-propagation* (BP) algorithm, which exploits gradient information of the *deep neural network* (DNN) models: BP efficiently computes the gradient of the loss function with respect to the weights of a DNN for a batch of examples, and such gradient can be used by stochastic gradient descent to perform learning / optimization of the DNN model. Despite recent advances in deep learning like DNN training, there are still important scenarios where we can also use gradient to tackle optimization difficulty. In a broader aspect of deep learning rather than DNN training, a significant challenge faced by ML practitioners is thus whether we can design efficient algorithms to use the model gradient in the training / optimization in various deep learning scenarios. This thesis identifies four important scenarios and, for each of them, proposes a novel algorithm to utilize the gradient information for effective optimization that is both theoretically grounded and practically effective.

Firstly, the training process of a machine learning (ML) model may be subject to adversarial attacks from an attacker who attempts to undermine the test performance of the ML model by perturbing the training minibatches, and thus needs to be protected by a defender. Such a problem setting is referred to as training-time adversarial ML. We formulate it as a two-player game and propose a principled Recursive Reasoning-based Training-Time adversarial ML (R2T2) framework to model this game. R2T2 models the reasoning process between the attacker and the defender and captures their bounded reasoning capabilities (due to bounded computational resources) through the recursive reasoning formalism. In particular, we associate a deeper level of recursive reasoning with

the use of a higher-order gradient to derive the attack (defense) strategy, which naturally improves its performance while requiring greater computational resources. R2T2 can empirically achieve state-of-the-art attack and defense performances on benchmark image datasets.

Secondly, probabilistic modelling with neural network architectures constitute a well-studied and popular area of deep learning. In contrast to a frequentist approach which is easy to overfit to the available dataset and risk learning unwanted biasing in the dataset, Gaussian process (GP) models were introduced as a fully probabilistic substitute and is one of the dominant approaches in Bayesian learning. A multi-layer *deep Gaussian process* (DGP) model is a hierarchical composition of GP models with a greater expressive power, and is more useful when dealing with complicated dataset. Exact DGP inference is intractable, and the approximation methods either yields a biased posterior belief (deterministic approximation by variational inference) or is computationally costly (stochastic approximation by Monte Carlo sampling). These difficulties have motivated our recent development of an *implicit posterior variational inference* (IPVI) framework for DGPs that can ideally recover an unbiased posterior belief and still preserve time efficiency. However, as a generator and a discriminator are integrated in each layer of the DGP, the training becomes unstable and is prone to optimization difficulties. To resolve such issues, we propose a novel gradient-bridging architecture of the generator and discriminator for the DGP model, which uses the inducing inputs as the context, thus leads to faster training and more accurate predictions. Empirical evaluation shows that IPVI with our proposed architecture outperforms the state-of-the-art methods for DGPs.

Thirdly, many widely adopted Bayesian meta-learning frameworks model the uncertainty in the predictions with a set of particles or a variational distribution (of the meta-parameters), which does not allow latent task modeling.[1] We present a novel *implicit process-based meta-learning* (IPML) algorithm that, in contrast to existing works,

[1]Latent task modeling requires mapping each task to a finite length latent vector. However, representing each task by a set of particles or a variational distribution of the meta-parameters is troublesome, since they have a too large dimension. In the real-world datasets that are usually considered, the dimension of the meta-parameters can be close to a size of 1 Million.

explicitly represents each task as a continuous latent vector and models its probabilistic belief within the highly expressive *implicit processes* (IP) framework. IP is a stochastic process with highly flexible implicit priors over functions, and is suitable as a Bayesian (meta) learning model for complicated datasets (e.g., when the priors are non-Gaussian unlike in GP). We tackle the meta-training in IPML with a novel expectation-maximization algorithm based on the stochastic gradient Hamiltonian Monte Carlo sampling method. Our delicate design of the neural network architecture for meta-training in IPML allows competitive meta-learning performance to be achieved. Unlike existing works, IPML offers the benefits of being amenable to the characterization of a principled distance measure between tasks using the maximum mean discrepancy, active task selection without needing the assumption of known task contexts, and synthetic task generation by modeling task-dependent input distributions. Empirical evaluation on benchmark datasets shows that IPML outperforms existing Bayesian meta-learning algorithms.

Last but not least, in the problem of active task selection which involves selecting the most informative tasks for meta-learning, we propose a novel active task selection criterion based on the mutual information between latent task vectors. Unfortunately, such a criterion scales poorly in the number of candidate tasks when optimized. To resolve this issue, we exploit the submodularity property of our new criterion for devising the first active task selection algorithm for meta-learning with a near-optimal performance guarantee. To further improve our efficiency, we propose an online variant of the Stein variational gradient descent to perform fast belief updates of the meta-parameters via maintaining a set of forward (and backward) particles when learning (or unlearning) from each selected task. We empirically demonstrate the superior performance of our proposed algorithm on real-world datasets.

List of Publications

Below is a list of submitted papers and publications whose contributions are included in this thesis:

- **Chen, Y.**, Dai, Z., Yu, H., Low, B. K. H., & Ho, T. H. (2021). Recursive reasoning-based training-time adversarial machine learning. Plan to submit to *Artificial Intelligence (Special Issue on Risk-aware Autonomous Systems: Theory and Practice)*.

- **Chen, Y.***, Yu, H.*, Dai, Z., Low, B. K. H., & Patrick, J. (2019). Implicit Posterior Variational Inference for Deep Gaussian Processes. Accepted by *NeurIPS 2019 (Spotlight Presentation)*.

- **Chen, Y.**, Low, B. K. H., et al. (2021). Meta-learning with implicit processes. Plan to extend for journal submission.

- **Chen, Y.**, Zhang, S., Low, B. K. H. (2021). Near-Optimal Task Selection with Mutual Information for Meta-Learning. Submitted to *AISTATS*.

Listed below are some other publications:

- Dai, Z., **Chen, Y.**, Low, B. K. H., Jaillet, P., & Ho, T. H. (2020). R2-B2: Recursive reasoning-based Bayesian optimization for no-regret learning in games. In *Proc. ICML*.

* Equal contribution.

Chapter 1

Introduction

1.1 Background and Motivation

In recent years, *machine learning* (ML), especially *deep learning* (DL), has demonstrated an unprecedented level of performance in a number of applications such as image recognition, natural language processing, complex board games, among other [LeCun et al., 2015, Silver et al., 2016]. Many of these achievements are attributed to the *back-propagation* (BP) algorithm, which exploits gradient information of the *deep neural network* (DNN) models. More specifically, BP efficiently computes the gradients of the loss function with respect to the weights of a DNN for a batch of examples, and such gradients can be used by stochastic gradient descent to perform learning/optimization of the DNN model.

DNN training is an example showing that a key component in DL is the usage of gradients. Despite significant progresses in DL, there are still important scenarios where we can also use gradients to achieve effective optimization. In a broader aspect of DL rather than DNN training, a significant challenge faced by ML practitioners is thus whether we can design efficient algorithms to use the model gradients in the training/optimization in various DL scenarios.

Firstly, the widespread use of ML models has inevitably raised concerns about the security and safety of their applications in the real world, which has galvanized the recent

popularity of adversarial ML. In the case when a (surrogate) model is available to an attacker, the attacker can compute the gradient of the model with respect to the input seeking to maximally compromise the predictive performance of the model (by perturbing the input using such gradient). As a countermeasure, a defender can be invoked to correct the received input before using them. Naturally, an interesting question arises: How should the attacker (defender) choose its strategy of using gradients to achieve more effective attacks (defenses) and thus gain an advantage in adversarial ML?

Secondly, we want to perform learning (inference) of *deep Gaussian process* (DGP) using our *implicit posterior variational inference* (IPVI) model [Yu et al., 2019], where each layer of the DGP is equipped with a generator and a discriminator. The generator produces the samples of the inducing output variables, and propagate its prediction layer by layer based on the learnable inducing inputs. However, a difficulty is that a naive design of the generator fails to adequately capture the dependency of the inducing output variables on the corresponding inducing inputs, such that the inducing inputs will not receive useful gradients during training.

Thirdly, a number of meta-learning algorithms [Finn et al., 2018, Jerfel et al., 2019, Ravi and Beatson, 2018, Rusu et al., 2019, Yoon et al., 2018] have recently adopted a probabilistic perspective to characterize the uncertainty in the predictions via a Bayesian treatment of the meta-parameters. Though they can consequently represent different (training or testing) tasks with different values of meta-parameters, doing so introduces difficulties when the downstream applications require finite-length vector representations of the tasks, since the dimension of the meta-parameters are usually too large. To resolve this, we turn to perform meta-learning of a set of tasks using the *implicit process* (IP) model, which explicitly represents each task as a continuous latent vector. Meta-learning consists of two loops of calculation: the *outer* loop performs model update by optimizing the parameters of the IP model through gradient descent of the meta-learning objective, which relies on the posterior belief of the latent vector; and the *inner* loop performs posterior belief inference (usually referred to as task adaptation) of the latent vector (of

a specific task). However, a difficulty in meta-learning using IP is that exact inner loop inference is intractable, and variational inference using a Gaussian posterior does not yield satisfactory performance. Nevertheless, in the case that the IP model is differentiable, we can compute the gradient of the task data likelihood with respect to the latent vector. Using such gradient information, we manage to solve the difficulty of inner loop inference and obtain posterior samples of the latent vector.

Lastly, we want to efficiently evaluate our criterion called *mutual information between latent task vectors* (MILT) to perform active task selection for meta-learning while having a near-optimal guarantee. However, frequent belief updates of the meta-parameters are required during the evaluation, which largely affects the computational efficiency of the proposed criterion.

In the scenarios like above, we ask the question: *Can we design practical algorithms to utilize the gradient information for effective optimization in deep learning?* We answer this question affirmatively in this thesis by introducing novel algorithms for the two aforementioned scenarios. Our proposed algorithm for the first scenario exploits a nested projected gradient descent-based method to approximate the optimal attack (defense) strategy. For the second scenario, we designed a novel gradient-bridging architecture of the generator to largely alleviate the optimization difficulties in our IPVI-DGP. For the third scenario, we propose an expectation-maximization algorithm based on the *stochastic gradient Hamiltonian Monte Carlo sampling* (SGHMC) method to perform meta-training. In the mean time, a novel IP architecture is designed to enable effective gradient exploitation. For the last scenario, we proposed a forward-backward method based on online *Stein variational gradient descent* (SVGD) to perform fast belief updates of the meta-parameters, thus improving the efficiency in evaluating our MILT criterion. In the next section, we summarize our contributions in this thesis.

1.2 Contributions

Here we give a short summary of our contributions in this thesis, including two of our recent works which involve designing practical algorithms to utilize the gradient information for effective optimization in deep learning scenarios.

Gradient-based strategical recursive reasoning in training-time adversarial machine learning. The training process of a machine learning (ML) model may be subjected to adversarial attacks from an attacker who attempts to undermine the test performance of the ML model by perturbing the training minibatches, and thus needs to be protected by a defender. Such a problem setting is referred to as training-time adversarial ML. We formulate it as a two-player game and propose a principled Recursive Reasoning-based Training-Time adversarial ML (R2T2) framework to model this game. R2T2 models the reasoning process between the attacker and the defender and captures their bounded reasoning capabilities (due to bounded computational resources) through the recursive reasoning formalism. In particular, we associate a deeper level of recursive reasoning with the use of a higher-order gradient to derive the attack (defense) strategy, which naturally improves its performance while requiring greater computational resources. Interestingly, our R2T2 framework encompasses a variety of existing adversarial ML methods which correspond to attackers (defenders) with different recursive reasoning capabilities. We show how an R2T2 attacker (defender) can utilize our proposed nested projected gradient descent-based method to approximate the optimal attack (defense) strategy at an arbitrary level of reasoning. R2T2 can empirically achieve state-of-the-art attack and defense performances on benchmark image datasets.

Implicit posterior variational inference with gradient-bridging design for deep Gaussian processes. In our proposed *implicit posterior variational inference* (IPVI) framework for DGPs which can ideally recover an unbiased posterior belief, the posterior is modelled with a neural network-based generator. However, during the training of DGP, the model parameters (including the generator) are optimized based on first-

order optimization, and performs badly without careful design of the architecture. We identified two possible problems: (i) Overfitting due to overly-parameterized generator; (ii) Optimization difficulties due to the inefficient gradient back-propagation. Ablation studies are performed and eventually we identified that the optimization difficulties due to the inefficient gradient back-propagation is the main obstacle in the realization of our algorithm. To this end, we proposed a novel gradient-bridging architecture to solve such optimization difficulties. We also empirically evaluate the performance of our novel IPVI architecture on several real-world datasets.

Stochastic gradient Hamiltonian Monte Carlo for meta-learning with implicit processes. We present a novel *implicit process-based meta-learning* (IPML) algorithm that, in contrast to existing works, explicitly represents each task as a continuous latent vector and models its probabilistic belief within the highly expressive *implicit processes* (IP) framework. Unfortunately, meta-training in IPML is computationally challenging due to its need to perform intractable exact IP inference in task adaptation. To resolve this, we propose a novel expectation-maximization algorithm based on the stochastic gradient Hamiltonian Monte Carlo sampling method to perform meta-training. Our delicate design of the neural network architecture for meta-training in IPML allows competitive meta-learning performance to be achieved. Unlike existing works, IPML offers the benefits of being amenable to the characterization of a principled distance measure between tasks using the maximum mean discrepancy, active task selection without needing the assumption of known task contexts, and synthetic task generation by modeling task-dependent input distributions. Empirical evaluation on benchmark datasets shows that IPML outperforms existing Bayesian meta-learning algorithms. We have also empirically demonstrated on an e-commerce company's real-world dataset that IPML outperforms the baselines and identifies "outlier" tasks which can potentially degrade meta-testing performance.

Online Stein variational gradient descent for near-optimal task selection in meta-learning. We propose the first active task selection algorithm for meta-learning with a near-optimal performance guarantee. To evaluate our proposed active task selection

criterion, we design a forward-backward method based on our online variant of SVGD to largely improve the efficiency of our algorithm. Finally, empirical evaluation on several benchmark datasets demonstrate the state-of-the-art performance of our algorithm.

1.3 Organization

In the remainder of this thesis, we firstly introduce the related works in Chapter 6. Next, the following four chapters present each of the four works in detail: Recursive reasoning-based training-time adversarial machine learning (Chapter 2), Implicit posterior variational inference for deep Gaussian processes (Chapter 3), Meta-learning with implicit processes (Chapter 4), and Near-optimal task selection with mutual information for meta-learning (Chapter 5). Lastly, we conclude the thesis in Chapter 7.

Chapter 2

Gradient-based Strategical Recursive Reasoning in Training-time Adversarial Machine Learning

2.1 Background

The widespread use of *machine learning* (ML) models has inevitably raised concerns about the security and safety of their application in the real world, which has galvanized the recent popularity of *adversarial ML*. In particular, the training process of an ML model may be subject to adversarial attacks whose objective is to fool the ML model into performing well during training yet inadequately during deployment. Therefore, a defense mechanism is needed to ensure the reliability of the ML model.

An important motivating scenario is an online learning system in which the model learning performed by a central server requires training data that are sent in minibatches from some data source(s). This is an increasingly prevalent setting in modern ML where large amounts of data are available [Jones, 2014, McMahan et al., 2013]. Due to memory constraints, the data is usually gathered and stored locally, and then transmitted to the central server for model update. This system is susceptible to attacks from (a)

an adversarial data provider who sends out poisonous minibatch data or (b) a malicious hacker who can access and modify the transmitted data. These attackers usually attempt to compromise the test-time performance of the trained ML model on clean data samples. As a countermeasure, the central server can invoke a defender to correct the received data before using them for model update.

Formally, we refer to this problem setting as *training-time adversarial ML* which can be formulated as a *two-player game* between the *training-time attacker* and *training-time defender*. In such a game, it is usually too complicated to solve for a Nash equilibrium and hence not realistic for both players to employ Nash equilibrium strategies.[1] Nevertheless, an interesting research question remains to be answered: *How should the attacker (defender) choose its strategy to achieve more effective attacks (defenses) and thus gain an advantage in the game?*

Unfortunately, most works on adversarial ML have (a) focused on test-time attacks/defenses and (b) studied either attacks or defenses solely. Test-time attacks [Goodfellow et al., 2015] and defenses [Szegedy et al., 2014] are well-studied. However, as we will show in our experiments, they do not transfer well to training-time attacks and defenses. The work of [Feng et al., 2019] has proposed learning an auto-encoder-like network to generate training-time attacks without considering defenses. To the best of our knowledge, a detailed study of training-time defenses is still lacking.

In training-time adversarial ML, the training of the ML model (referred to as the *target ML model* hereafter) is affected by the strategies of both the attacker and defender. So, for the attacker to gain an advantage in the game, it has to reason about the defender's strategy. Similarly, to derive a more effective defense strategy, the defender can adopt a reasoning process that accounts for the attacker's potential strategy and belief about the defender. This gives rise to the *recursive reasoning* process which is captured by the *level K* (LK) model from behavioral game theory for explaining the behavior of humans [Ho et al., 1998, Nagel, 1995, Stahl and Wilson, 1994, Stahl and Wilson, 1995].

[1]The work of [Pal and Vidal, 2020] studies Nash Equilibrium strategies under the assumption of a binary classifier and locally linear decision boundaries.

In the LK model, every player (person) reasons at a particular level: A level-0 player selects a null strategy while a level-$k \geq 1$ player best-responds to the level-$(k-1)$ behavior of the opponent.

A person with a more sophisticated strategic thinking is capable of a deeper level of reasoning and usually performs better in a game. Similarly, in the game of training-time adversarial ML, a computational agent (attacker/defender) with access to more computational resources is expected to be able to find a better attack/defense strategy. In behavioral game theory, a person's reasoning level is determined by her cognitive capability, otherwise known as *cognitive limit* [Camerer et al., 2004, Gill and Prowse, 2016, Jin, 2018]. This naturally brings up a question: How should we characterize the cognitive limit of a computational agent with bounded computational resources and thus determine its reasoning level? Particularly in adversarial ML, the *gradient* has been constantly used to craft the attack and perform the defense. Thus in this work, we associate the cognitive limit and hence reasoning level of an attacker/defender with the *highest order of gradient* that can be computed by the player. This association turns out to be natural and elegant since having access to more gradient information is likely to improve its performance and the cost of computing a gradient grows exponentially in its order.

This chapter presents the first principled *Recursive Reasoning-based Training-Time adversarial ML* (R2T2) framework (Chapter 2.3) to model training-time adversarial ML as a 2-player game. Interestingly, our R2T2 framework encompasses a variety of existing adversarial ML methods which correspond to attackers (defenders) with different recursive reasoning capabilities. We show how an R2T2 attacker (defender) can utilize our proposed nested projected gradient descent-based method to approximate the optimal attack (defense) strategy at an arbitrary level of reasoning (Chapter 2.3.4). We empirically demonstrate that R2T2 can achieve state-of-the-art attack and defense performances on benchmark image datasets (Chapter 2.5) and, for the first time, provide principled guidelines to protect the training of an ML model against training-time attacks.

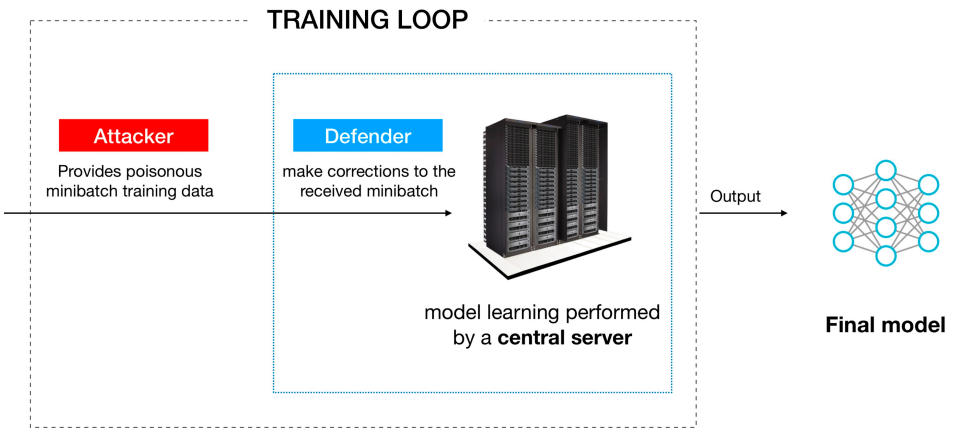

Figure 2.1: The illustration of the interaction between the attacker and the defender.

2.2 Problem Formulation

Utility Functions of the Players. Following the work of [Feng et al., 2019], we assume that the goal of the training-time attacker is to misguide the model into performing poorly on the clean data samples during test time, given that the model training converges. Meanwhile, the training-time defender tries to guide the model into converging properly in the presence of the attacker to ensure a good test-time performance. We focus on *non-targeted* attacks, that is, the attacker's objective is to cause an incorrect prediction. This is in contrast to *targeted* attacks in which the attacker intends to misguide the model into producing a specific output label. Nevertheless, the extension to targeted attacks is provided in Chapter 2.5.4 with experimental results.

Attack Strategy. Suppose that the target ML model is to be trained for a total of T iterations. In each training iteration $t = 0, 1, \ldots, T-1$, the attacker can intercept the minibatch data $\mathcal{D}_t$ transmitted to the ML model.[2] Let $\mathcal{D}_t^X$ denote the *inputs* of the minibatch $\mathcal{D}_t$. An *attack strategy* δ_t is a function mapping each input $x_t \in \mathcal{D}_t^X$ to a perturbation $\delta_t(x_t)$ that is added back to x_t to yield the perturbed input:

$$x_t' \triangleq x_t + \epsilon_{\text{atk}}\, \delta_t(x_t) \tag{2.1}$$

[2]It is thus implicitly assumed that both players do not have access to the entire training dataset.

such that $\|\delta_t(x_t)\| \leq 1$ which can be satisfied using a function, denoted by $\vec{\mathbf{e}}(\cdot)$, that outputs the unit vector $\delta_t(x_t)$ of its unnormalized perturbation.[3]

Defense Strategy. In training iteration t, the defender receives the corrupted minibatch data containing perturbed inputs x'_t's from the attacker and cannot access its original inputs x_t's.[2] A *defense strategy* σ_t is a function mapping each perturbed input x'_t to a transformation $\sigma_t(x'_t)$ that is added back to x'_t to yield the transformed input:

$$x''_t \triangleq x'_t + \epsilon_{\text{def}}\, \sigma_t(x'_t) \tag{2.2}$$

s.t. $\|\sigma_t(x'_t)\| \leq 1$.[3] Finally, x''_t is used by the target ML model on the central server to perform the model update for the current training iteration t. In (2.1) and (2.2), ϵ_{atk} and ϵ_{def} are pre-defined, fixed scale constants determined by external factors (e.g., transmission bandwidth). For simplicity, we assume $\epsilon \triangleq \epsilon_{\text{atk}} = \epsilon_{\text{def}}$ in our theoretical analysis.

We assume both players know that the target ML model is trained using the widely used *stochastic gradient descent* (SGD)[4] method which is an iterative 1st-order optimization scheme. In training iteration t, the model parameters θ_t are updated via SGD using the minibatch $\mathcal{D}_t$ with step size η:

$$\theta_{t+1} = \theta_t - \eta\, \nabla_\theta \sum_{x_t \in \mathcal{D}_t^X} \mathcal{L}_{\theta_t}(x''_t) \tag{2.3}$$

where the loss function $\mathcal{L}_\theta(x)$ denotes the loss incurred on input x by using the ML model with parameters θ to predict the ground-truth label $y(x)$ and is assumed to be smooth (i.e., infinitely differentiable) with respect to θ and x. We focus on **white-box attacks** and **white-box defenses**, that is, in each training iteration t, both players know the current parameters θ_t of the target ML model. Nevertheless, the attacker can realize **black-box attacks** by training a local surrogate model which follows the common practice

[3]Let $\|\cdot\|$ denote the L_2 norm which is chosen to simplify the theoretical analysis. Some previous works [Goodfellow et al., 2015, Szegedy et al., 2014] use the L_∞ norm instead: $\|\delta_t(x_t)\|_\infty \leq 1$ can be easily adapted from $\|\delta_t(x_t)\| \leq 1$ by replacing $\vec{\mathbf{e}}(\cdot)$ with the sign function.

[4]In practice, using a momentum-based optimizer can better ensure training convergence (Appendix A.1.1.1).

in black-box attacks [Papernot et al., 2016]. On the other hand, since the defender is with the central server, we assume that it always performs white-box defenses. We provide experimental results using black-box attacks (Chapter 2.5) and include more details in Chapter 2.5.5.

The game between the non-targeted attacker and the defender can be formulated as a zero-sum *extensive-form game* which consists of a number of repetitions of some *base games*, as illustrated in Fig. 2.1. Every base game corresponds to one iteration of the model training, in which both players modify the minibatch data once. Given that the test set is not known, the attacker tries to maximize the following utility (empirical risk) which the defender attempts to minimize and is a sum of base game utilities over T iterations:

$$\sum_{t=0}^{T-1} \mathcal{U}_t \quad \text{s.t.} \quad \mathcal{U}_t \triangleq \sum_{z \in \mathcal{D}_{\text{val}}^X} \left[\mathcal{L}_{\theta_{t+1}}(z) - \mathcal{L}_{\theta_t}(z) \right] \tag{2.4}$$

where $\mathcal{D}_{\text{val}}^X$ denotes the *inputs* of the validation set $\mathcal{D}_{\text{val}}$. To understand (2.4), $\mathcal{U}_t$ is the *increment* in the validation loss in iteration t while its sum over T iterations is equivalent to the final validation loss after training is completed. Unfortunately, directly solving this extensive-form game in (2.4) is infeasible mainly due to the following two reasons: (a) The strategy spaces are infinite. (b) In iteration t, both players may not have access to the future minibatches in iterations $t+1, \ldots, T-1$, which makes it difficult to reason about the future utilities $\mathcal{U}_{t'}$ for $t' = t+1, \ldots, T-1$. These issues hinder the use of existing approaches (such as counterfactual regret minimization and backward induction) to solve this extensive-form game. Therefore, we adopt a more practical solution by disentangling the game across different iterations. That is, we assume that in every iteration t, both players greedily optimize the base game utilities $\mathcal{U}_t$.

To simplify notations, we assume each minibatch to be of size one, i.e., $\mathcal{D}_t^X = \{x_t\}$. Nevertheless, our analysis can be trivially generalized to each minibatch being of size more than one, as shown in Appendix A.2.4. Note that in iteration t, θ_t is known and $\mathcal{U}_t$ depends on x_t'' through θ_{t+1} following the model update (2.3). Therefore, we denote it as a

function of x_t'': $\mathcal{U}_t(x_t'')$. The greedy approach solves the following base game in iteration t, which is also zero-sum:

$$\begin{aligned} \textbf{Attacker:} \quad & \max_{\delta_t(x_t):\|\delta_t(x_t)\|\leq 1} \mathcal{U}_t(x_t'') \ , \\ \textbf{Defender:} \quad & \max_{\sigma_t(x_t'):\|\sigma_t(x_t')\|\leq 1} -\mathcal{U}_t(x_t'') \ . \end{aligned} \tag{2.5}$$

To avoid confusion, we emphasize that the training set is where the minibatch is sampled from and perturbed, the validation set is clean and public, and the test set is clean and unknown to both players. In our experiments, we evaluate their performance using the test accuracy. To allow both players to evaluate their utilities, we assume that a public validation set $\mathcal{D}_{\text{val}}$ is available to both players. However, in practice, the attacker can evaluate its utility even if $\mathcal{D}_{\text{val}}$ is unknown, i.e., simply by replacing $\mathcal{D}_{\text{val}}$ with the known $\mathcal{D}_t$ which is sampled from the same data distribution [Feng et al., 2019]. Such details and experimental results are in Appendix A.1.2.3. In contrast, the known validation set $\mathcal{D}_{\text{val}}$ is required by the defender since it does not have access to the clean minibatches. This is in the same spirit as the works on adversarial defense with data sanitization [Li and Ji, 2019, Meng and Chen, 2017, Samangouei et al., 2018] which assume that the defender has clean data for training the de-noising models. These defense methods reform the perturbed inputs towards the distribution of clean inputs using an auto-encoder [Meng and Chen, 2017], a *variational auto-encoder* (VAE) [Li and Ji, 2019], or a *generative adversarial network* (GAN) [Samangouei et al., 2018]. In contrast, instead of naively reforming the perturbed inputs using a de-noising model, our defender aims at directly reducing the validation loss in a principled way. Empirical evaluations show that our defender can potentially outperform perfect reconstruction (Chapter 2.5.3).

2.2.1 Connections with Adversarial Training

In adversarial training, the attacker is playing against a level-0/null-strategy defender under our R2T2 framework. In general, most adversarial training methods [Goodfellow et al., 2015,

Kurakin et al., 2017, Li et al., 2018, Madry et al., 2018, Miyato et al., 2019, Singla and Feizi, 2020, Tramèr et al., 2018] share the following utility function of the attacker to be maximized:

$$\textbf{Attacker:} \quad \max_{\delta_t(x_t):\|\delta_t(x_t)\|_\infty \leq 1} \left[\mathcal{L}_{\theta_t}(x'_t) - \mathcal{L}_{\theta_t}(x_t)\right]$$

whose utility function is different from $\mathcal{U}_t(x''_t)$ (2.5) in our framework. A key difference with adversarial training is that their utility function neglects the effect of training (i.e., no θ_{t+1} involved). This approximation is justified in their setting since test-time attacks, which they are defending against, occur only on the trained "static model". Therefore, in every base game of adversarial training, the attacker simulates such a test-time attack to the particular model instance in that base game.

2.3 Recursive Reasoning-based Training-Time Adversarial ML (R2T2)

This section introduces the R2T2 framework which makes use of the *level K* (LK) model to solve the game shown in (2.5) through recursive reasoning. In this framework, a level-0 player plays a null strategy and a level-$k \geq 1$ player chooses its strategy by best-responding to the level-$(k-1)$ opponent. We associate the cognitive limit and hence the reasoning level of the attacker or defender with the highest order of gradient that can be computed by the player:

Assumption 2.1 (**Cognitive limit**). *A level-k player can calculate and utilize at most the k-th-order gradients of the target ML model when deriving its strategy (i.e., the player can only calculate mixed partial $\nabla_\theta^a \nabla_x^b \mathcal{L}_\theta(x)$ where $a, b \leq k$), and assumes all higher-than-k-th-order gradients ($a > k$ or $b > k$) to be* **0**.

Under Assumption 2.1, we can formally define the LK model for the R2T2 framework and illustrate it in Fig. 2.2:

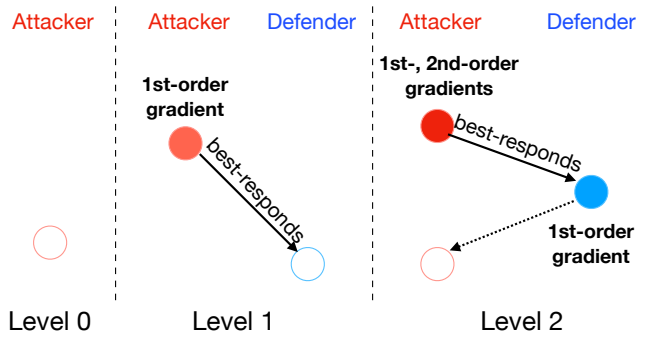

Figure 2.2: Illustration of how a level-$k = 0, 1, 2$ attack strategy is computed under the LK model for the R2T2 framework.

Definition 2.1 (LK). *A level-$k \geq 1$ player best-responds to the level-$(k-1)$ opponent by solving* (2.5) *using the k'-th-order gradients of the target ML model for all $k' \leq k$.*

This notion of cognitive limit preserves the inherent relationship in the LK model between the required computational resources and performance: As k increases, the performance of the level-k strategy is likely to improve. However, this requires greater computational resources to derive the level-k strategy since the cost of computing the k-th-order gradient grows exponentially in k. As an analogy, second-order optimization techniques usually have a better convergence guarantee than first-order ones but require more computations. Such a relationship will be explored in our experiments in Chapter 2.5.3. Note that although a level-k player best-responds to the level-$(k-1)$ opponent's strategy, the player can perform effectively (a) even if the opponent does not reason at level $k-1$ but a lower level or (b) when the opponent does not follow our recursive reasoning framework and instead adopts some existing attack/defense baselines, as demonstrated in our experiments (Chapter 2.5). In the subsections to follow, let the level-k attack (2.1) and defense (2.2) strategies be denoted by $\delta_t^k(x_t)$ and $\sigma_t^k(x'_t)$, respectively. Similarly, let their optimal strategies be denoted by $\delta_t^{k^*}(x_t)$ and $\sigma_t^{k^*}(x'_t)$, respectively.

2.3.1 R2T2 Level-0 (Null) Strategies

A level-0 player asserts that the model parameters are not updated (i.e., $\theta_{t+1} = \theta_t$) since all gradients are $\mathbf{0}$ due to Assumption 2.1 (i.e., $\nabla_\theta \mathcal{L}_{\theta_t}(x''_t) = \mathbf{0}$). So, both players think

their utilities are *constants*. Without loss of generality, we assume that both players play the null strategies: $\delta_t^{0^*}(x_t) \triangleq \mathbf{0}$ and $\sigma_t^{0^*}(x_t') \triangleq \mathbf{0}$ which correspond to no attack/defense. In fact, the level-0 strategies can be arbitrary (e.g., random noise) and do not affect the functional form of the higher level-$k \geq 1$ strategies under our R2T2 framework, as proven in Appendix A.2.1.

2.3.2 R2T2 Level-1 Strategies

Theorem 2.1 (**R2T2 Level-1 strategies**). *The optimal level-1 attack and defense strategies are*[5]

$$\delta_t^{1^*}(x_t) = \vec{\mathbf{e}}\left(\nabla_x \mathcal{U}_t(x_t)\right), \;\; \sigma_t^{1^*}(x_t') = -\vec{\mathbf{e}}\left(\nabla_x \mathcal{U}_t(x_t')\right) \tag{2.6}$$

where $\vec{\mathbf{e}}(\cdot)$ outputs the unit vector of its input and

$$\begin{aligned} \nabla_x \mathcal{U}_t(x) &= -\eta\, \mathbf{B}_{\theta_t}^x \big[\textstyle\sum_{z \in \mathcal{D}_{val}^X} \nabla_\theta \mathcal{L}_{\theta_t}(z)\big], \\ \mathbf{B}_{\theta_t}^x &\triangleq [\nabla_x \nabla_\theta \mathcal{L}_{\theta_t}(x)]^\top. \end{aligned} \tag{2.7}$$

Its proof is in Appendix A.2.1. Theorem 2.1 suggests that the optimal level-1 strategies follow from linearizing the utility functions, as implied by the linearized loss function under Assumption 2.1. The *deep confuse* (DC) strategy [Feng et al., 2019] can be interpreted as a variant of our level-1 attack strategy. Specifically, it trains an auto-encoder to learn an averaged $\delta_t^{1^*}$ over $t = 0, \ldots, T-1$ (or, equivalently, over the trajectory of model training), which makes their attack strategy independent of t. DC also differs from R2T2 by assuming access to the entire training set and not a known validation set $\mathcal{D}_{\text{val}}$. An advantage of DC is that their strategy can be computed using the auto-encoder without explicit gradient calculation. Its downside is that every trial (500 trials in total) of the auto-encoder training requires a full trajectory of model training which includes T iterations of gradient calculation and is hence time-consuming.

[5]Unless otherwise stated, the gradient of a scalar and the vectors are all column vectors.

Some well-known adversarial training methods can be interpreted as level-1 attack strategies: (a) *fast gradient sign method* (FGSM) perturbing inputs with first-order gradient-based attacks [Goodfellow et al., 2015], and (b) adversarial transformation network learning $\delta_t^{1^*}$ with first-order information [Baluja and Fischer, 2018].

2.3.3 R2T2 Level-2 Strategies: Special Case of $\epsilon \to 0$

Level-2 players are capable of computing the Hessian $H_{\theta_t} \triangleq \sum_{z \in \mathcal{D}_{\text{val}}^X} \nabla_\theta^2 \mathcal{L}_{\theta_t}(z)$ with respect to θ.

Theorem 2.2 (R2T2 Level-2 strategies). *The optimal level-2 attack and defense strategies as ϵ approaches 0 are*

$$\lim_{\epsilon \to 0} \delta_t^{2^*}(x_t) = -\vec{\mathbf{e}}\big(\mathbf{B}_{\theta_t}^{x_t}\big[\sum_{z \in \mathcal{D}_{val}^X} \nabla_\theta \mathcal{L}_{\theta_t}(z) - \eta H_{\theta_t} \nabla_\theta \mathcal{L}_{\theta_t}(x_t)\big]\big),$$

$$\lim_{\epsilon \to 0} \sigma_t^{2^*}(x_t') = \vec{\mathbf{e}}\big(\mathbf{B}_{\theta_t}^{x_t'}\big[\sum_{z \in \mathcal{D}_{val}^X} \nabla_\theta \mathcal{L}_{\theta_t}(z) - \eta H_{\theta_t} \nabla_\theta \mathcal{L}_{\theta_t}(x_t')\big]\big).$$

Its proof (Appendix A.2.2) utilizes the idea that since the search space has a bounded L_2 norm and a level-2 player assumes the k-th-order gradients to be $\mathbf{0}$ for all $k \geq 3$ (Assumption 2.1), finding the optimal level-2 strategy of a player can be framed as a constrained quadratic programming problem and is tractable as ϵ approaches 0.

2.3.3.1 Optimal Contraction (OC) & Connections with Influence Function Attacks (IF)

A further insight on our level-2 attacker can be drawn when the Hessian H_{θ_t} is positive definite. In this case, *optimal attacker's parameters* $\theta^\star \triangleq \arg\max_\theta \sum_{z \in \mathcal{D}_{\text{val}}^X} \mathcal{L}_\theta(z)$ exist such that the attacker achieves maximum validation loss since now $\sum_{z \in \mathcal{D}_{\text{val}}^X} \mathcal{L}_\theta(z)$ is a convex polynomial of θ (due to cognitive limit, a level-2 player visualizes the loss function as a second-order polynomial). So, instead of optimizing the base game utility (2.5), the attacker can choose to directly minimize $\|\theta_{t+1} - \theta^\star\|$ in each iteration t to make

the parameters of the target ML model move close to $\theta^\star$. In the ideal case where $\theta^\star$ is finally reached (i.e., $\|\theta_T - \theta^\star\| = 0$), the attacker achieves maximum extensive-form game utility (2.4). In this sense, the attacker is said to perform the *optimal contraction* (OC) strategy:

Corollary 2.1 (**Optimal contraction attack**). *Suppose that H_{θ_t} is positive definite and attacker is at level* 2. *The OC strategy $\delta_t^{\text{OC}}(x_t) \triangleq \arg\min_{\delta_t(x_t):\|\delta_t(x_t)\|\leq 1} \|\theta_{t+1} - \theta^\star\|$ as ϵ approaches* 0 *is*

$$\lim_{\epsilon\to 0} \delta_t^{\text{OC}}(x_t) = -\vec{\mathbf{e}}\big(\mathbf{B}_{\theta_t}^{x_t}\big[H_{\theta_t}^{-1} \sum_{z\in\mathcal{D}_{val}^X} \nabla_\theta \mathcal{L}_{\theta_t}(z) - \eta \nabla_\theta \mathcal{L}_{\theta_t}(x_t)\big]\big).$$

Its proof is in Appendix A.2.2. Interestingly, the IF strategy [Koh and Liang, 2017] also assumes a positive definite Hessian and adopts a similar attack strategy as OC:

$$\delta_t^{\text{IF}}(x_t) \triangleq -\vec{\mathbf{e}}\big(\mathbf{B}_{\theta_t}^{x_t}\big[H_{\theta_t}^{-1} \textstyle\sum_{z\in\mathcal{D}_{\text{val}}^X} \nabla_\theta \mathcal{L}_{\theta_t}(z)\big]\big). \tag{2.8}$$

It has been shown that IF is suitable for data poisoning [Koh and Liang, 2017]. In fact, IF can also be used directly for training-time attacks. By comparing (2.8) with Corollary 2.1, IF can be viewed as an approximation of OC by ignoring the $\eta\mathbf{B}_{\theta_t}^{x_t}\nabla_\theta\mathcal{L}_{\theta_t}(x_t) = (\eta/2)\nabla_x\|\nabla_\theta\mathcal{L}_{\theta_t}(x_t)\|^2$ term. Mathematically, this term tends to reform the input to increase the norm of model gradients $\|\nabla_\theta\mathcal{L}_{\theta_t}(x_t)\|^2$. This is indeed observed in our experiments and we discover that the addition of this term can potentially further intervene the training, thus resulting in slightly better attacks.

However, OC and IF are restrictive in two aspects: (a) They require the existence of $\theta^\star$ and access to its tractable expression, which is only guaranteed with a level-2 attacker and a positive definite Hessian. (b) Even with access to a tractable expression of $\theta^\star$, OC and IF may not reach $\theta^\star$ in the long run because the defender (i.e., if it reasons at level $k \geq 1$) or the model training itself may undo the contraction, thus potentially increasing $\|\theta_{t+1} - \theta^\star\|$ over t even though the attacker tries to minimize it. So, our level-2 attack

strategy in Theorem 2.2 is more general since it does not require the existence of (or any knowledge about) $\theta^\star$. An extensive empirical comparison in Chapter 2.5.2 shows that our level-2 attack strategy is indeed more effective. During implementation, we follow the work of [Koh and Liang, 2017] and approximate the Hessian via conjugate gradients. A few adversarial training methods [Li et al., 2018, Miyato et al., 2019] can be classified as second-order attack strategies (albeit based on a different utility function (Chapter 2.2.1) and without accounting for defense) and approximate the calculation of the Hessian by power iterations.

2.3.4 R2T2 Level-$k \geq 2$ Strategies

Deriving the optimal level-$k \geq 2$ strategies requires finding the optimum of $k \geq 2$-th-order polynomials with constraints, which is generally intractable. Calculating the optimal higher-level strategy tractably may not be of practical interest since it involves the computation of higher-order gradients, which is time-consuming for modern deep learning models due to huge no. of parameters. So, we resort to approximation methods to solve for level-$k \geq 2$ strategies. Some previous works can be viewed as heuristics to approximate higher-level test-time attacks by *projected gradient descent* (PGD) [Kurakin et al., 2017, Madry et al., 2018] that do not account for defenses.

In this subsection, we propose different training-time PGDs to approximate the optimal level-$k \geq 2$ attack and defense strategies under our R2T2 framework. We show that the convergence of our PGDs can be guaranteed under convexity and other mild conditions and the number of PGD steps required to approximate the optimal level-$k \geq 2$ strategy to a constant accuracy is exponential in k.

Remark 2.1 (**Attacker's *nested PGD* (NPGD)**). *The following NPGD approximates the optimal level-$k \geq 2$ attack strategy $\delta_t^{k^*}$ at input x_t with $\delta_{[i]}$ and starts from PGD step $i = 0$ with $\delta_{[0]} = \sigma_{[0]} = \mathbf{0}$:*

$$\begin{aligned}\delta_{[i+1]} &= \text{Proj}\left(\delta_{[i]} + (\Gamma_k/\epsilon)\nabla_x \mathcal{U}_t\big(x'_{[i]} + \epsilon\,\sigma_{[i]}\big)\right), \\ \sigma_{[i+1]} &= \text{Proj}\left(\sigma_{[i]} - (\Gamma_{k-1}/\epsilon)\nabla_x \mathcal{U}_t(x''_{[i+1]})\right)\end{aligned} \tag{2.9}$$

where $x'_{[i]} \triangleq x_t + \epsilon\,\delta_{[i]}$, $x''_{[i+1]} \triangleq x'_{[i+1]} + \epsilon\,\sigma_{[i]}$, Γ_k *defines the step size to approximate* $\delta_t^{k^*}$, $\text{Proj}(x) \triangleq \vec{\mathbf{e}}(x) \times \min(\|x\|, 1)$ *is a projection to ensure* $\|\delta_{[i+1]}\|, \|\sigma_{[i+1]}\| \leq 1$, *and* $\nabla_x \mathcal{U}_t(x)$ *is given in* (2.7). *For image inputs in our real-world experiments, we also ensure that the pixel values are within the valid range at all time.*

Remark 2.2 (**Defender's PGD**). *The following PGD approximates the optimal level-*$k \geq 2$ *defense strategy* $\sigma_t^{k^*}$ *at input* x'_t *with* $\sigma_{[i]}$ *and starts from PGD step* $i = 0$ *with* $\sigma_{[0]} = \mathbf{0}$*:*

$$\sigma_{[i+1]} = \text{Proj}\left(\sigma_{[i]} - (\Gamma_k/\epsilon)\nabla_x \mathcal{U}_t(x''_{[i]})\right) \tag{2.10}$$

where $x''_{[i]} \triangleq x'_t + \epsilon\,\sigma_{[i]}$.

By comparing (2.9) and (2.10), it can be observed that the attacker follows a more complicated PGD because in each iteration t, the attacker adds its perturbations first and hence does not observe how the defense strategy (which adapts to the attacks) transforms the perturbed inputs in the same iteration. This then requires the attacker to additionally know the level-$(k-1)$ defense strategy $\sigma_t^{(k-1)^*}$ to perform PGD, which cannot be computed analytically in practice. So, it utilizes our proposed NPGD which approximates $\sigma_t^{(k-1)^*}$ with $\sigma_{[i]}$ by recursively reasoning about the defender's behavior (i.e., second PGD step of (2.9)).

Theorem 2.3 (**Convergence of defender's PGD**). *Suppose that* $\mathcal{U}_t(x)$ *is a convex function of* x. *Then, there exists a constant* M *such that by setting the step size* $\Gamma_k \triangleq 1/[M^2k^2(1+2M\epsilon)^{k-2}]$, *the defender's PGD* (2.10) *has the following convergence guarantee in PGD step* i*:*

$$\left|\mathcal{U}_t\big(x'_t + \epsilon\,\sigma_{[i]}\big) - \mathcal{U}_t\big(x'_t + \epsilon\,\sigma_t^{k^*}(x'_t)\big)\right| \leq 2\epsilon^2/(\Gamma_k i)\ .$$

Furthermore, if $\mathcal{U}_t(x)$ *is* $\mu-$*strongly convex, then*

$$\left\|\sigma_{[i]} - \sigma_t^{k^*}(x_t')\right\| \leq \exp(-\mu\Gamma_k i) \ .$$

Its proof is in Appendix A.2.3 and builds upon that for the convergence guarantee of PGD [Nesterov, 2018]. If the attacker's approximation of $\sigma_t^{(k-1)^*}$ is accurate, then the attacker's NPGD enjoys a similar guarantee as the defender's PGD (Appendix A.2.3). A key ingredient for deriving Theorem 2.3 is Assumption 2.1 which turns out to constrain the smoothness of the loss function at a particular reasoning level. Note that the convexity assumptions are not guaranteed to hold in practice. However, they are only adopted for the purpose of the theoretical analysis and thus not strictly required in practice for our PGD to deliver an effective performance.

Setting the step size and number of PGD steps. The convergence guarantee indicates that a desired accuracy can be achieved with a step size of $\Gamma_k \triangleq 1/[M^2k^2(1+2M\epsilon)^{k-2}]$ in $i = \mathcal{O}(k^2(1+2M\epsilon)^k)$ PGD steps where M is a small constant that can be estimated heuristically (Appendix A.2.3.2). In our experiments, we thus approximate the optimal level-k attack and defense strategies, respectively, through (2.9) and (2.10) by heuristically estimating M and setting the number of PGD steps according to Appendix A.2.3.2. When $\Gamma_1 \triangleq \infty$, performing 1 step of the attacker's NPGD and defender's PGD exactly recovers the level-1 strategies in Theorem 2.1. Both the attacker's NPGD and the defender's PGD can be computed for all the inputs of the minibatch in parallel using modern deep learning libraries.

2.3.5 Unifying Existing Adversarial ML Methods under our R2T2 Framework

Table 2.1 shows how our R2T2 framework can unify various existing adversarial ML methods and classify them as attackers or defenders with different reasoning levels.

Table 2.1: Unifying existing adversarial ML methods under our R2T2 framework which classifies them as attackers or defenders with different reasoning levels. Level-k (i.e., last row) means that the method is not restricted to a specific reasoning level. The methods marked with * can be easily adapted to the setting of training-time adversarial ML considered in our work here, while the methods marked with † explicitly consider adversarial defense.

Reasoning Level of Attacker	Reasoning Level of Defender	Adversarial ML Method	Perturbation Constraint	Original Paradigm
1	0	DC [Feng et al., 2019]*	L_∞	Training-time attack
	0	FGSM [Goodfellow et al., 2015]	L_∞	Test-time attack
	0	ATN [Baluja and Fischer, 2018]	Any	Test-time attack
	k	defense-GAN [Samangouei et al., 2018]*†	Any	Test-time defense
2	0	IF [Koh and Liang, 2017]*	L_2	Data poisoning
	0	VAT [Miyato et al., 2019], S-O [Li et al., 2018], CRT [Singla and Feizi, 2020]	L_2	Adversarial training
k	0	FGSM-PGD [Kurakin et al., 2017], FGSM-PGD [Madry et al., 2018]	L_∞	Adversarial training
	k	R2T2 (Ours)	L_2	Training-time attack/defense

2.4 A Synthetic Experiment

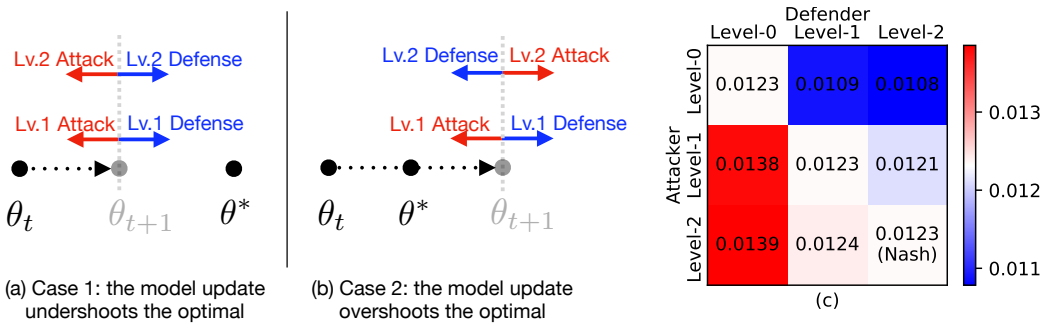

Figure 2.3: Behaviors of level-1 and level-2 players when the original updated model parameters θ_{t+1} (i.e., not subject to attack/defense) (a) undershoots and (b) overshoots the optimal parameters θ^*. (c) shows mean test loss after 30 training iterations.

We first use a synthetic experiment to show that by reasoning at a higher level, a player is able to find a better strategy by behaving more intelligently. We consider the game of a linear regression task with the ground truth model $y(x) = \theta^{*\top} x$ and squared error loss function $\mathcal{L}_{\theta_t}(x) = \|\theta_t^\top x - \theta^{*\top} x\|^2$. In iteration t, the attacker intends to increase $\|\theta_{t+1} - \theta^*\|$ to prevent the model from learning the optimal parameters θ^*; meanwhile, the defender attempts to decrease it. In this experiment, the optimal level-1 and level-2

strategies can be computed tractably.[6] The results reveal that when the original updated model parameters θ_{t+1} (i.e., not subject to attack/defense) undershoots θ^* (Fig. 2.3a), both level-1 and level-2 attackers (defenders) behave correctly since both attackers (defenders) learn to move θ_{t+1} away from (towards) θ^*. However, in the more challenging scenario where θ_{t+1} overshoots θ^* (Fig. 2.3b), only the level-2 attacker (defender) is able to increase (decrease) the distance between θ_{t+1} and θ^*, while the level-1 attacker (defender) fails, as explained in Appendix A.1.2.1. Interestingly, in this experiment, Nash equilibrium is attained from level 2 onwards, i.e., every optimal level-$k \geq 2$ strategy is the Nash equilibrium strategy (Appendix A.1.2.1). Fig. 2.3c reveals the benefit of reasoning at a higher level. Fig. 2.3c also shows that when the opponent's level is fixed at 0, reasoning at a higher level (2 instead of 1) still brings benefit, although the gain diminishes (i.e., proceeding from level 1 to 2 offers less marginal benefit than from progressing level 0 to 1).

2.5 Real-world Experiments and Discussion

Table 2.2: Mean test accuracy (%) $\pm$ 1 standard deviation (i.e., over 5 runs) for attacks against level-0 and level-1 defenders.

	MNIST		CIFAR-10		CIFAR-100	
Defender (Ours) / Attacker	Level-0	Level-1	Level-0	Level-1	Level-0	Level-1
No attack	98.8 ± 0.5	97.2 ± 0.8	74.8 ± 3.5	71.4 ± 2.2	55.4 ± 1.3	56.4 ± 1.9
FGSM-PGD [Madry et al., 2018]	80.5 ± 2.2	$\mathbf{91.3 \pm 4.5}$	45.6 ± 1.7	57.4 ± 2.6	40.5 ± 2.9	44.6 ± 3.8
CW [Carlini and Wagner, 2017]	75.4 ± 4.9	92.8 ± 3.2	50.7 ± 2.7	63.4 ± 3.7	38.4 ± 2.5	43.8 ± 3.6
YOPO [Zhang et al., 2019]	87.5 ± 2.3	94.0 ± 3.8	45.2 ± 4.3	57.0 ± 2.5	39.6 ± 2.5	42.9 ± 4.9
DC [Feng et al., 2019]	28.5 ± 6.4	96.1 ± 1.9	24.7 ± 5.6	62.5 ± 5.3	19.6 ± 6.6	43.1 ± 3.9
Level-1 (Ours)	$\mathbf{9.5 \pm 2.9}$	94.8 ± 2.8	17.1 ± 3.2	$\mathbf{55.3 \pm 4.0}$	$\mathbf{7.6 \pm 1.5}$	$\mathbf{42.8 \pm 5.2}$
Level-1 *black-box* (Ours)	11.7 ± 2.2	95.5 ± 1.7	$\mathbf{15.8 \pm 2.7}$	60.9 ± 5.9	7.9 ± 1.9	44.2 ± 3.1

We perform real-world experiments on MNIST, CIFAR-10/100, and a sub-sampled 2-class ImageNet dataset (2000 images of size $256 \times 256 \times 3$ each) of *owl* and *jellyfish*. All inputs are images whose pixel values lie within $[0, 1]$. For MNIST, CIFAR-10, and

[6] More details, including the tractable expressions of the optimal attack/defense strategies, are provided in Appendix A.1.2.1.

ImageNet, we set ϵ to be 3, 2, 4;[7] minibatch size to be 500, 400, 100; target ML model to be *convolutional neural networks* (CNN) with around 0.1M, 1M, and 1M parameters, respectively. We split 5000 images from the training data of MNIST and CIFAR, and 250 images from the ImageNet to serve as the validation set for each task. For other methods under comparison, we normalize their perturbations (transformations) to enforce the bounded L_2 norm constraint. Appendix A.1.1 gives more details on the network architectures, hyperparameter settings, among others. Besides the real-world experiments, we have also conducted a synthetic experiment (Appendix 2.4) which has tractable Nash Equilibrium strategies to demonstrate that by reasoning at a higher level, a player is able to find a better strategy by behaving more intelligently.

2.5.1 R2T2 Level-1 Strategies

R2T2 Level-1 Attack. We compare the performance of our level-1 attack against that of other attack methods including test-time attack/adversarial training baselines of FGSM-PGD [Madry et al., 2018], CW [Carlini and Wagner, 2017], and YOPO [Zhang et al., 2019], and the training-time attack baseline of DC [Feng et al., 2019]. For FGSM-PGD, we perform 5 PGD steps in total. For CW, we have followed its original implementation (with L_2 constraint and hinge loss on logits). For YOPO, we have implemented YOPO-5-3.

Baselines (except DC) are not designed for training-time attack and thus do not attack well even if against a null-strategy defender, as shown in Table 2.2. Our level-1 attack outperforms the other tested attack methods and decreases the test accuracy considerably in most cases (Table 2.2 and Fig. 2.4a), except when against our level-1 defender on MNIST. The results on CIFAR-10 (Table 2.2) shows that our black-box level-1 attacks (Chapter 2.2) can potentially outperform our white-box attacks, which agrees with recent discoveries on test-time attacks [Tramèr et al., 2018]. In Fig. 2.4a, note that DC achieves better attack performance in the early iterations (<50 epochs) because its attacker is pre-trained using

[7] An image perturbation/transformation scaled by $\epsilon = 3, 2, 4$ for MNIST, CIFAR, and Imagenet has an equivalent bounded L_2 norm as that under the bounded L_∞ norm constraint scaled by $\varepsilon = 0.107, 0.036, 0.009$, respectively.

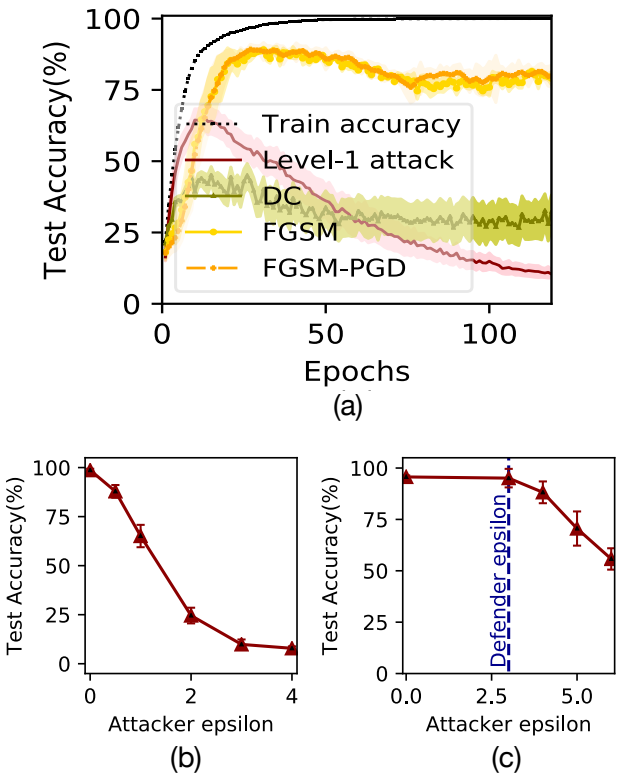

Figure 2.4: (a) Learning curves for various attacks on MNIST against level-0 defender, and test accuracy for our level-1 attack on MNIST with varying ϵ_{atk} against (b) level-0 and (c) level-1 defenders. *Lower* accuracy reflects *better* attack performance. Error bars indicate standard deviations over 5 runs.

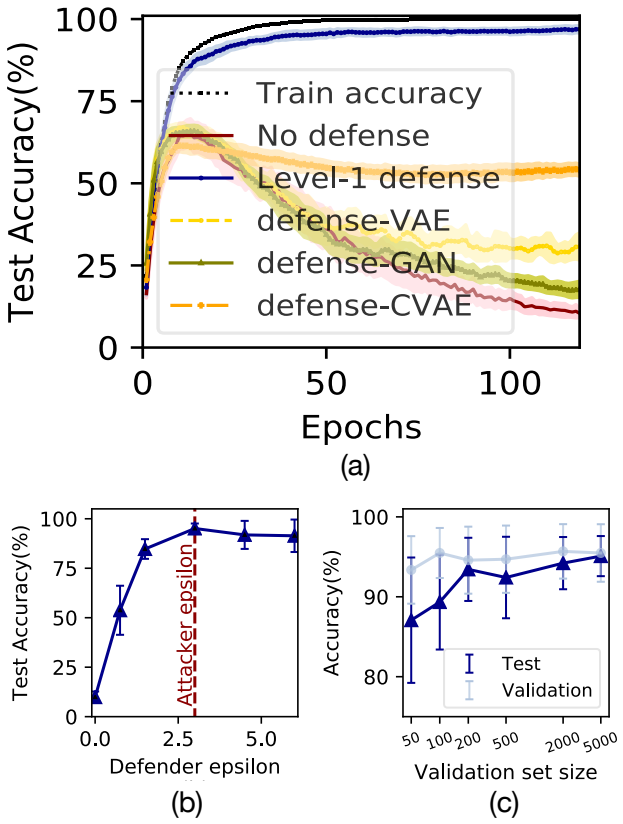

Figure 2.5: (a) Learning curves for various defenses on MNIST against level-1 attacker, and test accuracy for our level-1 defense on MNIST with varying (b) ϵ_{def} and (c) size of validation set known to the defender. *Higher* accuracy reflects *better* defense performance. Error bars indicate standard deviations over 5 runs.

the entire training dataset before the game starts. Figs. 2.4b and 2.4c show that ϵ_{atk} should not be too small for the attacker to deliver effective attacks and a level-1 defense will require ϵ_{atk} to be increased for the attacker to achieve similar attack effectiveness.

R2T2 Level-1 Defense. Under level-1 attack, we compare the performance of our level-1 defense with that of the other defense methods such as defense-VAE [Li and Ji, 2019], defense-GAN [Samangouei et al., 2018], and defense-CVAE where we replace the VAE by a *conditional VAE* (CVAE). The generative models of those defense baselines (implemented with CNN) are trained on the known validation set. The GAN is carefully tuned to avoid severe mode collapse.

The results are shown in Fig. 2.5a. Our level-1 defense strategy is proven to be effective and largely outperforms existing methods, and it is also effective when defending against

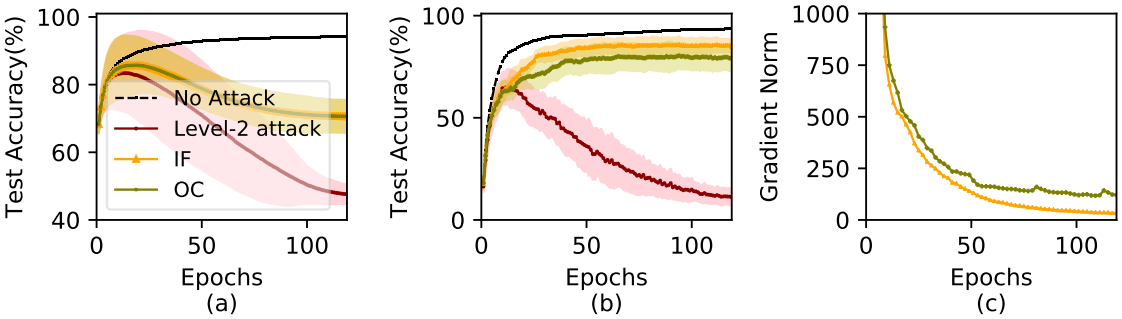

Figure 2.6: Test accuracy of (a) binary classifier and (b) CNN classifier for 3 different level-2 attack methods, and (c) gradient norm $\sum_{x\in\mathcal{D}_t} \|\nabla_\theta \mathcal{L}_{\theta_t}(x)\|_2$ for CNN classifier. The shaded regions indicate standard deviations over 10 runs.

the other attack methods (Table 2.2). Fig. 2.5b shows that ϵ_{def} should be in a scale similar to ϵ_{atk} to achieve good defense performance, and further increasing ϵ_{def} may result in slight overfitting to the validation set. Fig. 2.5c shows that when a much smaller validation set is used, overfitting to the validation set can also be observed.

2.5.2 Level-2 Attacks: R2T2 Level-2, OC, and IF

We examine three attacks discussed in Chapter 2.3.3: our R2T2 level-2 attack (Theorem 2.2), its variant called OC, and IF [Koh and Liang, 2017]. We first attack the training of a binary logistic regression classifier on a subsampled MNIST dataset of digits 1 vs. 7. This is a proof-of-concept experiment where the Hessian of the model is guaranteed to be positive definite. We then attack the training of a CNN (10-class) where the Hessian is not guaranteed to be positive definite.

It can be observed from Fig. 2.6 that IF and OC perform nearly identically for binary classification but differently for CNN. As analyzed in Chapter 2.3.3, IF ignores the "gradient increment" term $(\eta/2)\nabla_x \|\nabla_\theta \mathcal{L}_{\theta_t}(x)\|_2^2$ and thus results in a smaller gradient norm in the CNN experiment (Fig. 2.6c), which potentially explains why IF is outperformed by OC.

R2T2 level-2 attack outperforms OC and IF as we expected (Chapter 2.3.3). Though the performance improvement is already noticeable in binary classification, our R2T2 level-2 attack outperforms OC and IF by an even larger margin in CNN, potentially because

our R2T2 level-2 attack is not subject to the constraint of a positive definite Hessian, as explained in Chapter 2.3.3.

2.5.3 R2T2 Level-k Strategies

We investigate further using CIFAR-10 and ImageNet. We approximate the optimal level-k attack and defense strategies, respectively, through (2.9) and (2.10) by heuristically setting the number of PGD steps according to Appendix A.2.3.2. From Fig. 2.7a, inspection of a specific column or row reveals that reasoning at a higher level can still bring benefit even if against a fixed-level opponent. From Fig. 2.7b, it is more evident that in such a game, the marginal benefit of reasoning at a higher level quickly diminishes as a player's reasoning level goes beyond 1 (e.g., level-1 defender can improve over level 0 by 28% accuracy but level-4 defender can only improve over level 3 by 1%). Meanwhile, Fig. 2.7b also shows that the time needed to compute a higher-level strategy increases significantly with the level, which is in accordance with our theoretical analysis in Chapter 2.3.4. This implies that in practice, a player can only gain limited marginal benefit by executing significantly more steps of NPGD/PGD.

From the first row of Fig. 2.7c, we can observe that even if against a null-strategy attacker, a level-$k \geq 1$ defender improves test accuracy over a null-strategy defender, thus implying that our defense mechanism can potentially outperform perfect reconstruction (i.e., level-0 attacker vs. level-0 defender). This is indeed true as the defense alone can make the training process robust and improvement in the test accuracy can be due to the defender utilizing the clean validation set as an "augmentation" to the training data. Such a phenomenon is not always obvious on simple datasets like MNIST (Table 2.2) or CIFAR-10 (Fig. 2.7a), but is also observed in the more complex dataset CIFAR-100 (Table 2.2). The first row of Fig. 2.7d shows that the defense sometimes negates the attack's perturbation (in our visualization, green is the negation of purple), while the second row shows that the defense is sometimes more abstract to interpret. More visualizations are provided in Appendix A.1.2. The relationship between test accuracy and time needed to

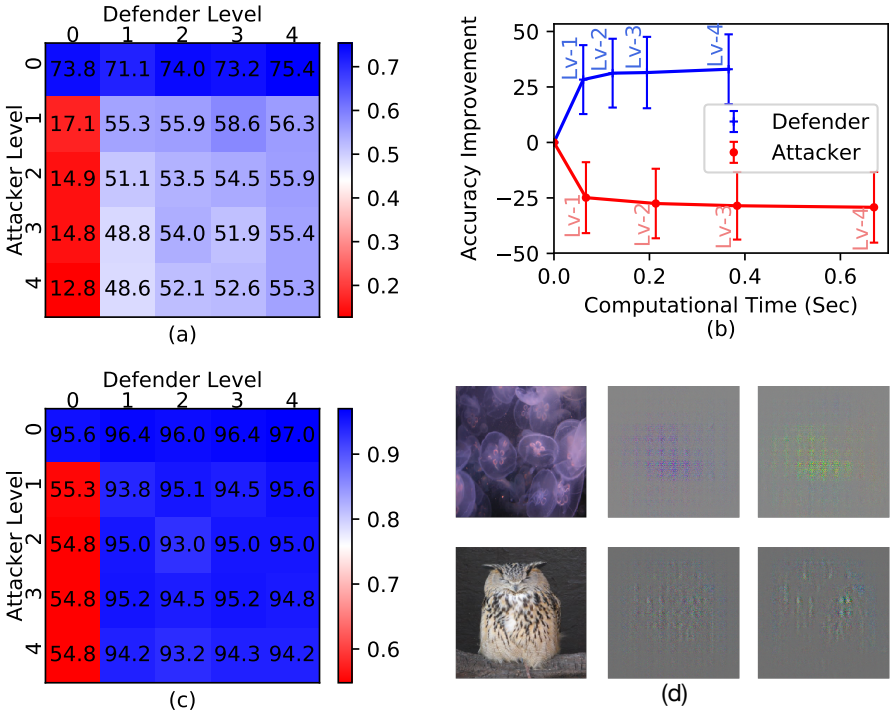

Figure 2.7: (a) Mean test accuracy (%) on CIFAR-10. (b) Graph of mean accuracy improvement (over null strategy) vs. time needed to compute the strategies given a minibatch of CIFAR-10. (c) Mean test accuracy for 2-class ImageNet. (d) From left to right are visualizations of the original sampled images from ImageNet, perturbations from level-1 attacks, and transformations from level-1 defenses, respectively (color rescaled for viewing).

compute the strategies for ImageNet and MNIST, and a table of test accuracy for MNIST are also provided in Appendix A.1.2, both of which show similar trends to the above.

2.5.4 A Generalization to Targeted Attacks

We also investigate the case when the attacker is *targeted*. The first scenario is the *conversion attack* where the attacker intends to misguide the classifier model into outputting a targeted label during test time. The attacker's utility function can be represented as $\mathcal{U}_t^\star$ and the base game becomes

$$\textbf{Attacker:} \quad \max_{\delta_t(x_t):\|\delta_t(x_t)\|\leq 1} \mathcal{U}_t^\star(x_t'') \,,$$

$$\textbf{Defender:} \quad \max_{\sigma_t(x_t'):\|\sigma_t(x_t')\|\leq 1} -\mathcal{U}_t(x_t'') \,.$$

where $\mathcal{U}_t^\star \triangleq \sum_{z\in\mathcal{D}_{\text{val}}^X}\left[\mathcal{L}_{\theta_{t+1}}^\star(z) - \mathcal{L}_{\theta_t}^\star(z)\right]$ and $\mathcal{L}_{\theta_t}^\star(z)$ is the loss incurred if the prediction of z is not the targeted label. Note that we assume that the defender is not targeted and just wants to improve the test performance, hence adopting the same utility function as (2.5).

For the results in Table 2.3, we evaluate the performance of the attack by the conversion rate, that is, how many test predictions (whose ground-truth is not the targeted label) are successfully turned into the targeted label. Interestingly, it can be observed that the conversion attack on MNIST is not very successful but is relatively successful on CIFAR-10, and the level-1 defender (even when it has a general utility function) can drastically reduce the success rate of the conversion attack.

Table 2.3: Mean attack conversion rate (%) $\pm$ 1 standard deviation (i.e., over 5 runs) for targeted attacks against level-0 and level-1 defenders. The higher the better.

	MNIST		CIFAR-10	
Attacker \ Defender	Level-0	Level-1	Level-0	Level-1
Level-1	23.7±9.6	2.8±2.3	78.1±3.0	15.7±9.7

Another targeted attack scenario is the *evasion attack* where the attacker's goal is to guide the model into *not* outputting a correct targeted label during test time. So now, $\mathcal{L}_{\theta_t}^\star(z)$ is the loss incurred if the prediction of z, whose ground truth is the targeted label, is correct.

For the results in Table 2.4, we evaluate the performance of the attack by the evasion rate, that is, how many test predictions (whose ground truth is the targeted label) are successfully turned into another label. Again, it can be observed that the evasion attack on CIFAR-10 is much more successful than on MNIST, possibly because it is easier to train well with MNIST to yield a more robust model against such attacks, and the level-1 defender (even when it has a general utility function) can largely reduce the success rate of the evasion attack.

Table 2.4: Mean attack evasion rate ($\%$) $\pm$ 1 standard deviation (i.e., over 5 runs) for targeted attacks against level-0 and level-1 defenders. The higher the better.

	MNIST		CIFAR-10	
Attacker \ Defender	Level-0	Level-1	Level-0	Level-1
Level-1	37.5±12.0	14.4±10.3	98.1±0.8	20.4±13.4

2.5.5 A Generalization to Black-Box Attack Scenario

We examine a different scenario where the model parameters and the model architecture are not known to the attacker. Attacks under such scenario are known as *black-box attacks*. To realize black-box attacks, we assume the attacker possesses the entire training dataset.[8] This is a common assumption so that the attacker can train a local model to reason about the training process.

Before the game starts, the attacker trains a local model $\theta_t^\star$ for $t = 0, \ldots, T'$, and meanwhile the attacker performs our level-k white-box attacks on the local model during training. The whole process lasts until the attack performance (i.e., validation loss) on the local model converges and the final perturbation $\delta_{T'}^{k^*}(x)$ on every input x in the training dataset is recorded down. During the game, when a minibatch (i.e., selected from the same dataset that the attacker possesses) is being transmitted to the central server, the attacker identifies each input x_t in the minibatch and adds a corresponding perturbation $\delta_{T'}^{k^*}(x_t)$ based on its record; if the training uses a different training dataset from the dataset that the attacker possesses and x_t is not in the record, then the attacker can compute the perturbation using our white-box attack strategy by setting $\theta_t = \theta_{T'}^\star$. We refer to such attack as *transferred* attack. On the other hand, since the defender is on the central server, we assume the defender always knows the true model parameter. Thus the defender always conducts white-box defenses.

The results are reflected on Table 2.5. And the result on CIFAR-10 shows that the black-box attacks could potentially outperform the white-box attacks.

[8]The training dataset that the attacker possesses does not need to be the same as the training data used for actual model training, but is assumed to be from the same underlying distribution.

Table 2.5: Mean test accuracy (%) $\pm$ standard deviation (5 runs) for attacks against level-0&1 defenders.

	MNIST		CIFAR-10	
Defender / Attacker	Level-0	level-1	Level-0	level-1
Transferred level-1	11.7$\pm$2.2	95.5$\pm$1.7	15.8$\pm$2.7	60.9$\pm$5.9

2.6 Conclusion

We proposed a principled Recursive Reasoning-based Training-Time adversarial ML (R2T2) framework to model the game between the attacker and the defender and captures their bounded reasoning capabilities (due to bounded computational resources) through the recursive reasoning formalism. Our R2T2 framework encompasses a variety of existing adversarial ML methods which correspond to attackers (defenders) with different recursive reasoning capabilities. We show how an R2T2 attacker (defender) can utilize our proposed nested projected gradient descent-based method to approximate the optimal attack (defense) strategy at an arbitrary level of reasoning. R2T2 empirically achieves state-of-the-art attack and defense performances on benchmark image datasets.

In this chapter, gradient first arises as a natural regularization, such that a deeper level of recursive reasoning is associated with the use of a higher-order gradient to derive the attack (defense) strategies. Then the first-order gradient is calculated and utilized to approximate the optimal strategies in our proposed nested projected gradient descent-based method. Such approximation can be viewed as an optimization, since conducting more steps of nested projected gradient descent yields a more optimal (higher level) strategy. In the following chapter, we shall see that a design of model architecture that allows more efficient gradient back-propagation can largely alleviates optimization difficulties, which is a clearer application of utilizing the gradient to achieve effective optimizations.

Chapter 3

Implicit Posterior Variational Inference with Gradient-bridging Design for Deep Gaussian Processes

3.1 Background and Preliminary

Gaussian process (GP) [Rasmussen and Williams, 2006] is a Bayesian non-parametric model whose expressive power can be significantly boosted by composing them hierarchically into a multi-layer *deep GP* (DGP) model, as shown in the seminal work of [Damianou and Lawrence, 2013]. The DGP model usually exploits the notion of inducing variables [Quiñonero-Candela and Rasmussen, 2005] to improve its scalability to large dataset, which introduces the inducing inputs (learnable parameters) as a representation/summarization of the dataset, and models the belief of the corresponding inducing output. Our recently developed *implicit posterior variational inference* (IPVI) framework [Yu et al., 2019] is the most advanced inference framework for DGPs that can ideally recover an unbiased posterior belief and still preserve time efficiency. To do so, IPVI framework casts the DGP inference problem as a two-player game like in the generative adversarial networks, in which an unbiased posterior belief can be achieved upon reaching

Nash equilibrium.

However, as a generator and a discriminator are integrated in each layer of the DGP, the training becomes unstable and the performances are unsatisfactory for IPVI. Particularly, we identified two possible problems: (i) Overfitting due to overly-parameterized generator; (ii) Optimization difficulties due to the inefficient gradient back-propagation. Ablation studies are performed, and eventually we identified that the optimization difficulties due to the inefficient gradient back-propagation is the main obstacle in the realization of our IPVI framework and algorithm. To this end, we proposed a novel gradient-bridging architecture to solve such optimization difficulties. We also empirically evaluate the performance of our novel IPVI architecture on several real-world datasets.

Here we introduce the relevant models (GP, DGP) and the IPVI frameworks as the preliminary.

Gaussian Process (GP). Let an unknown function f be distributed by a GP with a zero prior mean and covariance function k. That is, suppose that for some set $\mathbf{X}$ of training inputs, a set $\mathbf{y} = \{y_\mathbf{x}\}_{\mathbf{x}\in\mathbf{X}}$ of noisy observed outputs $y_\mathbf{x} \triangleq f(\mathbf{x})+\varepsilon(\mathbf{x})$ (i.e., corrupted by an i.i.d. Gaussian noise $\varepsilon(\mathbf{x})$ with noise variance ν^2) are available. Then, the set $\mathbf{f}_\mathbf{x} \triangleq \{f(\mathbf{x})\}_{\mathbf{x}\in\mathbf{X}}$ of latent outputs follow a Gaussian prior belief $p(\mathbf{f}) \triangleq \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K}_{\mathbf{X}\mathbf{X}})$ where $\mathbf{K}_{\mathbf{X}\mathbf{X}}$ denotes a covariance matrix with components $k(\mathbf{x}, \mathbf{x}')$ for $\mathbf{x}, \mathbf{x}' \in \mathbf{X}$. It follows that $p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \nu^2\mathbf{I})$. The GP predictive/posterior belief of the latent outputs $\mathbf{f}^\star \triangleq \{f(\mathbf{x}^\star)\}_{\mathbf{x}^\star\in\mathbf{X}^\star}$ for any set $\mathbf{X}^\star$ of test inputs can be computed in closed form [Rasmussen and Williams, 2006] by marginalizing out $\mathbf{f}$: $p(\mathbf{f}^\star|\mathbf{y}) = \int p(\mathbf{f}^\star|\mathbf{f})\, p(\mathbf{f}|\mathbf{y})\, \mathrm{d}\mathbf{f}$ but incurs cubic time in the number of data points, hence scaling poorly to massive datasets. To improve its scalability to linear time in the number of data points, the *sparse GP* (SGP) models spanned by the unifying view of [Quiñonero-Candela and Rasmussen, 2005] exploit a set $\mathbf{u} \triangleq \{u_m \triangleq f(\mathbf{z}_m)\}_{m=1}^M$ of inducing output variables for some small set $\mathbf{Z} \triangleq \{\mathbf{z}_m\}_{m=1}^M$ of inducing inputs (i.e., $M \ll |\mathbf{X}|$). Then,

$$p(\mathbf{y}, \mathbf{f}, \mathbf{u}) = p(\mathbf{y}|\mathbf{f})\, p(\mathbf{f}|\mathbf{u})\, p(\mathbf{u}) \tag{3.1}$$

such that

$$p(\mathbf{f}|\mathbf{u}) = \mathcal{N}(\mathbf{f}|\mathbf{K_{XZ}}\mathbf{K_{ZZ}^{-1}}\mathbf{u}, \mathbf{K_{XX}} - \mathbf{K_{XZ}}\mathbf{K_{ZZ}^{-1}}\mathbf{K_{ZX}})$$

where $\mathbf{u}$ is treated as a column vector here, $\mathbf{K_{XZ}} \triangleq \mathbf{K_{ZX}^{\top}}$, $\mathbf{K_{ZZ}}$ and $\mathbf{K_{ZX}}$ denote covariance matrices with components $k(\mathbf{z}_m, \mathbf{z}_{m'})$ for $m, m' = 1, \ldots, M$ and $k(\mathbf{z}_m, \mathbf{x})$ for $m = 1, \ldots, M$ and $\mathbf{x} \in \mathbf{X}$, respectively. The SGP predictive belief can also be computed in closed form by marginalizing out $\mathbf{u}$: $p(\mathbf{f}^\star|\mathbf{y}) = \int p(\mathbf{f}^\star|\mathbf{u})\, p(\mathbf{u}|\mathbf{y})\, \mathrm{d}\mathbf{u}$.

The work of [Titsias, 2009a] has proposed a principled *variational inference* (VI) framework that approximates the joint posterior belief $p(\mathbf{f}, \mathbf{u}|\mathbf{y})$ with a variational posterior $q(\mathbf{f}, \mathbf{u}) \triangleq p(\mathbf{f}|\mathbf{u})\, q(\mathbf{u})$ by minimizing the *Kullback-Leibler* (KL) distance between them, which is equivalent to maximizing a lower bound of the log-marginal likelihood (i.e., also known as the *evidence lower bound* (ELBO)):

$$\mathrm{ELBO} \triangleq \mathbb{E}_{q(\mathbf{f})}[\log p(\mathbf{y}|\mathbf{f})] - D_{\mathrm{KL}}[q(\mathbf{u})\|p(\mathbf{u})]$$

where $q(\mathbf{f}) \triangleq \int p(\mathbf{f}|\mathbf{u})\, q(\mathbf{u})\, \mathrm{d}\mathbf{u}$. A common choice in VI is the Gaussian variational posterior $q(\mathbf{u}) \triangleq \mathcal{N}(\mathbf{u}|\mathbf{m}, \mathbf{B})$ of the inducing variables $\mathbf{u}$ [Deisenroth and Ng, 2015, Gal et al., 2014, Hensman et al., 2013, Hoang et al., 2015, Hoang et al., 2016, Titsias, 2009b] which results in a Gaussian marginal $q(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where

$$\begin{aligned}\boldsymbol{\mu} &\triangleq \mathbf{K_{XZ}}\mathbf{K_{ZZ}^{-1}}\mathbf{m}\\ \boldsymbol{\Sigma} &\triangleq \mathbf{K_{XX}} - \mathbf{K_{XZ}}\mathbf{K_{ZZ}^{-1}}(\mathbf{K_{ZZ}} - \mathbf{B})\mathbf{K_{ZZ}^{-1}}\mathbf{K_{ZX}}.\end{aligned}$$

Deep Gaussian Process (DGP). A multi-layer DGP model is a hierarchical composition of GP models. Consider a DGP with a depth of L such that each DGP layer is associated with a set $\mathbf{f}_{\ell-1}$ of inputs and a set $\mathbf{f}_\ell$ of outputs for $\ell = 1, \ldots, L$ and $\mathbf{f}_0 \triangleq \mathbf{X}$. Let $\mathbf{F} \triangleq \{\mathbf{f}_\ell\}_{\ell=1}^{L}$, and the inducing inputs and corresponding inducing output variables for DGP layers $\ell = 1, \ldots, L$ be denoted by the respective sets $\{\mathbf{Z}_\ell\}_{\ell=1}^{L}$ and $\mathbf{U} \triangleq \{\mathbf{u}_\ell\}_{\ell=1}^{L}$. Similar to the

joint probability distribution of the SGP model (3.1),

$$p(\mathbf{y}, \mathbf{F}, \mathbf{U}) = \underbrace{p(\mathbf{y}|\mathbf{f}_L)}_{\text{data likelihood}} \left[\prod_{\ell=1}^{L} p(\mathbf{f}_\ell|\mathbf{u}_\ell)\right] p(\mathbf{U}).$$

Similarly, the variational posterior is assumed to be $q(\mathbf{F}, \mathbf{U}) \triangleq \left[\prod_{\ell=1}^{L} p(\mathbf{f}_\ell|\mathbf{u}_\ell)\right] q(\mathbf{U})$, thus resulting in the following ELBO for the DGP model:

$$\text{ELBO} \triangleq \int q(\mathbf{f}_L) \log p(\mathbf{y}|\mathbf{f}_L)\, \mathrm{d}\mathbf{f}_L - D_{\text{KL}}[q(\mathbf{U})\|p(\mathbf{U})] \tag{3.2}$$

where $q(\mathbf{f}_L) \triangleq \int \prod_{\ell=1}^{L} p(\mathbf{f}_\ell|\mathbf{u}_\ell, \mathbf{f}_{\ell-1})\, q(\mathbf{U})\, \mathrm{d}\mathbf{f}_1 \ldots \mathrm{d}\mathbf{f}_{L-1}\mathrm{d}\mathbf{U}$. To compute $q(\mathbf{f}_L)$, the work of [Salimbeni and Deisenroth, 2017] has proposed the use of the reparameterization trick [Kingma and Welling, 2013] and Monte Carlo sampling, which are also adopted in IPVI.

Implicit Posterior Variational Inference (IPVI) for DGPs. IPVI first generates posterior samples $\mathbf{U} \triangleq g_{\theta_G}(\epsilon)$ with a black-box **generator** $g_{\theta_G}(\epsilon)$ parameterized by θ_G and a random noise $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. By representing the variational posterior as $q_{\theta_G}(\mathbf{U}) \triangleq \int p(\mathbf{U}|\epsilon)\mathrm{d}\epsilon$, the ELBO in (3.2) can be re-written as

$$\begin{aligned} \text{ELBO} &= \mathbb{E}_{q(\mathbf{f}_L)}[\log p(\mathbf{y}|\mathbf{f}_L)] - D_{\text{KL}}[q_{\theta_G}(\mathbf{U})\|p(\mathbf{U})] \\ &= \mathbb{E}_{q(\mathbf{f}_L)}[\log p(\mathbf{y}|\mathbf{f}_L)] - \mathbb{E}_{q_{\theta_G}(\mathbf{U})}[T^*(\mathbf{U})]\ , \end{aligned} \tag{3.3}$$

where the calculation of the KL distance term is given by a separate function T^* to be optimized, as below:

$$T^* = \max_T \left\{\mathbb{E}_{p(\mathbf{U})}[\log(1 - \sigma(T(\mathbf{U}))] + \mathbb{E}_{q_{\theta_G}(\mathbf{U})}[\log \sigma(T(\mathbf{U}))]\right\}\ . \tag{3.4}$$

T in (3.4) is a neural network **discriminator** T_{θ_D} with parameters θ_D tring to distinguish between $q_{\theta_G}(\mathbf{U})$ and $p(\mathbf{U})$ by outputting $\sigma(T(\mathbf{U}))$ as the probability of $\mathbf{U}$ being a sample

from $q_{\theta_G}(\mathbf{U})$ rather than $p(\mathbf{U})$. Finally, this simplifies to

$$\text{ELBO} = \mathbb{E}_{q_{\theta_G}(\mathbf{U})}[\mathcal{L}(\theta, \mathbf{X}, \mathbf{y}, \mathbf{U}) - T^*(\mathbf{U})] \tag{3.5}$$

where $\mathcal{L}(\theta, \mathbf{X}, \mathbf{y}, \mathbf{U}) \triangleq \mathbb{E}_{p(\mathbf{f}_L|\mathbf{U})}[\log p(\mathbf{y}|\mathbf{f}_L)]$ and θ denotes the DGP model hyperparameters. The ELBO can now be calculated given the optimal discriminator T^*. The ELBO in (3.5) is optimized with respect to θ_G and θ via first-order gradient method, which mimics the better-response dynamics searching for Nash equilibrium in a two-player game between **Player 1** (representing discriminator with strategy $\{\theta_D\}$) vs. **Player 2** (jointly representing generator and DGP model with strategy $\{\theta_G, \theta\}$) that is defined based on the following payoffs[1]:

$$\begin{aligned} &\textbf{Player 1 :} \quad \max_{\{\theta_D\}} \mathbb{E}_{p(\mathbf{U})}[\log(1 - \sigma(T_{\theta_D}(\mathbf{U}))] + \mathbb{E}_{q_{\theta_G}(\mathbf{U})}[\log \sigma(T_{\theta_D}(\mathbf{U}))] \,, \\ &\textbf{Player 2 :} \quad \max_{\{\theta, \theta_G\}} \mathbb{E}_{q_{\theta_G}(\mathbf{U})}[\mathcal{L}(\theta, \mathbf{X}, \mathbf{y}, \mathbf{U}) - T_{\theta_D}(\mathbf{U})] \,. \end{aligned} \tag{3.6}$$

Algorithm 1 illustrates the IPVI algorithm: In each iteration, each player takes its turn to improve its strategy to achieve a better payoff by performing a stochastic gradient optimization (3.6).

3.1.1 Classification with Robust-Max Likelihood

We have discussed the learning of regression tasks above. Here we discuss the learning on probabilistic classification tasks using the robust-max likelihood. Following the work of [Hernández-Lobato et al., 2011], the likelihood for the prediction of a data pair $(\mathbf{x}, y_{\mathbf{x}})$ in a N-way classification problem given IP output $\mathbf{f}$ (where this N-dimensional output is defined as $\mathbf{f} \triangleq [f_1, f_2, \ldots, f_N]$) and a binary variable a (one per data instance to indicate

[1]If $(\{\theta_D{}^*\}, \{\theta^*, \theta_G{}^*\})$ is a Nash equilibrium of the pure-strategy game in (3.6), then $\{\theta^*, \theta_G{}^*\}$ is a global maximizer of the ELBO in (3.3) such that (a) θ^* is the maximum likelihood assignment for the DGP model, and (b) $q_{\theta_G{}^*}(\mathbf{U})$ is equal to the true posterior belief $p(\mathbf{U}|\mathbf{y})$. It reveals that any Nash equilibrium coincides with a global maximizer of the original ELBO in (3.3). More details see Appendix B.1.

Algorithm 1 IPVI

Randomly initialize θ, θ_D, θ_G
while not converged **do**
 # Player 2
Sample $\{\mathbf{U'}_1, \ldots, \mathbf{U'}_K\}$ from $p(\mathbf{U})$, and sample $\{\mathbf{U}_1, \ldots, \mathbf{U}_K\}$ from $q_{\theta_G}(\mathbf{U})$
Compute gradient w.r.t. θ_D from (3.6):

$$g_{\theta_D} \triangleq \nabla_{\theta_D}\left[\frac{1}{K}\sum_{k=1}^{K}\log(1-\sigma(T_{\theta_D}(\mathbf{U'}_k))\right] + \nabla_{\theta_D}\left[\frac{1}{K}\sum_{k=1}^{K}\log\sigma(T_{\theta_D}(\mathbf{U}_k))\right]$$

Update for θ_D: $\theta_D \leftarrow \theta_D + \alpha_{\theta_D}\, g_{\theta_D}$;
 # Player 1
Sample mini-batch $(\mathbf{X}_b, \mathbf{y}_b)$ from $(\mathbf{X}, \mathbf{y})$, and sample $\{\mathbf{U}_1, \ldots, \mathbf{U}_K\}$ from $q_{\theta_G}(\mathbf{U})$
Compute gradients w.r.t. θ and θ_G from (3.6):

$$g_\theta \triangleq \nabla_\theta\left[\frac{1}{K}\sum_{k=1}^{K}\mathcal{L}(\theta, \mathbf{X}_b, \mathbf{y}_b, \mathbf{U}_k)\right]; g_{\theta_G} \triangleq \nabla_{\theta_G}\left[\frac{1}{K}\sum_{k=1}^{K}\mathcal{L}(\theta, \mathbf{X}_b, \mathbf{y}_b, \mathbf{U}_k) - T_{\theta_D}(\mathbf{U}_k)\right]$$

Updates for θ and θ_G: $\theta \leftarrow \theta + \alpha_\theta\, g_\theta$, $\theta_G \leftarrow \theta_G + \alpha_{\theta_G}\, g_{\theta_G}$;
end while

whether an $\arg\max$ prediction is satisfied or not) is

$$p(y_{\mathbf{x}}|\mathbf{x}, \mathbf{f}, a) = \prod_{c \neq y_{\mathbf{x}}} \Theta(f_{y_{\mathbf{x}}} - f_c)^{1-a}\ (1/N)^a$$

where $\Theta(\cdot)$ is the Heaviside step function and the binary variable a is defined to follow *a priori* a factorizing multivariate Bernoulli distribution:

$$p(a|\rho) \triangleq \text{Bern}(a|\rho) = \rho^a\ (1-\rho)^{1-a}$$

such that ρ is the fraction of training data pairs expected to be outliers. The prior for ρ is defined as a conjugate beta distribution:

$$p(\rho) \triangleq \text{Beta}(\rho|a_0, b_0) = \frac{\rho^{a_0-1}\ (1-\rho)^{b_0-1}}{B(a_0, b_0)}$$

where $B(\cdot, \cdot)$ is the beta function, and a_0 and b_0 are free hyperparameters which do not have a big impact on the final model, provided that $b_0 > a_0$ and that they are not too small [Hernández-Lobato et al., 2011]. By default, we set $a_0 = 1$ and $b_0 = 9$.

3.2 Problem Formulation

As a generator and a discriminator are integrated in each layer of the DGP, the training becomes unstable and the performances are unsatisfactory. To start with, we will now discuss how the architecture of the generator in our IPVI framework is usually designed. Recall from Section 3.1 that $\mathbf{U} = \{\mathbf{u}_\ell\}_{\ell=1}^L$ is a collection of inducing variables for DGP layers $\ell = 1, \ldots, L$. We consider a layer-wise design of the generator (parameterized by $\theta_G \triangleq \{\phi_\ell\}_{\ell=1}^L$) and discriminator (parameterized by $\theta_D \triangleq \{\psi_\ell\}_{\ell=1}^L$) such that $g_{\theta_G}(\epsilon) \triangleq \{g_{\phi_\ell}(\epsilon)\}_{\ell=1}^L$ with the random noise ϵ serving as a common input to induce dependency between layers and $T_{\theta_D}(\mathbf{U}) \triangleq \sum_{\ell=1}^L T_{\psi_\ell}(\mathbf{u}_\ell)$, respectively. For each layer ℓ, a naive design is to generate posterior samples $\mathbf{u}_\ell \triangleq g_{\phi_\ell}(\epsilon)$ from the random noise ϵ as input. However, we have identified that such a design suffers from two possible critical issues:

(i) Fig. 3.1a illustrates that to generate posterior samples of M different inducing variables $\mathbf{u}_{\ell 1}, \ldots, \mathbf{u}_{\ell M}$ ($\mathbf{u}_\ell \triangleq \{\mathbf{u}_{\ell m}\}_{m=1}^M$), the naive design of the generator adopts a dense connection (as such generator is implemented by a densely-connected neural network). Such naive design introduces a relatively large number of parameters and is thus prone to overfitting (to the dataset).

(ii) Such a design of the generator fails to adequately capture the dependency of the inducing output variables $\mathbf{u}_\ell$ on the corresponding inducing inputs $\mathbf{Z}_\ell$, which leads to the optimization difficulties. Note that the inducing inputs are learnable parameters in DGP that needs to be jointly optimized. However, its optimization is rarely taken care of in the literature [Titsias and Lázaro-Gredilla, 2014, Havasi et al., 2018]. To begin with, it is initialized randomly without careful tuning: in single layer GP, it is usually initialized by K-Means on the dataset. In the contrary, in DGP, the inducing inputs in the first layer is initialized by K-Means likewise, while the inducing inputs in the consecutive layers $\ell = 1, \ldots, L$ are initialized by doing K-Means on the propagated inputs $\mathbf{f}_\ell$.[2] The propagated inputs inherits a large degree of randomness and is inaccurate (fully random)

[2]If the dimensionality is reduced during forward propagation, *principle component analysis* (PCA) is conducted before doing K-Means on the propagated inputs $\mathbf{f}_\ell$.

in the beginning of training. As a result, the initialization of the inducing inputs in the consecutive layers are inaccurate, which indicates the importance of its optimization. Secondly, the optimization of the inducing inputs are usually not well conducted. If the generator fails to capture the dependency on the corresponding inducing inputs $\mathbf{Z}_\ell$, all the terms containing the inducing inputs will otherwise be associated with (the inverse of) the Gram matrix $\mathbf{K}_{\mathbf{ZZ}}^{-1}$ as indicated by the definition of DGP. In this case, during gradient back-propagation, the gradient of the inducing inputs are also associated with the differentiation of such inversion of the Gram matrix. However, to ensure the well-conditioning of the inversion, the Gram matrix is usually post-processed, e.g., by adding a large enough center component (scalar) $\lambda\mathbf{I}$. Such post-processing further undermines the quality of the resulting gradient on the inducing inputs, hence restricting its capability to output the posterior samples of $\mathbf{U}$ accurately (Chapter 3.4).

3.3 Different Architectures of Generator and Discriminator for DGPs

3.3.1 The Old Design: Parameter-tying Architecture

To resolve the above issues, a parameter-tying architecture of the generator and discriminator is first proposed for a DGP model, as shown in Figs. 3.1b and 3.1e. For each layer ℓ, since $\mathbf{u}_\ell$ depends on $\mathbf{Z}_\ell$, the generator is designed g_{ϕ_ℓ} to generate posterior samples $\mathbf{u}_\ell \triangleq g_{\phi_\ell}(\epsilon \oplus \mathbf{Z}_\ell)$ from not just ϵ but also $\mathbf{Z}_\ell$ as inputs. Recall that the same ϵ is fed as an input to g_{ϕ_l} in each layer ℓ, which can be observed from the left-hand side of Fig. 3.1g. In addition, compared with the naive design in Fig. 3.1a, the posterior samples of M different inducing variables $\mathbf{u}_{\ell 1}, \ldots, \mathbf{u}_{\ell M}$ are generated based on only a single shared parameter setting (instead of M), which reduces the number of parameters drastically (Fig. 3.1b). A similar design is adopted for the discriminator, as shown in Fig. 3.1e.

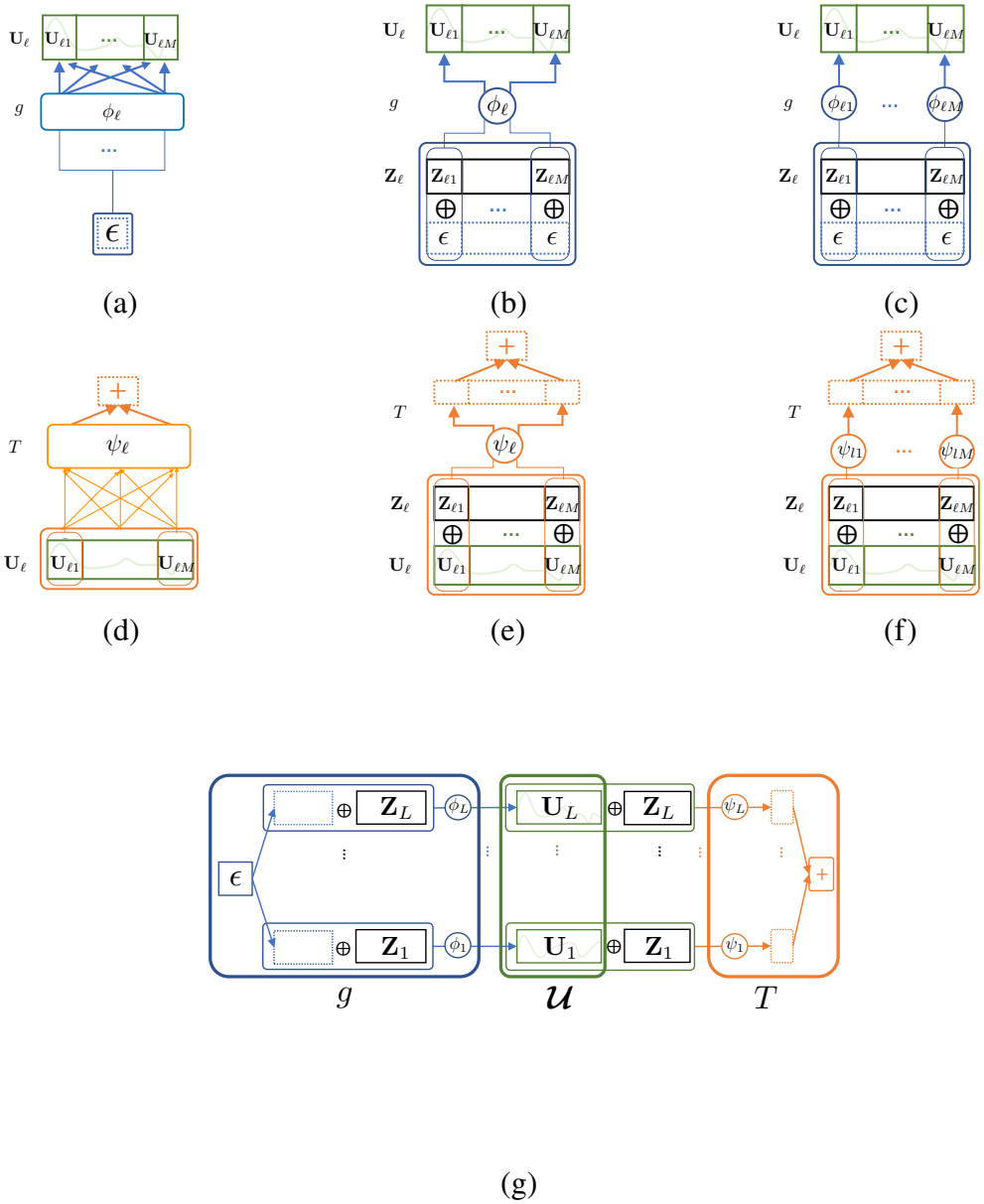

Figure 3.1: (a) Illustration of a naive design of the generator for each layer ℓ. (b) Parameter-tying architecture of the generator. (c) Gradient-bridging architecture of the generator. (d) Illustration of a naive design of the discriminator for each layer ℓ. (e) Parameter-tying architecture of the discriminator. (f) Gradient-bridging architecture of the discriminator. (g) Overall architecture of the generator and discriminator in our IPVI framework for DGPs. '$+$' denotes addition and '$\oplus$' denotes concatenation. See the main text for the definitions of notations.

3.3.2 Our Advanced Design: Gradient-bridging Architecture

The proposed parameter-tying architecture solves the potential issue of overfitting by limiting the number of parameters in the generator. That it, different inducing variables $\mathbf{u}_{\ell 1}, \ldots, \mathbf{u}_{\ell M}$ are generated based on a single shared parameter setting which greatly reduces the parameterization compared with the densely connected naive design. Limiting the number of parameters could be a good approach against severe overfitting, however, if overfitting is not the main issue for the bad performance of the naive design, doing so will largely reduce the expressiveness of the generator, thus undermines the final model whose predictions depend on the distribution produced by the generator (Chapter 3.4).

We then proposed another architecture, called the gradient-bridging architecture of the generator and discriminator for a DGP model, as shown in Figs. 3.1c and 3.1f. The design of the generator stems from the following motivation: If the overfitting is not a main issue for the unsatisfactory performance of the naive design, it is then natural for the generator to adopt M different parametric settings $\phi_{\ell 1}, \ldots, \phi_{\ell M}$ ($\phi_\ell \triangleq \{\phi_{\ell m}\}_{m=1}^M$) to maximize the expressiveness, such that $\mathbf{u}_{\ell m} \triangleq g_{\phi_{\ell m}}(\epsilon \oplus \mathbf{Z}_{\ell m})$, while still adequately captures the dependency of the inducing output variables $\mathbf{u}_\ell$ on the corresponding inducing inputs $\mathbf{Z}_\ell$. Doing so also allows faster and more accurate optimization of the inducing inputs, since the gradient back-propagation now is specific for different inducing inputs $\mathbf{Z}_{\ell m}, m = 1, \ldots, M$ and not affected by each other (since there is no parameter-tying)[3]. Experiments in Chapter 3.4 provide empirical evidence for this phenomenon. The optimization of the inducing inputs can also be meaningful even when the post-processed Gram matrix does not provide accurate gradient information, as empirically studied in Chapter 3.4. We adopt a similar design for the discriminator, as shown in Fig. 3.1f.

Fig. 3.1g illustrates the design of the overall parameter-tying architecture of the generator and discriminator.

Generator/Discriminator Details. We have described the architecture design, we will describe here the neural network represented by g_{ϕ_ℓ} of the parameter-tying architecture,

[3]Note that the inducing inputs are still statistically dependent because the same ϵ is fed as an input

or $g_{\phi_{\ell 1}}, \ldots, g_{\phi_{\ell M}}$ of the gradient-bridging architecture. Since the gradient-bridging architecture adopts M different parametric settings $\phi_{\ell 1}, \ldots, \phi_{\ell M}$, each has the same neural network architecture as ϕ_ℓ of the parameter-tying architecture, we will discuss g_{ϕ_ℓ} (and corresponding T_{ψ_ℓ}) for simplicity. Firstly, the noise ϵ has the same dimension as the inputs $\mathbf{X}$ of the dataset. We implement g_{ϕ_ℓ} using a two-layer neural network with hidden dimension being equal to the dimension of $\mathbf{Z}_\ell$ and leaky ReLU activation in the middle. Similarly, we implement T_{ψ_ℓ} using a two-layer neural network with hidden dimension being equal to the dimension of $\mathbf{Z}_\ell$ and leaky ReLU activation in the middle. The network initialization follows random normal distribution.

Although gradient-bridging architecture has M times larger parameterization as that of the parameter-tying architecture, it can be implemented in parallel in both forward and backward propagation, which does not incur much wall-clock time (see Chapter 3.4). We have observed in our own experiments that our proposed gradient-bridging architecture not only speeds up the training (converges in fewer iterations), but also improves the predictive performance of IPVI considerably. We will empirically evaluate our IPVI framework with this gradient-bridging architecture in Section 3.4.

3.4 Experiments and Discussion

We empirically evaluate and compare the performance of our gradient-bridging based IPVI framework against that of the state-of-the-art parameter-tying based IPVI, and the naive designed IPVI. Our implementation is built on GPflow [Matthews et al., 2017] which is an open-source GP framework based on TensorFlow [Abadi et al., 2016].

3.4.1 Performance Comparison: Real-world Datasets

For our experiments in the regression tasks, the depth L of the DGP models are varied from 2 to 5 with 128 inducing inputs per layer. The dimension of each hidden DGP layer is set to be (i) the same as the input dimension for the UCI benchmark regression and

Airline datasets, and (ii) 98 for the classification tasks.

UCI Benchmark Regression Datasets. Our experiments are first conducted on UCI benchmark regression datasets. We have performed a random $0.9/0.1$ train/test split.

Airline Dataset (Large-Scale Regression). We then evaluate the performance of IPVI on a real-world large-scale regression dataset, the Airline dataset, with input dimension $D = 8$ and a large data size $N \approx 2$ million. For Airline dataset, we set the last 100000 examples as test data.

In the above regression tasks, the performance metric is the mean log-likelihood (MLL) of the test data (or test MLL). Note that larger values of MLL represent better results. Table 3.1 shows results of the test MLL and standard deviation over 10 runs. It can be observed that IPVI with gradient-bridging architecture generally outperforms the others. For large-scale regression tasks, the performance of IPVI with either parameter-tying architecture or gradient-bridging architecture consistently increases with a greater depth.

Classification on Real-world Datasets. We evaluate the performance of IPVI in three classification tasks using the real-world MNIST, fashion-MNIST, and CIFAR-10 datasets. Both MNIST and fashion-MNIST datasets are grey-scale images of 28×28 pixels. The CIFAR-10 dataset consists of colored images of 32×32 pixels. We utilize a 4-layer DGP model with 100 inducing inputs per layer and a robust-max multiclass likelihood [Hernández-Lobato et al., 2011]; for MNIST dataset, we also consider utilizing a 4-layer DGP model with 800 inducing inputs per layer to assess if its performance improves with more inducing inputs. Table 3.2 reports the mean test accuracy over 10 runs, which shows that our IPVI framework with gradient-bridging architecture for a 4-layer DGP model performs the best in all three datasets.[4]

[4]Note that all the results in Table 3.2 are obtained without using *convolutional neural networks* (CNN) as mean functions of GP.

Table 3.1: Test MLL achieved by our IPVI framework with different architectures for UCI benchmark regression datasets. Higher test MLL is better.

Dataset	Boston				Power			
DGP Layers	2	3	4	5	2	3	4	5
Naive design	-2.37	-2.48	-2.51	-2.57	-2.79	-2.74	-2.73	-2.75
Parameter-tying	**-2.08**	-2.13	-2.09	-2.10	-2.69	-2.67	-2.70	-2.71
Gradient-bridging	-2.10	**-2.06**	**-2.05**	**-2.06**	**-2.67**	**-2.65**	**-2.60**	**-2.60**
Dataset	Protein				Airline			
DGP Layers	2	3	4	5	2	3	4	5
Naive design	-2.72	-2.69	-2.70	-2.67	-4.82	-4.83	-4.84	-4.87
Parameter-tying	**-2.57**	-2.56	-2.59	-2.62	-4.77	-4.75	-4.74	-4.73
Gradient-bridging	-2.59	**-2.55**	**-2.55**	**-2.57**	**-4.75**	**-4.74**	**-4.73**	**-4.72**

Table 3.2: Mean test accuracy (%) achieved by our IPVI framework with three different architectures for a 4-layer DGP on three classification datasets.

Dataset	MNIST	MNIST ($M = 800$)	fashion-MNIST	CIFAR-10
Naive design	97.45	97.56	88.01	52.76
Parameter-tying	97.80	98.23	88.90	53.27
Gradient-bridging	**98.12**	**98.42**	**89.04**	**53.50**

3.4.2 Ablation Studies

Gradient Back-propagation Property of Different Architectures. As mentioned in Chapter3.3, the gradient back-propagation in the gradient-bridging architecture is now specific for different inducing inputs $\mathbf{Z}_{\ell m}, m = 1, \ldots, M$ and not affected by each other. Doing so immediately allows faster optimization of the inducing inputs, which can be verified by evaluating the magnitude of gradient on the inducing inputs. Table 3.3 reflects the averaged gradient norm $\frac{1}{\sqrt{M}} \left\| \nabla_{\mathbf{Z}_\ell} [\frac{1}{K} \sum_{k=1}^{K} \mathcal{L}(\theta, \mathbf{X}, \mathbf{y}, \mathbf{U}_k)] \right\|_2$ during training of DGP. We can observe that, the gradient-bridging architecture allows a larger gradient back-propagation in the starting phase (1$\sim$1000 iterations) and the middle phase (1001$\sim$5000 iterations) of the training, which explains its faster convergence in terms of the total number of iterations (Table 3.4). In the ending phase (5001 iterations onward) of the training, the inducing inputs in the gradient-bridging architecture already stabilizes as we observe a relatively smaller gradient magnitude. In comparison, the inducing inputs in the parameter-tying architecture still undergoes a certain degree of optimization, as reflected

by the relatively larger gradient magnitude.

Both parameter-tying and gradient-bridging architectures have much larger magnitude of gradient than the naive design, because the dependency of the inducing outputs on the inducing inputs are explicitly considered.

Table 3.3: Averaged gradient norm $\frac{1}{\sqrt{M}}\left\|\nabla_{\mathbf{Z}_\ell}[\frac{1}{K}\sum_{k=1}^{K}\mathcal{L}(\theta,\mathbf{X},\mathbf{y},\mathbf{U}_k)]\right\|_2$ of the inducing inputs in the last layer of a 4-layer DGP model for Airline dataset.

	Naive design	Parameter-tying	Gradient-bridging
1∼1000 iter.	4.01×10^1	2.21×10^3	$\mathbf{4.89 \times 10^3}$
1001∼5000 iter.	1.44×10^2	4.14×10^3	$\mathbf{7.84 \times 10^3}$
5001∼converge	1.06×10^1	$\mathbf{5.79 \times 10^3}$	3.66×10^3

Sensitivity to the Post-processing of the Gram Matrix. It is mentioned that the Gram matrix is post-processed by adding a scalar (default value 10^{-7}) in its diagonal elements to ensure the well-conditioning of its inversion, which potentially introduces the optimization difficulties for the inducing inputs. We studied the effect of the added scalar in such post-processing by training a 4-layer DGP on the Boston dataset. As can be seen from Fig. 3.2, the DGP based on the gradient-bridging architecture is robust to the post-processing of the Gram matrix, due to the fact that it enables more accurate optimization of the inducing inputs. Only when the added scalar is large (≈ 1) we start to observe a slight drop in the performance. In comparison, the parameter-tying architecture is more sensitive to the post-processing, but much better than the naive design because the parameter-tying architecture still captures the dependency of the inducing outputs on the corresponding inducing inputs. The naive design is very sensitive to the post-processing because such dependency is not considered, and the post-processing will directly affects the accuracy in the optimization of the inducing inputs.

Overfitting vs. Optimization Difficulties. We have hypothesized two possible reasons for the bad performance of the naive design architecture in Chapter 3.2. An interesting question to ask is whether overfitting is a big issue compared with optimization difficulties.

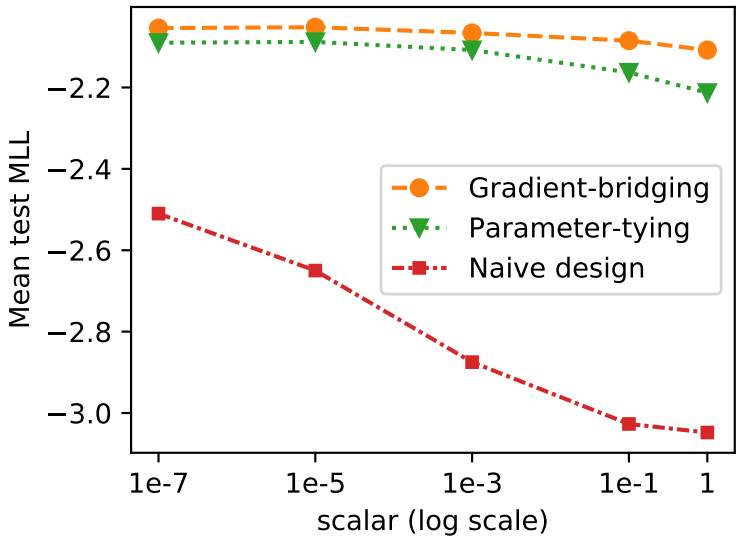

Figure 3.2: Mean test MLL achieved by our IPVI framework with three different architectures for a 4-layer DGP on Boston dataset. Different scalars (x-axis) are added to the diagonal elements of the Gram matrix $\mathbf{K_{ZZ}}$ to ensure its invertibility. Larger scalar will lead to worse gradient estimation of the inducing inputs.

To this end, we further investigate the effect of overfitting versus optimization difficulties by training different DGP models with varying number of parameters in the DGP generators on the Boston dataset. The results are reflected on Fig. 3.3.

Firstly, we can see that the train MLLs of the parameter-tying architecture (≤ -1.77) are lower than those of the gradient-bridging or the naive design architecture (> -1.76). The main reason is because the parameter-tying architecture has much fewer parameters than the other architectures, thus suffers from a certain degree of optimization difficulties during training. No obvious overfitting is observed for the parameter-tying architecture. Its test MLLs (≤ -2.08) are lower than the best test MLL (-2.05) achieved by the gradient-bridging architecture.

Secondly, the train MLLs of the naive design architecture improves consistently with the number of parameters while the test MLLs consistently degrades. It reflects that obvious overfitting has occurred. However, the DGP models with the gradient-bridging architecture have comparable numbers of parameters with that of the naive design, but overfitting is not immediately observed (only slight overfitting occurs when the number of

parameters is larger than 10^7). The difference implies that the naive design architecture might be "overfitting" the poorly optimized[5] inducing inputs rather than overfitting the training data, since the same training data are used by the gradient-bridging architecture. To verify our conjecture, we conducted another set of experiments by fixing the inducing inputs $\mathbf{Z}$ (as non-trainable parameters) for the gradient-bridging architecture. Doing so force the DGP model to "overfit" the inducing inputs without optimizing them. As can be seen Fig. 3.3, the results with fixed inducing inputs (represented by the red curves) have similar trends to the naive design architecture (blue curves), and obvious overfitting is again observed. This comparison proves our conjecture that the naive design architecture is prone to the "overfit" the inducing inputs.[6]

Consider that the commonly referred overfitting (and the overfitting we stated in Chapter 3.2) is about the dataset, "overfitting" the inducing inputs should be reckon as part of the optimization difficulties. In this regard, overfitting (the dataset) is not a big issue; and the optimization difficulties should account for the unsatisfactory performance of the naive design architecture. This observation further supports the idea in the design of our gradient-bridging architecture. Thus, in the case that the inducing inputs are well-optimized, the test performance of the overall model will be good, which is in consistent with our observation that the gradient-bridging architecture achieves the best test MLL (-2.05) out of all architectures.

Time Efficiency. Table 3.4 shows that better time efficiency of IPVI with the gradient-bridging architecture over the parameter-tying architecture for a 4-layer DGP model that is trained using the Airline dataset. The learning rates are both 0.005 for IPVI. Note that the gradient-bridging architecture has a larger number of parameters, thus incurs a larger average training time per iteration (Table 3.4). However, due to a better gradient back-propagation on the inducing inputs, our IPVI with the gradient-bridging architecture converges in much fewer iterations, thus converges faster in wall-clock time, despite

[5]Table. 3.3 and Fig. 3.2 have revealed such a fact.

[6]The DGP model with gradient-bridging architecture with fixed inducing inputs still achieves higher MLL despite having the same number of parameters, possibly due to its ability to use the (fixed) inducing inputs as the contextual information.

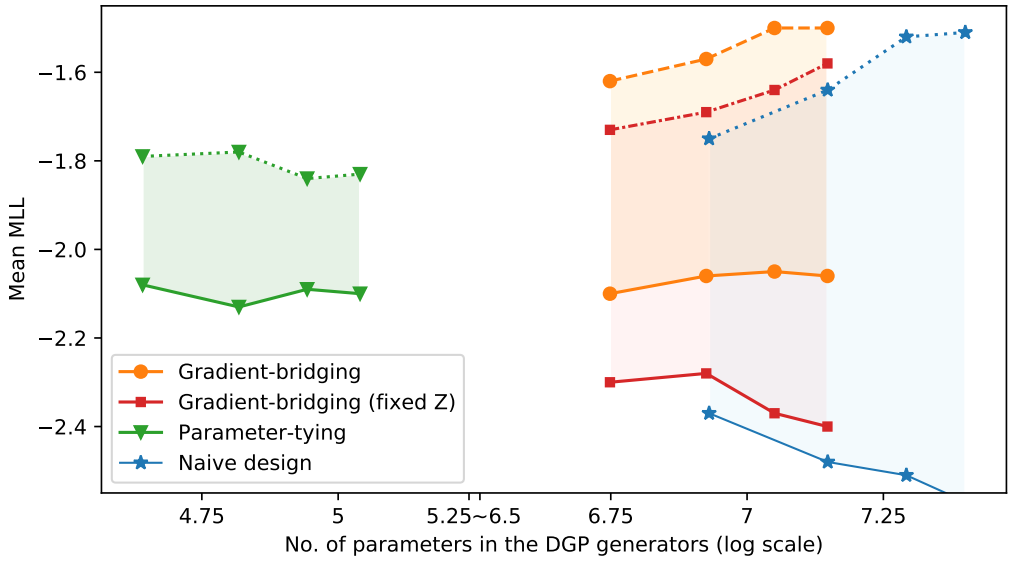

Figure 3.3: Mean train and test MLL achieved by our IPVI framework with three different architectures for a 4-layer DGP on Boston dataset. The solid lines represent the test MLL while the dotted lines represent the train MLL. The log scale is calculated with base 10.

having 128 times more parameters in the generator. Due to the parallel sampling in both architectures, our IPVI framework always enables posterior samples to be generated fast.

Table 3.4: Time incurred by a 4-layer DGP model for Airline dataset. A convergence is reckoned to be reached if the averaged test MLL over past 500 iteration is within an 0.5% range of the final averaged test MLL. As an illustration, the parameter-tying architecture has ≈ 8300 parameters in each layer's generator, while the gradient-bridging architecture has ≈ 1million parameters in each layer's generator.

	Parameter-tying	Gradient-bridging
Average training time (per iter.)	0.35 Chapter	0.40 Chapter
Average total iterations (until converge)	42500 iter.	20800 iter.
$\mathbf{U}$ generation (100 samples)	0.28 Chapter	0.33 Chapter

3.5 Conclusion

Due to the introduction of generators and discriminators into the DGP, the training becomes unstable and the test performances are unsatisfactory for our proposed IPVI framework. We identified the optimization difficulties of the inducing inputs as the main reason for the unsatisfactory test performances. We thus propose a novel gradient-bridging architecture

of the generator and discriminator in our IPVI framework for DGPs to alleviate the optimization difficulties and speed up training. Empirical evaluation shows that IPVI with the gradient-bridging architecture outperforms the state-of-the-art parameter-tying architecture due to a better gradient back-propagation on the inducing input.

In general, using a generator to output the samples of an implicit posterior is an effective way to (learn to) express the intractable posterior, given that the generator can be efficiently optimized with gradient descent. In the next chapter, we adopt the similar idea in our proposed model called *implicit processes for meta-learning* (IPML), where the intractable posterior (samples) of the model output is also implicitly expressed by a generator.

Chapter 4

Stochastic Gradient Hamiltonian Monte Carlo for Meta-Learning with Implicit Processes

4.1 Background

Few-shot learning (also known as *meta-learning*) is a defining characteristic of human intelligence. Its goal is to leverage the experiences from previous tasks to form a model (represented by meta-parameters) that can rapidly adapt to a new task using only a limited quantity of its training data. A number of meta-learning algorithms [Finn et al., 2018, Jerfel et al., 2019, Ravi and Beatson, 2018, Rusu et al., 2019, Yoon et al., 2018] have recently adopted a probabilistic perspective to characterize the uncertainty in the predictions via a Bayesian treatment of the meta-parameters. Though they can consequently represent different tasks with different values of meta-parameters, it is not clear how or whether they are naturally amenable to the following: (A) Characterization of a principled similarity/distance measure between tasks (e.g., for identifying outlier tasks that can potentially hurt training for the new task, procuring the most valuable/similar tasks/datasets to the new task, detecting task distribution shift, among others); (B) Active task selection, in which a

meta-learner wants to purchase task data from a decentralized data marketplace such as Streamr Marketplace (`https://streamr.network/marketplace`) and want to achieve best performance given a limited budget of task purchase (see also Appendix C.1.1.3 for another motivating example of a real-world use case); and (C) Synthetic task/dataset generation, in privacy-aware settings such as public healthcare data platforms like HIT Foundation [Eberhard and Paul, 2019], the data platforms can generate synthetic data to perform data anonymization to avoid revealing the sensitive real data. Synthetic task generation can also be used for augmenting a limited number of previous tasks to improve generalization performance and address task memorization [Yin et al., 2020].

To tackle the above challenge, this chapter presents a novel *implicit process-based meta-learning* (IPML) algorithm (Chapter 4.3) that, in contrast to existing works, explicitly represents each task as a continuous latent vector and models its probabilistic belief within the highly expressive IP framework. An IP [Ma et al., 2019] is a stochastic process such that every finite collection of random variables has an implicitly defined joint prior distribution. Some typical examples of IP include Gaussian processes, Bayesian neural networks, neural processes [Garnelo et al., 2018b], among others. An IP is formally defined in Def. 4.1. The IP framework adopts a generator-like network, and enjoys the advantage of being end-to-end differentiable. Compared to the Gaussian processes (GP) that require the inversion of a potentially large Gram matrix during the inference and training, IP can utilize gradient descent-based algorithm for the meta-training of its parameters without such matrix inversion, which is computationally favorable. IP is also a better choice for its expressiveness, unlike GP that constrains the distributions of the function outputs to be Gaussian. Unfortunately, unlike the GP that adopts a closed-form during inference, meta-training in IPML is somehow challenging due to its need to perform intractable exact IP inference in task adaptation. The work of [Ma et al., 2019] uses the well-studied Gaussian process as the variational family to perform variational inference in general applications of IP, which sacrifices the flexibility and expressivity of IP by constraining the distributions of the function outputs to be Gaussian. Such a

straightforward application of IP to meta-learning has not yielded satisfactory results in our experiments (see Appendix C.1.3).

To resolve this, we propose a novel *expectation-maximization* (EM) algorithm to perform meta-training (Chapter 4.3.1): In the E step, we perform task adaptation using the *stochastic gradient Hamiltonian Monte Carlo sampling* (SGHMC) method [Chen et al., 2014] to draw samples from IP posterior beliefs for all meta-training tasks. SGHMC is a sampling algorithm widely adopted in the deep learning community that can use the gradient information to improve the quality and the convergence speed of the sampling process. It is a natural choice particularly because the gradient on the latent (task) vector is easy to obtained in our end-to-end differentiable IPML framework. Doing so also eliminates the need to learn a latent encoder [Garnelo et al., 2018b]. In the M step, we optimize the meta-learning objective w.r.t. the meta-parameters using these samples through gradient descent. Our delicate design of the neural network architecture for meta-training in IPML allows competitive meta-learning performance to be achieved (Chapter 4.3.2). Our IPML algorithm offers the benefits of being amenable to (A) the characterization of a principled distance measure between tasks using maximum mean discrepancy [Gretton et al., 2012], (B) active task selection without needing the assumption of known task contexts in [Kaddour et al., 2020], and (C) synthetic task generation by modeling task-dependent input distributions (Chapter 4.3.3).

4.2 Problem Formulation

For simplicity, the inputs (outputs) for all tasks are assumed to belong to the same input (output) space. Consider meta-learning on probabilistic regression tasks (the meta-learning on probabilistic classification tasks uses the robust-max likelihood [Hernández-Lobato et al., 2011] as introduced in Chapter 3.1.1): Each task is generated from a task distribution and associated with a dataset $(\mathcal{X}, \mathbf{y}_\mathcal{X})$ where the set $\mathcal{X}$ and the vector $\mathbf{y}_\mathcal{X} \triangleq (y_\mathbf{x})^\top_{\mathbf{x}\in\mathcal{X}}$ denote,

respectively, the input vectors and the corresponding noisy outputs

$$y_{\mathbf{x}} \triangleq f(\mathbf{x}) + \epsilon(\mathbf{x}) \tag{4.1}$$

which are outputs of an unknown underlying function f corrupted by an i.i.d. Gaussian noise $\epsilon(\mathbf{x}) \sim \mathcal{N}(0, \sigma^2)$ with variance σ^2. Let f be distributed by an *implicit process* (IP), as follows:

Definition 4.1 (Implicit process for meta-learning)**.** *Let the collection of random variables $f(\cdot)$ denote an IP parameterized by meta-parameters θ, that is, every finite collection $\{f(\mathbf{x})\}_{\mathbf{x}\in\mathcal{X}}$ has a joint prior distribution $p(\mathbf{f}_{\mathcal{X}} \triangleq (f(\mathbf{x}))^{\top}_{\mathbf{x}\in\mathcal{X}})$ implicitly defined by the following generative model:*

$$\mathbf{z} \sim p(\mathbf{z}), \quad f(\mathbf{x}) \triangleq g_\theta(\mathbf{x}, \mathbf{z}) \tag{4.2}$$

for all $\mathbf{x} \in \mathcal{X}$ where $\mathbf{z}$ is a latent task vector to be explained below and generator g_θ can be an arbitrary model (e.g., deep neural network) parameterized by meta-parameters θ.

Definition 4.1 defines valid stochastic processes if $\mathbf{z}$ is finite dimensional [Ma et al., 2019]. Though, in reality, a task may follow an unknown distribution, we assume the existence of an unknown function that maps each task to a latent task vector $\mathbf{z}$ satisfying the desired known distribution $p(\mathbf{z})$, like in [Kaddour et al., 2020].[1] Using $p(\mathbf{y}_{\mathcal{X}}|\mathbf{f}_{\mathcal{X}}) = \mathcal{N}(\mathbf{f}_{\mathcal{X}}, \sigma^2\mathbf{I})$ (4.1) and the IP prior belief $p(\mathbf{f}_{\mathcal{X}})$ from Def. 4.1, we can derive the marginal likelihood $p(\mathbf{y}_{\mathcal{X}})$ by marginalizing out $\mathbf{f}_{\mathcal{X}}$.

Remark 4.1. Two sources of uncertainty exist in $p(\mathbf{y}_{\mathcal{X}})$: *Aleatoric uncertainty* in $p(\mathbf{y}_{\mathcal{X}}|\mathbf{f}_{\mathcal{X}})$ reflects the noise (i.e., modeled in (4.1)) inherent in the dataset, while *epistemic uncertainty* in the IP prior belief $p(\mathbf{f}_{\mathcal{X}})$ reflects the model uncertainty arising from the *latent task prior belief* $p(\mathbf{z})$ in (4.2). Note that our work here considers a point estimate of meta-parameters

[1] $p(\mathbf{z})$ is often assumed to be a simple distribution like multivariate Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$ [Garnelo et al., 2018b].

θ instead of a Bayesian treatment of θ [Finn et al., 2018, Yoon et al., 2018]. This allows us to interpret the epistemic uncertainty in $p(\mathbf{f}_{\mathcal{X}})$ via $p(\mathbf{z})$ directly.

Let the sets $\mathcal{T}$ and $\mathcal{T}_*$ denote the meta-training and meta-testing tasks, respectively. Following the convention in [Finn et al., 2018, Gordon et al., 2019, Ravi and Beatson, 2018, Yoon et al., 2018], for each meta-training task $t \in \mathcal{T}$, we consider a support-query (or train-test) split of its dataset $(\mathcal{X}_t, \mathbf{y}_{\mathcal{X}_t})$ into the *support set* (or training dataset) $(\mathcal{X}_t^s, \mathbf{y}_{\mathcal{X}_t^s})$ and *query set* (or test/evaluation dataset) $(\mathcal{X}_t^q, \mathbf{y}_{\mathcal{X}_t^q})$ where $\mathcal{X}_t = \mathcal{X}_t^s \cup \mathcal{X}_t^q$ and $\mathcal{X}_t^s \cap \mathcal{X}_t^q = \emptyset$. Specifically, for a N-way K-shot classification problem, the support set has K examples per class and N classes in total.

Meta-learning can be defined as an optimization problem [Finn et al., 2017, Finn et al., 2018] and its goal is to learn meta-parameters θ that maximize the following objective defined over all meta-training tasks:

$$\mathcal{J}_{\text{meta}} \triangleq \log \prod_{t \in \mathcal{T}} p\Big(\mathbf{y}_{\mathcal{X}_t^q} | \mathbf{y}_{\mathcal{X}_t^s}\Big) = \sum_{t \in \mathcal{T}} \log \int p\Big(\mathbf{y}_{\mathcal{X}_t^q} | \mathbf{f}_{\mathcal{X}_t^q}\Big)\, p\Big(\mathbf{f}_{\mathcal{X}_t^q} | \mathbf{y}_{\mathcal{X}_t^s}\Big)\, \mathrm{d}\mathbf{f}_{\mathcal{X}_t^q} \,. \tag{4.3}$$

The optimization of (4.3) with respect to θ is reckon as the *outer* loop calculation in meta-learning. In every iteration of outer loop, there is an *inner* loop calculation involving task *adaptation*, i.e., calculation of the $p(\mathbf{f}_{\mathcal{X}_t^q} | \mathbf{y}_{\mathcal{X}_t^s})$. Task *adaptation* $p(\mathbf{f}_{\mathcal{X}_t^q} | \mathbf{y}_{\mathcal{X}_t^s})$ is performed via IP inference after observing the support set:

$$p\Big(\mathbf{f}_{\mathcal{X}_t^q} | \mathbf{y}_{\mathcal{X}_t^s}\Big) = \int_{\mathbf{z}} p\Big(\mathbf{f}_{\mathcal{X}_t^q} | \mathbf{z}\Big)\, p\big(\mathbf{z} | \mathbf{y}_{\mathcal{X}_t^s}\big)\, \mathrm{d}\mathbf{z} \,. \tag{4.4}$$

The objective $\mathcal{J}_{\text{meta}}$ (4.3) is the "test" likelihood on the query set, which reflects the idea of "learning to learn" by assessing the effectiveness of "learning on the support set" through the query set. An alternative interpretation views $p(\mathbf{f}_{\mathcal{X}_t^q} | \mathbf{y}_{\mathcal{X}_t^s})$ as an "informative prior" after observing the support set. The objective $\mathcal{J}_{\text{meta}}$ (4.3) is also known as the Bayesian held-out likelihood [Gordon et al., 2019]. In a meta-testing task, adaptation is also performed via IP inference after observing its support set and evaluated on its query

set. Similar to GP or any stochastic process, the input vectors of the dataset are assumed to be known/fixed beforehand (thus we neglect all the conditionals on $\mathcal{X}_t^s$, $\mathcal{X}_t^q$ or $\mathcal{X}_t$). We will relax this assumption by allowing them to be unknown when our IPML algorithm is exploited for synthetic task generation (Chapter 4.3.3).

4.3 Implicit Process-based Meta-Learning (IPML)

4.3.1 Expectation Maximization (EM) Algorithm for IPML

Recall that task adaptation (inner loop) requires evaluating $p(\mathbf{f}_{\mathcal{X}_t^q}|\mathbf{y}_{\mathcal{X}_t^s})$ (4.4). From Def. 4.1, if generator g_θ (4.2) can be an arbitrary model (e.g., deep neural network), then $p(\mathbf{f}_{\mathcal{X}_t^q}|\mathbf{y}_{\mathcal{X}_t^s})$ and $p(\mathbf{f}_{\mathcal{X}_t^q})$ cannot be evaluated in closed form and have to be approximated. In our case, the approximation is done by sampling. Inspired by the Monte Carlo EM algorithm [Wei and Tanner, 1990] which utilizes posterior samples to obtain a maximum likelihood estimate of some hyperparameters, we propose an EM algorithm for IPML: The E step uses the *stochastic gradient Hamiltonian Monte Carlo* (SGHMC) sampling method to draw samples from $p(\mathbf{f}_{\mathcal{X}_t^q}|\mathbf{y}_{\mathcal{X}_t^s})$ (4.4), while the M step maximizes the meta-learning objective $\mathcal{J}_{\text{meta}}$ (4.3) w.r.t. meta-parameters θ:

Expectation (E) step. This step corresponds to the inner loop calculation. Note that since $\mathbf{f}_{\mathcal{X}_t^q} = (g_\theta(\mathbf{x}, \mathbf{z}))_{\mathbf{x}\in\mathcal{X}_t^q}^\top$ (4.2), no uncertainty exists in $p(\mathbf{f}_{\mathcal{X}_t^q}|\mathbf{z})$ in (4.4). So, $p(\mathbf{f}_{\mathcal{X}_t^q}|\mathbf{y}_{\mathcal{X}_t^s})$ can be evaluated using the same generator g_θ (4.2) and the *latent task posterior belief* $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$, as follows:

Remark 4.2. Drawing samples from $p(\mathbf{f}_{\mathcal{X}_t^q}|\mathbf{y}_{\mathcal{X}_t^s})$ is thus equivalent to first drawing samples of $\mathbf{z}$ from $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$ and then passing them as inputs to generator g_θ to obtain samples of $\mathbf{f}_{\mathcal{X}_t^q}$. Hence, given a task t, adaptation $p(\mathbf{f}_{\mathcal{X}_t^q}|\mathbf{y}_{\mathcal{X}_t^s})$ (4.4) essentially reduces to a task identification problem by performing IP inference to obtain the latent task posterior belief $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$. This is a direct consequence of epistemic uncertainty arising from $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$ and $p(\mathbf{z})$ (Remark 4.1).

In general, $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$ also cannot be evaluated in closed form. Instead of using *variational inference* (VI) and approximating $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$ with a potentially restrictive variational distribution [Garnelo et al., 2018b, Kaddour et al., 2020, Ma et al., 2019], we draw samples from $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$ using SGHMC [Chen et al., 2014]. SGHMC introduces an auxiliary random vector $\mathbf{r}$ and samples from a joint distribution $p(\mathbf{z}, \mathbf{r}|\mathbf{y}_{\mathcal{X}_t^s})$ following the Hamiltonian dynamics [Brooks et al., 2011, Neal, 1993]: $p(\mathbf{z}, \mathbf{r}|\mathbf{y}_{\mathcal{X}_t^s}) \propto \exp(-U(\mathbf{z}) - 0.5\mathbf{r}^\top\mathbf{M}^{-1}\mathbf{r})$ where the negative log-probability $U(\mathbf{z}) \triangleq -\log p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$ resembles the potential energy and $\mathbf{r}$ resembles the momentum. SGHMC updates $\mathbf{z}$ and $\mathbf{r}$, as follows:

$$\Delta\mathbf{z} = \alpha\mathbf{M}^{-1}\mathbf{r}, \quad \Delta\mathbf{r} = -\alpha\nabla_{\mathbf{z}}U(\mathbf{z}) - \alpha\mathbf{C}\mathbf{M}^{-1}\mathbf{r} + \mathcal{N}(\mathbf{0}, 2\alpha(\mathbf{C} - \mathbf{B}))$$

where α, $\mathbf{C}$, $\mathbf{M}$, and $\mathbf{B}$ are the step size, friction term, mass matrix, and Fisher information matrix, respectively. Note that $\nabla_{\mathbf{z}}U(\mathbf{z}) = -\nabla_{\mathbf{z}}\log p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s}) = -\nabla_{\mathbf{z}}\log p(\mathbf{z}, \mathbf{y}_{\mathcal{X}_t^s}) = -\nabla_{\mathbf{z}}[\log p(\mathbf{y}_{\mathcal{X}_t^s}|\mathbf{f}_{\mathcal{X}_t^s} = (g_\theta(\mathbf{x}, \mathbf{z}))_{\mathbf{x}\in\mathcal{X}_t^s}^\top) + \log p(\mathbf{z})]$ can be evaluated tractably. The last term is a normally distributed noise term that can reduce the effect of inaccurate gradient estimation due to the random minibatch sampling. The sampler hyperparameters α, $\mathbf{C}$, $\mathbf{M}$, and $\mathbf{B}$ are set according to the auto-tuning method of [Springenberg et al., 2016] which has been verified to work well in our experiments; more details are given in Appendix C.1.1.1.

Maximization (M) step. This step corresponds to the outer loop calculation. We optimize $\mathcal{J}_{\text{meta}}$ (4.3) w.r.t. θ using samples of $\mathbf{z}$. The original objective $\mathcal{J}_{\text{meta}} = \sum_{t\in\mathcal{T}}\log(\mathbb{E}_{p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})}[p(\mathbf{y}_{\mathcal{X}_t^q}|\mathbf{f}_{\mathcal{X}_t^q} = (g_\theta(\mathbf{x}, \mathbf{z}))_{\mathbf{x}\in\mathcal{X}_t^q}^\top)])$ is not amenable to stochastic optimization with data minibatches, which is usually not an issue in a few-shot learning setting. When a huge number of data points and samples of $\mathbf{z}$ are considered, we can resort to optimizing the lower bound $\mathcal{J}_{\text{s-meta}}$ of $\mathcal{J}_{\text{meta}}$ by applying the Jensen's inequality:

$$\mathcal{J}_{\text{meta}} \geq \mathcal{J}_{\text{s-meta}} \triangleq \sum_{t\in\mathcal{T}}\mathbb{E}_{p(\mathbf{f}_{\mathcal{X}_t^q}|\mathbf{y}_{\mathcal{X}_t^s})}\Big[\log p(\mathbf{y}_{\mathcal{X}_t^q}|\mathbf{f}_{\mathcal{X}_t^q})\Big] = \sum_{t\in\mathcal{T}}\mathbb{E}_{p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})}\Big[\log p(\mathbf{y}_{\mathcal{X}_t^q}|\mathbf{f}_{\mathcal{X}_t^q})\Big].$$

4.3.2 Architecture Design for Meta-Training

Our generator g_θ is implemented using a *deep neural network* (DNN) parameterized by meta-parameters θ. Under this setup, we have empirically observed that the design of the coupling of $\mathbf{z}$ with the DNN $g_\theta(\mathbf{x}, \cdot)$ is crucial to achieving competitive performance of our IPML algorithm. A naive design by concatenating $\mathbf{z}$ with $\mathbf{x}$ (or higher-level abstractions of $\mathbf{x}$) as a contextual input during forward passes has not worked well as the resulting gradients w.r.t. $\mathbf{z}$ may not have provided enough guidance for SGHMC to learn a sufficiently useful representation of $\mathbf{z}$ in meta-training.

To this end, inspired by the *attention* mechanism [Vaswani et al., 2017] and *dropout* method [Srivastava et al., 2014], we introduce a design of the coupling by applying $\mathbf{z}$ as a *mask* to the last DNN layer's parameters: The last DNN layer's parameters are first masked by $\mathbf{z}$ (i.e., point-wise product with $\mathbf{z}$), as illustrated in Figs. 4.1a and 4.1b. Different tasks can now be distinguished by different masks, hence resembling different attentions on the last DNN layer's connections during forward propagation. We adopt soft masks (i.e., continuous values) instead of hard masks (i.e., either 0 or 1). The latent task prior belief $p(\mathbf{z})$ is thus assumed to be a multivariate Gaussian $\mathcal{N}(\mathbf{1}, \mathbf{I})$.

In our design, $\mathbf{z}$ serves as a "feature selector" (masking out the undesired signals) rather than a contextual input in which $\mathbf{z}$ is naively concatenated with $\mathbf{x}$ as the input to the model g. This is because in the beginning of the training, our estimation of $p(\mathbf{z})$ is inaccurate, and using it as a contextual information will lead to some degree of optimization difficulties. Instead, using $\mathbf{z}$ as a feature selector will allow our estimation of $p(\mathbf{z})$ to be optimized rapidly, and thus quickly improves our estimation of the meta-parameters. Such a mask design of the coupling is empirically demonstrated to be effective in our experiments (Appendix C.1.3.3).

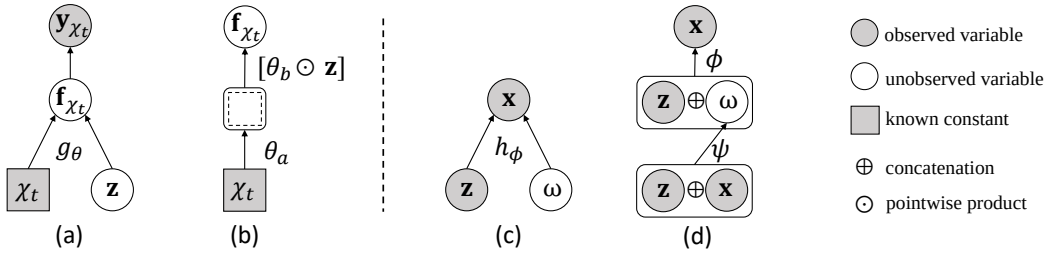

Figure 4.1: (a) Graphical model corresponding to IPML. (The usage of support and query sets are distinguished in the objective $\mathcal{J}_{\text{meta}}$ (4.3).) (b) DNN implementation of generator g_θ where $\theta \triangleq (\theta_a, \theta_b)$ and θ_a can be convolutions to obtain high-level representations of the input vector, while θ_b is the last DNN layer's parameters which are masked by $\mathbf{z}$ during the forward passes. (c) Graphical model corresponding to input generation by X-Net. (d) CVAE implementation of X-Net (i.e., decoder neural network with parameters ϕ).

4.3.3 Architecture Design for Synthetic Task Generation

Recall the assumption of known/fixed input vectors in $\mathcal{X}_t$ in the last paragraph of Chapter 4.2,[2] which we will have to relax here. Synthetic task generation can be performed by the following procedure if $\mathbf{x}$ is task-independent (e.g., $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x})p(\mathbf{z})$): After meta-training is completed (Chapter 4.2), draw a sample of latent task vector $\mathbf{z} \sim p(\mathbf{z})$, draw samples of $\mathbf{x} \sim p(\mathbf{x})$ to form $\mathcal{X}_t$, and then generate noisy outputs $\mathbf{y}_{\mathcal{X}_t} = (g_\theta(\mathbf{x}, \mathbf{z}) + \epsilon(\mathbf{x}))^\top_{\mathbf{x} \in \mathcal{X}_t}$ to obtain the dataset $(\mathcal{X}_t, \mathbf{y}_{\mathcal{X}_t})$ for task t.

When $\mathbf{x}$ is task-dependent (e.g., for image classifications of different objects, $p(\mathbf{x}, \mathbf{z}) \neq p(\mathbf{x})p(\mathbf{z})$), not modeling $p(\mathbf{x}|\mathbf{z})$ limits the ability to generate t-dependent $\mathcal{X}_t$. To resolve this, our IPML algorithm includes an *X-generative network* (X-Net): $\mathbf{x} \triangleq h_\phi(\mathbf{z}, \boldsymbol{\omega})$ that learns to generate an input vector $\mathbf{x}$ given samples of the latent task vector $\mathbf{z}$ and random vector $\boldsymbol{\omega} \sim p(\boldsymbol{\omega}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ where $\boldsymbol{\omega}$ models the diversity of the input distribution given a fixed task represented by the sample of $\mathbf{z}$. There are several options to implement X-Net: Note that during the training of X-Net, both $\mathcal{X}_t$ and the samples of $\mathbf{z} \sim p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$ for all meta-training task $t \in \mathcal{T}$ are available. So, generative models such as the *conditional variational autoencoder* (CVAE) [Sohn et al., 2015] or conditional generative adversarial networks [Mirza and Osindero, 2014] are suitable for X-Net as they can utilize $\mathbf{z}$ as the *contextual* information. Our work here uses (the decoder of) CVAE to implement X-Net.

[2]This assumption is reasonable for meta-training since only $p(\mathbf{y}_\mathcal{X})$ (and not $p(\mathbf{x})$) needs to be modeled.

Figs. 4.1c and 4.1d illustrate such a design. We have empirically observed that a simple concatenation with $\mathbf{z}$ suffices here as our delicate architecture design for meta-training (Chapter 4.3.2) can yield a useful representation of $\mathbf{z}$ for training X-Net well. Further details and a method to ensure balanced data generation are given in Appendix C.1.4. The training objective for synthetic task generation is the empirical lower bound [Sohn et al., 2015] of VI on $p(\boldsymbol{\omega}|\mathbf{x}, \mathbf{z})$:

$$\mathcal{J}_{\mathrm{X}} \triangleq \sum_{t\in\mathcal{T}} \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})} \Big[|\mathcal{X}_t|^{-1} \sum_{\mathbf{x}\in\mathcal{X}_t} \Big(\mathbb{E}_{q_\psi(\boldsymbol{\omega}|\mathbf{x},\mathbf{z})}[\log p_\phi(\mathbf{x}|\mathbf{z},\boldsymbol{\omega})] - D_{\mathrm{KL}}[q_\psi(\boldsymbol{\omega}|\mathbf{x},\mathbf{z})\|p(\boldsymbol{\omega})] \Big) \Big]$$

where ϕ and ψ are, respectively, the parameters of X-Net (decoder neural network) and the encoder neural network, and D_{KL} denotes the KL distance. In the training of X-Net, we sample one $\mathbf{z}$ per update. We also sample one $\boldsymbol{\omega}$ per update to train with reparameterization tricks. Algorithms 2 and 3 describe meta-training (with training of X-Net) and synthetic task generation, respectively. In the case that synthetic task generation is not needed, we can skip the training of X-Net (* in Algorithm 2).

Algorithm 2 IPML: Meta-Training

while not converged **do**

Sample task t from $\mathcal{T}$

`E step` : Sample $\{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ with SGHMC (Chapter 4.3.1)

`M step` : Sample $\mathbf{z}$ from $\{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$

$$\theta \leftarrow \theta + \eta \nabla_\theta \mathcal{J}_{\text{meta}}$$

*Update X-Net (Chapter 4.3.3) with $\mathbf{z}$ and $\mathcal{X}_t$:

$$\phi \leftarrow \phi + \eta \nabla_\phi \mathcal{J}_{\mathrm{X}} , \quad \psi \leftarrow \psi + \eta \nabla_\psi \mathcal{J}_{\mathrm{X}}$$

end while

return θ, ϕ, ψ

Algorithm 3 Synthetic Task Generation

Sample $\mathbf{z} \sim p(\mathbf{z})$

Initialize synthetic task t and $\mathcal{X}_t = \emptyset$

for $i = 1, \ldots,$ *final size of* $\mathcal{X}_t$ **do**

Sample $\boldsymbol{\omega} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Compute $\mathbf{x} = h_\phi(\mathbf{z}, \boldsymbol{\omega})$

Compute $y_{\mathbf{x}} = g_\theta(\mathbf{x}, \mathbf{z}) + \epsilon(\mathbf{x})$

$(\mathcal{X}_t, \mathbf{y}_{\mathcal{X}_t}) \leftarrow (\mathcal{X}_t \cup \{\mathbf{x}\}, \mathbf{y}_{\mathcal{X}_t \cup \{\mathbf{x}\}})$

end for

return $(\mathcal{X}_t, \mathbf{y}_{\mathcal{X}_t})$ for task t

4.4 Experiments and Discussion

Benchmark datasets: sinusoid regression and few-shot image classification. We first empirically evaluate the performance of our IPML algorithm against that of several Bayesian meta-learning baselines like the *neural process* (NP) [Garnelo et al., 2018b], *Bayesian model-agnostic meta-learning* (BMAML) [Yoon et al., 2018], PLATIPUS [Finn et al., 2018], and *amortized Bayesian meta-learning* (ABML) [Ravi and Beatson, 2018] on benchmark meta-learning datasets. For few-shot image classification, we also empirically compare IPML with a strong baseline: *prototypical network* (PN) [Snell et al., 2017]. We run experiments on three datasets: sinusoid, Omniglot [Lake et al., 2011], and mini-ImageNet [Ravi and Larochelle, 2017]. Sinusoid is a regression task of sine waves with uniformly sampled amplitude in $[0.1, 5.0]$, phase in $[0, \pi]$, and input $\mathbf{x}$ in $[-5, 5]$. The generator of IPML and the baseline regressors are neural networks with 2 hidden layers of size 40 with ReLU nonlinearities. The Omniglot dataset consists of 20 instances of 1623 characters from 50 different alphabets. The mini-ImageNet dataset involves 64 training classes, 12 validation classes, and 24 test classes. For Omniglot and mini-ImageNet, our implementation and baselines all use the same data pre-processing, same train-test

split, and same data augmentation as that in [Finn et al., 2017]. The generator of IPML and the baseline classifiers are convolutional neural networks with 4 modules of 3×3 convolutions and 64 filters, followed by batch normalization, ReLU nonlinearities, and strided convolutions (Omniglot) or 2×2 max-pooling (mini-ImageNet). More details of the experimental settings can be found in Appendix C.1.1.2.

For sinusoid regression (Table 4.1), IPML outperforms MAML and BMAML by a fair margin. For Omniglot (Table 4.2), IPML is competitive with MAML and PN. For mini-ImageNet (Table 4.3), IPML outperforms MAML and all tested Bayesian meta-learning algorithms,[3] while being competitive with PN. PN achieves a higher classification accuracy for 1-shot 20-way Omniglot and 5-shot 5-way mini-ImageNet because PN utilizes more information from the extra classes during training [Snell et al., 2017]. Specifically, though meta-testing involves N-way classification for all tested algorithms, the training of PN requires more than N classes, that is, 60-way classification which is also the setting adopted in [Snell et al., 2017]. As a result, since PN utilizes more information from the extra classes during training, it is reasonable to expect that PN achieves a higher classification accuracy at times.[4] Overall, IPML is effective for benchmark datasets.

For both sinusoid regression (Table 4.1) and Omniglot (Table 4.2), NP performs unsatisfactorily as compared to IPML, likely because (a) it performs amortized variational inference of $\mathbf{z}$ through a heavily parameterized encoder which may introduce optimization difficulties and overfitting during meta-training, and (b) the encoder of NP takes in the simple concatenation of $(\mathbf{x}, y_{\mathbf{x}})$ and thus does not explicitly capture the $\mathbf{x} \rightarrow y_{\mathbf{x}}$ relationship in the support set. An ablation study of the limitations of NP can be found in Appendix C.1.7.

Active task selection. We can evaluate the effectiveness of the uncertainty measure arising from latent task posterior belief $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$ by performing active task selection. Unlike

[3]Some of the results are taken from [Finn et al., 2018, Nguyen et al., 2020a, Yoon et al., 2018]. The 5-shot 5-way results for PLATIPUS and ABML are missing because there are no publicly available implementations.

[4]Although it seems that PN utilizing extra classes during training might be an unfair comparison, it can still be viewed as a performance "upper-bound" when tested on the same N-way K-shot setting.

Table 4.1: *Mean square error* (MSE) on few-shot sinusoid regression.

	Sinusoid 5-shot	Sinusoid 10-shot
NP	0.460	0.264
MAML	0.712	0.287
BMAML	0.409	0.200
IPML(Ours)	**0.373**	**0.123**

Table 4.2: Few-shot classification accuracy (%) on held-out Omniglot characters.

	Omniglot 1-shot 5-way	Omniglot 1-shot 20-way
NP	95.9	55.3
MAML	98.7	92.5
PN	**98.8**	**96.0**
IPML(Ours)	**98.8**	94.0

Table 4.3: Few-shot classification accuracy (%) on mini-Imagenet test set.

	mini-ImageNet 1-shot 5-way	mini-ImageNet 5-shot 5-way
MAML	48.6	65.9
PN	49.4	**68.2**
PLATIPUS	50.1	-
BMAML	49.1	64.2
ABML	45.0	-
IPML(Ours)	**50.5**	67.6

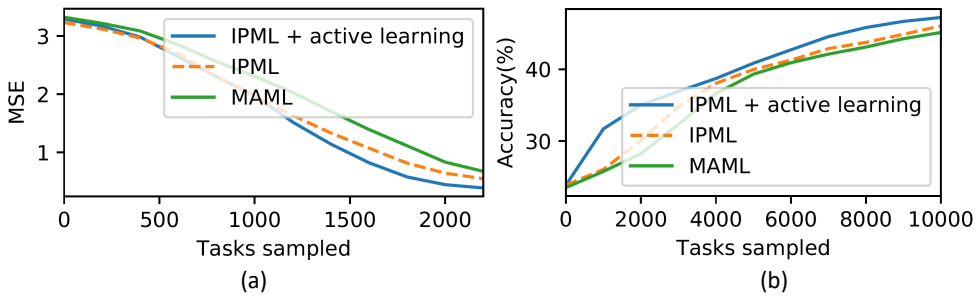

Figure 4.2: Results of active task selection on (a) 5-shot sinusoid and (b) 1-shot 5-way mini-ImageNet.

previous works [Yoon et al., 2018, Finn et al., 2018] that can only perform active learning by querying data points, IPML can perform active learning by querying *tasks* and does not need the assumption of known task contexts in [Kaddour et al., 2020]. In every iteration, a set of tasks are proposed with only the support set $(\mathcal{X}_t^s, \mathbf{y}_{\mathcal{X}_t^s})$ given; in image classification, it is usually one-shot. IPML will select among them the task with the maximum variance in $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$ (with samples from the E step/SGHMC): $\arg\max_t \text{Var}(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$, and request for its query set to perform meta-training. This corresponds to a variance-based active task selection criterion. We test on both sinusoid regression and mini-ImageNet classification. Fig. 4.2 shows that the performance of IPML with active task selection improves over that of both MAML or IPML without active task selection, that is, it reaches a given MSE/accuracy with less training tasks. This shows that the uncertainty measure arising from $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$ can be exploited to benefit meta-training.

Measuring distance between tasks using latent task representation. A most interesting question yet to be answered is the following: Does IPML learn a useful latent task representation? IPML learns to model the task through $\mathbf{z}$. If IPML learns the correct representation, then it can reflect patterns of task distribution in the latent space. While a solid criterion for assessing the correctness of learned latent task representation is hard to define, we can resort to an oracle (e.g., human expert with prior knowledge in designing the tasks). Our visualization of the latent task representation and quantitative evaluation of distance measure between tasks using *maximum mean discrepancy* (MMD) [Gretton et al., 2012] provide ways to assess the correctness of the learned task representation. We denote the set of samples from $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$ as $\mathcal{Z}_t$. The MMD between tasks t_1 and t_2 can be calculated using

$$\text{MMD}[\mathcal{H}, t_1, t_2] \triangleq \sup_{\varkappa \in \mathcal{H}} \left(|\mathcal{Z}_{t_1}|^{-1} \sum_{\mathbf{z} \in \mathcal{Z}_{t_1}} \varkappa(\mathbf{z}) - |\mathcal{Z}_{t_2}|^{-1} \sum_{\mathbf{z} \in \mathcal{Z}_{t_2}} \varkappa(\mathbf{z}) \right)$$

where $\mathcal{H}$ is a unit ball in the reproducing kernel Hilbert space with a radial basis function kernel. It is worth mentioning that, the reason we don't use a Euclidean measure on the

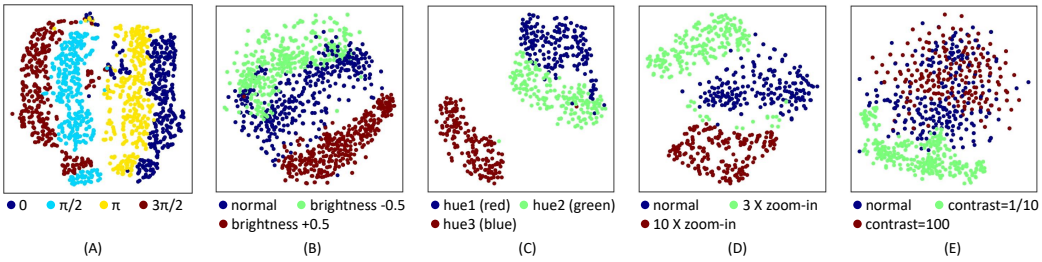

Figure 4.3: Visualization of latent task embeddings from settings A to E.

mean values of two sets of samples is because the latent task posterior belief is usually not single-modal, as can be observed from Fig. 4.3. Thus a point estimation of the latent task vector could be inaccurate.

We conduct experiments with the following 5-way 1-shot settings. **Setting A**: For subsampled Omniglot, we applied one rotation out of 4 possibilities $(0, \pi/2, \pi, 3\pi/2)$ uniformly across all the input images for each sampled task.[5] **Setting B**: For subsampled mini-ImageNet, a random artistic filter (normal, brighten, or darken) is applied for each sampled task. **Setting C**: For subsampled mini-ImageNet, a random artistic filter (3 different types of hue) is applied for each sampled task. **Setting D**: For subsampled mini-ImageNet, a random zooming (no zooming, zooming 3 times, or zooming 10 times) is applied for each sampled task. **Setting E**: On subsampled mini-ImageNet, a random artistic filter (normal, low contrast, or high contrast) is applied for each sampled task. **Setting A** has 4 types of tasks while **settings B to E** result in 3 types of tasks.

For each setting mentioned above, we first train our models in IPML to converge, and then sample tasks from their latent task posterior beliefs (i.e., one sample of $\mathbf{z}$ per task). Finally, we visualize their latent task embeddings in the 2D space using TSNE [van der Maaten and Hinton, 2008]. Furthermore, for setting A, we evaluate the distance measure between tasks using the well-known MMD metric with radial basis function kernels on the $\mathbf{z}$ samples. It can be observed from Fig. 4.3 and Table 4.4 that IPML successfully distinguishes 4 types of rotations for Omniglot. Both Fig. 4.3 and Table 4.4 contemporaneously show that flipping upside down (i.e., either right half of the

[5]In the previous experiment, the Omniglot dataset is augmented with rotations, but is random across the classes in a single task.

Table 4.4: Values of MMD metric between 4 types of tasks for Omniglot (setting A). Larger value means larger dissimilarity.

Rotations	0	$\pi/2$	π	$3\pi/2$
0	0	1.166	0.594	1.134
$\pi/2$	1.166	0	0.913	0.596
π	0.594	0.913	0	0.917
$3\pi/2$	1.134	0.596	0.917	0

Table 4.5: Results of meta-testing for training with real and generated tasks.

Train on	Accuracy (%)
real	73.83
generated	78.33
real + generated	88.16

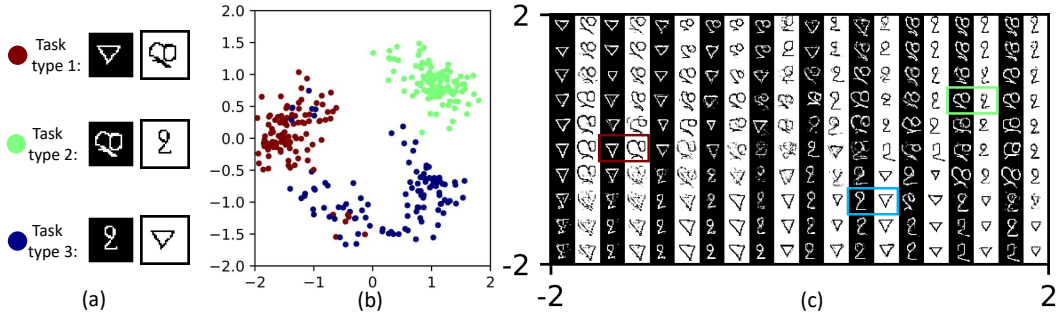

Figure 4.4: (a) TSNE visualization of (samples of) 3 types of binary classification tasks; images of black/white background are black/white samples ($y_{\mathbf{x}} = 1/y_{\mathbf{x}} = 0$). (b) Visualization of latent embedding of real tasks in (normalized) $\mathbf{z}$ space $[-2, 2]^2$. (c) Sampled generated task data by walking through the (normalized) $\mathbf{z}$ space $[-2, 2]^2$; note that the inversion of color is only for visualization to distinguish black and white samples. In training, NO images are inverted.

embedding $0 \rightleftarrows \pi$ or left half of the embedding $\pi/2 \rightleftarrows 3\pi/2$) are closer tasks compared with rotation of $\pi/2$, thus revealing that our visualization and evaluation of distance measure between tasks are in accordance. From Fig. 4.3B to Fig. 4.3D, IPML successfully distinguishes different types of transformations on the tasks while revealing interesting facts: for example, tasks of high brightness are more isolated from that of low or normal brightness. Fig. 4.3E shows that tasks of low contrast are more distinct from that of normal or high contrast. The values of MMD metric for settings B to E and comparison of the visualizations of other models are provided in Appendix C.1.5. On the overall, both the visualization and evaluation of distance measure between tasks reveal that IPML successfully learns useful latent task representations and even provides interesting insights.

Synthetic task generation for Omniglot. We assess the usefulness of latent task representation $\mathbf{z}$ by performing synthetic task generation. The training tasks we consider are three types of sub-sampled binary classifications: classification of characters A vs. B, B vs. C, and C vs. A, as in Fig. 4.4a. During meta-learning, we train a X-Net concurrently to learn to generate task-related input images (Chapter 4.3.3). The CVAE implementation of X-Net contains a decoder neural network with 3 hidden layers of size $[128, 128, 256]$ and ReLU nonlinearity, and a symmetric design of the encoder. After meta-training is completed, we continue to train the X-Net to converge. In this experiment, the dimension of $\mathbf{z}$ is set as 2, which further allows walking through such a latent space/embedding to visualize how the generated tasks map to their latent representations. Fig. 4.4b shows the latent embedding of real tasks. Fig. 4.4c shows the sampled synthetic tasks by walking through the latent space. It can be observed that X-Net successfully captures the task-dependent input distributions and can generate high-quality data of task type 1, 2, and 3 when sampled from their corresponding latent clusters (see samples of task type 1, 2, and 3 in the colored bounding boxes in Fig. 4.4c).

We further evaluate the quality of generated tasks by training on it. We hold out half of the images for each character during meta-training to construct the meta-testing tasks. The results are presented in Table 4.5. When training on both real and generated tasks, we first train on the generated tasks to converge and then train on the real tasks for another 30 iterations. It can be observed that compared to only using real tasks, a higher accuracy is achieved with training merely using generated tasks. When training on both real and generated tasks, a huge boost in accuracy is observed. We conjecture that due to their diversity, generated tasks (i.e., sometimes containing more ambiguous tasks) alleviate overfitting and provide a promising direction on meta-task augmentation.

Real-world risk detection. We perform experiments on a real-world risk detection dataset provided by an anonymous e-commerce company. The task is to classify whether an item in the online shop has risks (e.g., fraud, pornography, contraband). Such risks appear in different forms and in different categories (of items). It is hard to detect risks

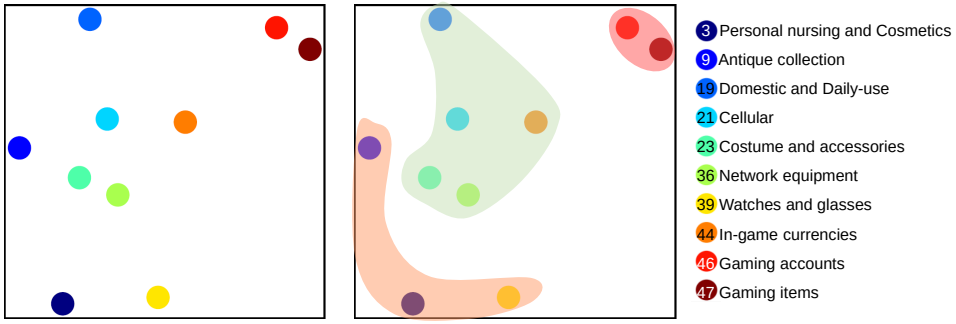

Figure 4.5: (a) TSNE visualization of latent task embedding of 10 meta-testing categories and (b) their analysis (see main text). Legend shows IDs and names of categories.

Table 4.6: Averaged meta-testing performance on 10 meta-testing categories.

	Accuracy (%)	F1
IPML	**84.5**	**70.5**
Multi-task	84.1	60.5

Table 4.7: Averaged meta-testing performance on 5 desired categories (IDs 19, 21, 23, 36, 44).

	Accuracy (%)	F1
Setting A	**87.4**	**75.8**
Setting B	86.4	74.4

in different categories by training models separately for each category because some categories have only very limited amounts of black samples (i.e., < 50). The similarities of the detected risks in different categories, if discovered, can help improve the performance. Meta-learning is thus a suitable algorithm for its ability to perform (a) detection of risks across different categories of items and (b) adaptation to new categories. The input $\mathbf{x}$ of the dataset is the text (title and descriptions) embedding obtained from self-supervised learning, while its label is a binary variable indicating whether it contains risks (i.e., $y_{\mathbf{x}} = 1$ for black samples and $y_{\mathbf{x}} = 0$ for white samples). The data are separated by categories of items to yield 47 categories in total. Initially, we hold out 10 categories for meta-testing (their category names and IDs are given in Fig. 4.5), while the rest are used for meta-training.

Table 4.6 shows results comparing the performance of IPML vs. a multi-task learning baseline. Multi-task learning based on hard parameter sharing [Caruana, 1997] is the

company's best performing baseline in this problem before the usage of meta-learning. This baseline comparison is to justify the use of meta-learning in this real-world setting. When testing on an unseen category, multi-task learning performs adaptation by randomly initializing its untied parameters for retraining on the few-shot support data. It can be observed that IPML outperforms multi-task learning, which indicates its stronger ability to generalize to unseen categories. Fig. 4.5 visualizes the latent task embedding of the 10 meta-testing categories for analysis. IPML learns useful latent task representations: For example, from Fig. 4.5a, gaming-related categories with IDs 46 and 47 are mapped closely in the latent task space/embedding.

The individual meta-testing performance on the 10 meta-testing categories, which are given in Appendix C.1.2, can be further examined: For the five categories with IDs 19, 21, 23, 36, and 44 covered by the shaded light green zone in Fig. 4.5b, IPML outperforms multi-task learning by a large margin. They are mapped to the center of the latent task space (Fig. 4.5b), which may imply that IPML's adaptations to them can largely build on previous experiences of the meta-training categories and IPML's exploitation of such similarities allows their performance to improve over multi-task learning. For the three categories with IDs 3, 9, and 39 covered by the shaded light orange zone, IPML does not have a performance advantage over multi-task learning. For the two categories with IDs 46 and 47 covered by the shaded light pink zone, both IPML and multi-task learning perform unsatisfactorily. As a matter of fact, for IPML, the categories with unsatisfactory performance (i.e., either covered by the shaded light orange or pink zone) are all mapped to be some distance away from the center, which indicates that they are likely considered by IPML as "outlier"/dissimilar tasks.

We further compare meta-learning on (A) the same setting as before by holding out the 10 meta-testing categories vs. (B) training on all categories in setting A as well as the dissimilar ones with IDs 3, 9, 39, 46, and 47. Table 4.7 shows results on the desired categories with IDs 19, 21, 23, 36, and 44. It can be observed that when a meta-learning model is trained to perform well (during meta-testing) on the desired categories/tasks,

training alongside with dissimilar ones can compromise its performance. More details of the experimental settings and data preparation, experimental results, and analysis are given in Appendix C.1.2. We have also empirically compared the time efficiency of IPML against that of several meta-learning baselines and reported the results in Appendix C.1.6.

4.5 Conclusion

We describe a novel IPML algorithm that, in contrast to existing works, explicitly represents each task as a continuous latent vector and models its probabilistic belief within the highly expressive IP framework. We show that IPML can be trained with an EM algorithm, where the E step uses the gradient-based sampling method SGHMC (*stochastic gradient Hamiltonian Monte Carlo*) to draw posterior samples, and the M step maximizes the meta-learning objective through gradient descent. IPML offers the benefits of being amenable to (a) the characterization of a principled distance measure between tasks using MMD, (b) active task selection, and (c) synthetic task generation of complicated image classifications via modeling of task-dependent input distributions using our X-Net. Empirical evaluation shows that IPML outperforms existing Bayesian meta-learning algorithms. We have also empirically demonstrated the application of IPML on an anonymous e-commerce company's real-world dataset.

In this chapter, we have successfully illustrated IPML's ability to perform active task selection, though the criterion we used is a naive one (maximum variance), and does not come with any theoretical guarantee. In the next chapter, we dig deeply into the problem of active task selection for meta-learning, and proposed a novel criterion based on the *mutual information between latent task vectors* (MILT) to quantify the informativeness of any subset of task. We also come up with a greedy algorithm that has a near-optimal guarantee on the MILT criterion of the selected tasks.

Chapter 5

Online Stein Variational Gradient Descent for Near-Optimal Task Selection in Meta-Learning

5.1 Background

Meta-learning exploits the experience from previous tasks to form a model (represented by meta-parameters) that can rapidly adapt to a new task with its few-shot data. Though meta-training requires only a few data points from each task, these tasks have to be representative of the distribution of meta-test tasks [Luna and Leonetti, 2020] in order to attain strong generalization performance for any meta-test task. To achieve this, one can naively consider acquiring (the data associated with) a massive number of tasks like that in various benchmark datasets, which in practice is prohibitively costly in time and money (e.g., due to manual labeling or data anonymization effort). Given a limited budget of k tasks to be acquired, one can simply select them randomly, but this often leads to a sub-optimal meta-test performance [Liu et al., 2020]. This motivates the problem of *active task selection* which involves selecting a subset of k most informative tasks from a finite yet large collection of candidate tasks for meta-training. These informative tasks are

expected to be most representative of the distribution of meta-test tasks among any other k tasks and hence allow meta-training to generalize best to any meta-test task.[1]

To address the active task selection problem, we start by choosing a probabilistic meta-learning framework for grounding our active task selection criterion based on information theory. From the existing works on probabilistic meta-learning [Chen et al., 2021, Finn et al., 2018, Rusu et al., 2019, Yoon et al., 2018], we choose our *implicit process-based meta-learning* (IPML) framework [Chen et al., 2021] (Chapter 4) that explicitly represents each task as a continuous latent vector and models its probabilistic belief within the highly expressive *implicit process* (IP) framework [Ma et al., 2019]. Representing each task as a latent task vector has a distinct advantage in that the dimension of the representation does not increase with the number of data points in a task, hence allowing active task selection criteria based on mutual information or entropy to be computed efficiently (Chapter 5.3.2). We then propose a novel active task selection criterion based on the *mutual information between latent task vectors* (MILT) to quantify the informativeness of any subset of tasks. Unfortunately, the MILT criterion scales poorly in the number of candidate tasks when optimized. To resolve this issue, we exploit the submodularity property of the MILT criterion for devising the first active task selection algorithm for meta-learning with a near-optimal performance guarantee (Chapter 5.3).

Our active task selection algorithm requires frequent belief updates of the meta-parameters, which can be computationally expensive. To further improve our efficiency, we design a forward-backward method based on our online variant of the *Stein variational gradient descent* (SVGD) [Liu and Wang, 2016] to perform fast belief updates such that a set of forward (and backward) particles is maintained and updated by learning (or unlearning) from each selected task (Chapter 5.4).

[1]Note that the motivation of active task selection is not to meta-learn faster in terms of wall-clock time.

5.2 Problem Formulation

Among the existing probabilistic meta-learning frameworks [Chen et al., 2021, Finn et al., 2018, Yoon et al., 2018], we adopt ours [Chen et al., 2021] (Chapter 4) which explicit represents each task as a continuous latent vector and models its probabilistic belief. Following Chapter 4, the inputs (outputs) for all candidate tasks are assumed to belong to the same input (output) space. Consider meta-learning on probabilistic regression tasks (the meta-learning on probabilistic classification tasks uses the robust-max likelihood [Hernández-Lobato et al., 2011] as introduced in Chapter 3.1.1): Each candidate task is assumed to be generated from a task distribution and associated with a dataset $(X, \mathcal{Y} = \mathbf{y})$ where X is a set of known/fixed input vectors. From now on, for terms conditioned on the known/fixed inputs, we omit the inputs to ease notations. $\mathcal{Y} \triangleq (y_{\mathbf{x}})_{\mathbf{x}\in X}^{\top}$ denotes a vector of the corresponding noisy outputs (random variables): $y_{\mathbf{x}} \triangleq f(\mathbf{x}) + \epsilon(\mathbf{x})$, which are outputs of an unknown function f corrupted by an i.i.d. Gaussian noise $\epsilon(\mathbf{x}) \sim \mathcal{N}(0, \sigma^2)$ with variance σ^2, and the vector $\mathbf{y}$ denotes a realization of $\mathcal{Y}$. Let f be distributed by an *implicit process* (IP) [Ma et al., 2019], as follows:

Definition 5.1 (extended definition of the Implicit process (IP) for a Bayesian treatment of the meta-parameters). *Let the collection of random variables $f(\cdot)$ denote an IP defined by meta-parameters (random variables) Θ: Every finite collection $\{f(\mathbf{x})\}_{\mathbf{x}\in X}$ has a joint prior belief $p(\mathbf{f} \triangleq (f(\mathbf{x}))_{\mathbf{x}\in X}^{\top})$ implicitly defined as:*

$$\mathcal{Z} \sim P(\mathcal{Z} = \mathbf{z}) = p(\mathbf{z}), \quad f(\mathbf{x}) \triangleq g_{\Theta}(\mathbf{x}, \mathcal{Z}) \tag{5.1}$$

for all $\mathbf{x} \in X$ where every latent task vector $\mathcal{Z} = \mathbf{z}$ (representing a task) is drawn from the prior belief $p(\mathbf{z}) \triangleq \mathcal{N}(0, \mathbf{I})$, and generator g_{Θ} can be an arbitrary model (in our work here, a deep neural network (DNN)) parameterized by meta-parameters Θ.

Note that the above definition is extended from Def. 4.1 such that it adopts a Bayesian treatment of the meta-parameters instead of a point estimate (which not only empirically

improves performance (Chapter 5.5), but also allows us to reasoning about the uncertainty in the meta-parameters). Let the meta-parameters (random variables) Θ follow a prior belief $P(\Theta = \theta) = p(\theta)$. The goal of meta-learning is to infer the posterior belief $P(\Theta|\mathcal{Y} = \mathbf{y})$ of meta-parameters Θ. Using $p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{f}, \sigma^2\mathbf{I})$ and the IP prior belief $p(\mathbf{f})$ from Def. 5.1, the marginal likelihood $P(\mathcal{Y} = \mathbf{y})$ can be derived by marginalizing out $\mathbf{f}$. Following Chapter 4.3, the coupling of $\mathcal{Z}$ with the IP model is by masking the last DNN layer's parameters (i.e., point-wise product) with $\mathcal{Z}$ during forward propagation (Fig. 5.1c).

Notations. Let T denote a finite collection of candidate tasks. For each candidate task $t \in T$, we consider a split of its dataset $(X_t, \mathcal{Y}_t = \mathbf{y}_t)$ into a small *sample* dataset[2] $(X_t^s, \mathcal{Y}_t^s = \mathbf{y}_t^s)$ known *a priori* and a large *remaining* dataset $(X_t^r, \mathcal{Y}_t^r = \mathbf{y}_t^r)$ to be acquired/observed *only after* this task is selected by an active task selection algorithm. So, $X_t = X_t^s \cup X_t^r$ and $X_t^s \cap X_t^r = \emptyset$. Such a split is similar to the support-query split of a meta-training task [Finn et al., 2018, Gordon et al., 2019, Ravi and Beatson, 2018, Yoon et al., 2018], albeit serving a different aim of active task selection here. For a subset $A \subseteq T$ of tasks, let $\mathcal{Y}_A^r \triangleq (\mathcal{Y}_t^r)_{t\in A}$ and $\mathbf{y}_A^r \triangleq (\mathbf{y}_t^r)_{t\in A}$ denote a realization of $\mathcal{Y}_A^r$. Similarly, let $\mathcal{Z}_A \triangleq (\mathcal{Z}_t)_{t\in A}$ concatenate the latent task vectors representing the tasks in A. Let $A \cup t$ denote the union of A and $\{t\}$.

Problem Definition. Given a finite collection T of candidate tasks with small, *a priori* known sample datasets $(X_t^s, \mathbf{y}_t^s)$ for all $t \in T$ (Chapter 5.2), the problem of *active task selection* is about selecting a subset $A \subseteq T$ of *most informative* tasks subject to a budget of $|A| = k$ tasks. After selecting A, the datasets $(X_t^r, \mathbf{y}_t^r)$ for all tasks $t \in A$ are acquired/observed for meta-training. From now on, for terms conditioned on the *a priori* known sample datasets, we omit them to ease notations.

Note that in this work, we assume the candidate tasks are fixed prior to our selection. So, we are not allowed to generate our own tasks by selecting classes for meta-learning of classification tasks, like in [Liu et al., 2020]. Also, we do not assume the availability of any contextual information (e.g., hyperparameters generating task data) on the tasks, like

[2]For N-way K-shot classification problems, the sample dataset has K data points per class and N classes.

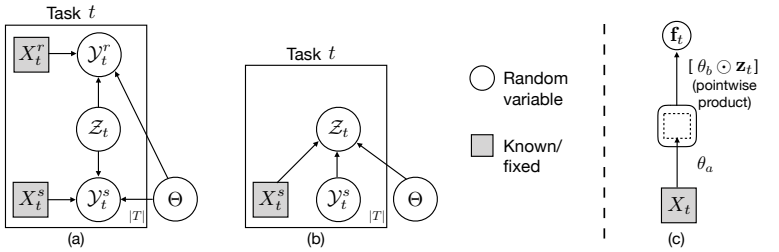

Figure 5.1: Graphical model of implicit process for (a) meta-learning and (b) task selection. We omit the random variable $f(\cdot)$ to simplify illustration since $f(\cdot) \rightarrow \mathcal{Y}$ is simply the addition of i.i.d. Gaussian noises. (c) DNN generator g_θ where $\theta \triangleq (\theta_a, \theta_b)$ and θ_a can be convolutions to obtain high-level representations of the input vector, while θ_b is the last DNN layer's parameters.

in [Kaddour et al., 2020]. Our only assumption is the availability of known sample datasets for each task, which is realistic and easy to achieve in practice. An example of our problem setting is when a meta-learner wants to purchase data from decentralized data marketplaces. These decentralized data marketplaces enable a secure data exchange between the participants. Powered by blockchain, they can maintain the anonymity of their participants which is why they are often used for the trading of personal data. They usually offer a period of free subscription such that the sample datasets can be easily acquired. In reality, such a decentralized data marketplace can be found in established data sharing platforms like Streamr Marketplace (`https://streamr.network/marketplace`) which offers data like real-time finance market data, and HIT Foundation [Eberhard and Paul, 2019] which offers healthcare data like radiology image data.

Meta-Learning Algorithms used after Active Task Selection. Given the selected A, in principle, we are free to use any meta-learning algorithms to obtain a final model, including probabilistic methods [Finn et al., 2018, Yoon et al., 2018] or non-probabilistic [Finn et al., 2017] methods. We ground our active task selection criterion on IP. Probabilistic meta-learning of IP can be defined as the inference of meta-parameters Θ and Chapter 4.3 has used an *expectation maximization* (EM) algorithm to perform meta-training such that the E step carries out the IP inference of $\mathcal{Z}$ while the M step optimizes Θ. Since our focus is on active task selection instead of deriving a meta-learning algorithm, a detailed discussion of the meta-learning algorithms can be referred to Appendix D.2,

where we have extended the EM algorithm we introduced in Chapter 4.3 so that it can deal with a Bayesian treatment of meta-parameters. Fig. 5.1 shows the graphical models of the IP for task selection as well as meta-learning.

5.3 Mutual Information between Latent Task vectors

5.3.1 MILT Criterion

To quantify the informativeness of a subset A of tasks, our proposed active task selection criterion measures its reduction in uncertainty/entropy of the latent task vectors $\mathcal{Z}_{T\setminus A}$ representing the other candidate tasks in $T \setminus A$ or, equivalently, its information gain $I(\mathcal{Z}_A; \mathcal{Z}_{T\setminus A})$ on them. We use the prior entropy $H(\mathcal{Z}_{T\setminus A})$ and posterior entropy $H(\mathcal{Z}_{T\setminus A}|\mathcal{Z}_A)$ to represent the uncertainty of $\mathcal{Z}_{T\setminus A}$ before and after conditioning on $\mathcal{Z}_A$. Then,

$$I(\mathcal{Z}_A; \mathcal{Z}_{T\setminus A}) \triangleq H(\mathcal{Z}_{T\setminus A}) - H(\mathcal{Z}_{T\setminus A}|\mathcal{Z}_A)\,, \tag{5.2}$$

which we call the *mutual information between latent task vectors* (MILT) criterion. For simplicity, we define the set function $\text{MILT}(A) \triangleq I(\mathcal{Z}_A; \mathcal{Z}_{T\setminus A})$. The k most informative candidate tasks thus correspond to the subset $A^\star \subseteq T$ with the largest information gain $\text{MILT}(A^\star)$ on $\mathcal{Z}_{T\setminus A^\star}$:

$$A^\star \triangleq \arg\max_{A\subseteq T, |A|=k} \text{MILT}(A)\,. \tag{5.3}$$

Unfortunately, the MILT criterion (5.2) scales poorly in the number $|T|$ of candidate tasks when optimized in (5.3). In fact, solving (5.3) has been shown to be NP-hard even when $\mathcal{Z}_T$ follows a tractable multivariate Gaussian distribution [Ko et al., 1995]. Inspired by the work of [Krause et al., 2008], we will now present a polynomial-time greedy algorithm that can exploit the submodularity of MILT to guarantee a $(1 - 1/e)$-factor approximation of that achieved by $A^\star$:

Algorithm 1 (**Near-optimal active task selection (informal)**). *Start with an empty set $A_0 = \emptyset$ of tasks. In each round $i = 1, \ldots, k$, greedily select the next task:*

$$t_i^\star \triangleq \arg\max_t \text{MILT}(A_{i-1} \cup t) - \text{MILT}(A_{i-1}) \tag{5.4}$$

and update the set $A_i = A_{i-1} \cup t_i^\star = \{t_1^\star, \ldots, t_i^\star\}$ of selected tasks.

Theorem 5.1 (**Near-optimal guarantee**). *Algorithm 1 is guaranteed to select a set A of k tasks s.t.*

$$\text{MILT}(A) \geq (1 - 1/e)(\text{OPT} - C_0) \tag{5.5}$$

where $\text{OPT} \triangleq \text{MILT}(A^\star)$ *and the constant $C_0 \triangleq H(\Theta)$ is the prior entropy of meta-parameters Θ.*

Its proof is in Appendix D.3.1. Note that for a monotonic submodular set function, a greedy algorithm can be designed to guarantee a $(1 - 1/e)$-factor approximation of OPT [Krause and Golovin, 2014]. Though the $\text{MILT}(\cdot)$ function is submodular, it is not strictly monotonic. Nevertheless, we can show that $\text{MILT}(\cdot)$ is approximately monotonic up to a constant error of C_0, which suffices for the proof of Theorem 5.1. This approximate monotonicity holds due to $\mathcal{Z}_t, \mathcal{Z}_{t'}, \forall t \neq t'$ being mutually independent given meta-parameters Θ.

5.3.2 Advantages of MILT over other Active Task Selection Criteria

For our active task selection problem, other criteria can be considered. For example, A can be selected to maximize the *mutual information between remaining datasets of tasks* (MIRD) criterion $I(\mathcal{Y}_A^r; \mathcal{Y}_{T\setminus A}^r) \triangleq H(\mathcal{Y}_{T\setminus A}^r) - H(\mathcal{Y}_{T\setminus A}^r | \mathcal{Y}_A^r)$. In fact, we can also prove a similar near-optimal performance guarantee for MIRD, as shown in Appendix D.3.2. However, any criterion involving probabilities conditioned on $\mathcal{Y}_A^r$ tends to be computationally challenging to evaluate: Suppose that each task has a potentially large remaining dataset of size R. Since the dimension of $\mathcal{Y}_A^r$ is proportional to $R|A|$, it becomes computationally challenging to compute an accurate Monte Carlo estimation of MIRD. So, it is not

Table 5.1: Comparison of different active task selection criteria. The last column indicates the dimension of the sample space of the Monte Carlo sampling step when evaluating the active task selection criterion.

Criterion	Expression	Submodular	Approx. monotonic	Near-optimal	Sample space
MILT	$H(\mathcal{Z}_{T\setminus A}) - H(\mathcal{Z}_{T\setminus A}\|\mathcal{Z}_A)$	✓	✓	✓	$\mathcal{O}(k)$
MIRD	$H(\mathcal{Y}^r_{T\setminus A}) - H(\mathcal{Y}^r_{T\setminus A}\|\mathcal{Y}^r_A)$	✓	✓	✓	$\mathcal{O}(Rk)$
ELT	$H(\mathcal{Z}_A)$	✓	✗	✗	$\mathcal{O}(k)$
Variance	$\mathrm{Var}(\mathcal{Z}_t\|\mathcal{Z}_{A_{i-1}})$ for $i = 1, \ldots, k$	✗	✗	✗	$\mathcal{O}(k)$

computationally feasible to evaluate the MIRD criterion when R is large. In contrast, MILT does not suffer from this curse of dimensionality since the dimension of a latent task vector $\mathcal{Z}$ in (5.1) does not increase with R. Such an advantage of latent task modeling motivates us to use the IP framework (Def. 5.1).

As another example, A can be selected to maximize the *entropy of latent task vectors* (ELT) criterion $H(\mathcal{Z}_A)$. However, directly maximizing ELT is NP-hard even when $\mathcal{Z}_T$ follows a tractable multivariate Gaussian distribution [Ko et al., 1995]. So, we usually resort to greedily selecting $\arg\max_t H(\mathcal{Z}_t|\mathcal{Z}_{A_{i-1}})$ for $i = 1, \ldots, k$, which has a comparable computational cost as Algo. 1. It is well-known that although the entropy criterion is submodular, it may not be monotonic [Krause and Golovin, 2014]. Hence, such a greedy algorithm based on ELT does not enjoy the near-optimal performance guarantee and often performs sub-optimally [Krause et al., 2008].

We have proposed to greedily select the next task with the maximum variance of $\mathcal{Z}_t$ in Chapter 4: $t^\star_i = \arg\max_t \mathrm{Var}(p(\mathbf{z}_t|\theta, \mathbf{y}^s_t))$ for $i = 1, \ldots, k$. However, doing so neglects the dependence of $\mathcal{Z}_t$ on $\mathcal{Z}_{A_{i-1}}$; it can be observed from the Fig. 5.1b that they are dependent when $\mathcal{Y}^s_t$ and $\mathcal{Y}^s_{A_{i-1}}$ are both observed. To correctly account for such a dependence, the following variance criterion should be considered instead: $t^\star_i = \arg\max_t \mathrm{Var}(\mathcal{Z}_t|\mathcal{Z}_{A_{i-1}})$ for $i = 1, \ldots, k$, which has a similar computational cost as the greedy algorithm based on ELT, but does not have any performance guarantee. A comparison of the above-discussed active criteria are summarized in Table 5.1.

5.4 Efficient Evaluation of the Greedy MILT Criterion

Let $\overline{A} \triangleq T \setminus A$. From (5.4),

$$\begin{aligned} & \text{MILT}(A_{i-1} \cup t) - \text{MILT}(A_{i-1}) \\ = \; & H(\mathcal{Z}_t|\mathcal{Z}_{A_{i-1}}) - H(\mathcal{Z}_t|\mathcal{Z}_{\overline{A_{i-1} \cup t}}) \; . \end{aligned} \tag{5.6}$$

In (5.6), the posterior entropy $H(\mathcal{Z}_t|\mathcal{Z}_A) = -\mathbb{E}_{\mathbf{z}_A \sim p(\mathbf{z}_A)} \left[\int_{\mathbf{z}_t} p(\mathbf{z}_t|\mathbf{z}_A) \log p(\mathbf{z}_t|\mathbf{z}_A) d\mathbf{z}_t \right]$ can be approximated by Monte Carlo sampling from $p(\mathbf{z}_A)$ such that for each sample of $\mathbf{z}_A$, we compute

$$p(\mathbf{z}_t|\mathbf{z}_A) = \int_\theta \underbrace{p(\theta|\mathbf{z}_A, \mathbf{y}_A^s)}_{\text{particles (Chapter 5.4.1)}} \underbrace{p(\mathbf{z}_t|\theta, \mathbf{y}_t^s)}_{\text{Gaussian (Chapter 5.4.2)}} \; \mathrm{d}\theta \; . \tag{5.7}$$

From (5.7), the selection of A affects that of the next task t through the common meta-parameters Θ in the IP model, as shown in Fig. 5.1.

5.4.1 Variational Inference (VI) with Particle Representation of Θ

In this subsection, we will describe the procedure to obtain the approximation of $p(\theta|\mathbf{z}_A, \mathbf{y}_A^s)$. We use a particle representation of Θ and a stochastic VI method for particles called the *Stein variational gradient descent* (SVGD) [Liu and Wang, 2016] to compute the posterior belief of Θ. Applied in a previous Bayesian meta-learning framework [Yoon et al., 2018], SVGD combines the strengths of MCMC and variational inference and enjoys a similar time efficiency as stochastic gradient descent. SVGD represents the belief of Θ with a set $\{\theta^m\}_{m=1}^M$ of M particles.

Though $p(\theta|\mathbf{z}_A, \mathbf{y}_A^s)$ in (5.7) cannot be evaluated in closed form, it can be computed via SVGD: Its gradient $\nabla_\theta \log p(\theta|\mathbf{z}_A, \mathbf{y}_A^s) = \nabla_\theta \log[p(\mathbf{y}_A^s|\theta, \mathbf{z}_A)p(\theta)]$ is available given our neural network implementation of IP: $p(\mathbf{y}_A^s|\theta, \mathbf{z}_A) = \prod_{t \in A} p(\mathbf{y}_t^s|\mathbf{f}_t^s = (g_\theta(\mathbf{x}, \mathbf{z}_t))_{\mathbf{x} \in X_t^s}^\top)$ where g_θ is a neural network parameterized by θ (Def. 5.1). We perform SVGD on the

observed tuples $\{\mathbf{z}_A, X_A^s, \mathbf{y}_A^s\}$ by first initializing each particle as a sample from $p(\theta)$[3] and then, in each SVGD iteration, updating each particle θ^m as

$$\begin{aligned}\theta^m \leftarrow \theta^m + \frac{\eta}{M} \sum_{\theta \in \{\theta^m\}_{m=1}^M} \big[k(\theta, \theta^m) \\ \times \nabla_\theta \log p(\theta|\mathbf{z}_A, \mathbf{y}_A^s) + \nabla_\theta k(\theta, \theta^m)\big]\end{aligned} \tag{5.8}$$

where η is the step size and $k(\cdot, \cdot)$ is a radial basis function kernel representing a repulsive force between particles to prevent them from collapsing. We denote the abstraction of the entire VI process (containing multiple SVGD iterations till convergence) to obtain $p(\theta|\mathbf{z}_A, \mathbf{y}_A^s)$ as $p(\theta|\mathbf{z}_A, \mathbf{y}_A^s) \leftarrow \text{SVGD}_\Theta(p(\theta), \{\mathbf{z}_A, \mathbf{y}_A^s\})$ where the first input is the initialization of the particles and the second input is the observed tuples. By setting $M = 1$, we will recover the gradient descent method to compute the point estimate of Θ.

5.4.2 VI with Gaussian Approximation of $\mathcal{Z}$

To model $p(\mathbf{z}_t|\theta, \mathbf{y}_t^s)$ in (5.7), we use VI with a Gaussian approximation of the posterior belief which makes use of the gradient on $\mathcal{Z}$: For each particle θ^m, we perform VI to obtain the mean and variance parameterization[4] of a Gaussian distribution $q(\mathbf{z}_t|\theta^m)$ by maximizing the evidence lower bound $\text{ELBO}_\mathcal{Z} \triangleq \mathbb{E}_{q(\mathbf{z}_t|\theta^m)}[\log p(\mathbf{y}_t^s|\mathbf{z}_t, \theta^m)] - \text{KL}(q(\mathbf{z}_t|\theta^m)\|p(\mathbf{z}_t))$ where KL stands for Kullback–Leibler divergence. As a result, $p(\mathbf{z}_t|\mathbf{z}_A)$ will be a *mixture of Gaussians* (since $p(\theta|\mathbf{z}_A, \mathbf{y}_A^s)$ is a set of M particles), which allows us to arrive at an easy approximation of the posterior entropy and hence (5.6). We denote the whole VI process as: $p(\mathbf{z}_t|\mathbf{z}_A) \leftarrow \text{VI}_\mathcal{Z}(p(\theta|\mathbf{z}_A, \mathbf{y}_A^s))$.

5.4.3 Forward-Backward Method for Efficiently Evaluating (5.6)

In this subsection, we will describe a forward-backward method based on online SVGD to perform fast belief updates of the meta-parameters, thus improving the efficiency

[3]Note that this is not strictly needed for SVGD to converge.

[4]The mean and variance are optimized using gradient descent via the reparametrization trick.

Algorithm 4 Near-Optimal Active Task Selection based on MILT

Set $A = \emptyset$;
Initialize forward particles: $p(\theta|\mathbf{z}_A, \mathbf{y}_A^s) = p(\theta)$;
Initialize backward particles:
  $p\Big(\theta|\mathbf{z}_{\overline{A}}, \mathbf{y}_{\overline{A}}^s\Big) = p\big(\theta|\mathbf{z}_T, \mathbf{y}_T^s\big) \leftarrow \text{SVGD}_\Theta\left(p(\theta), \{\mathbf{z}_T, \mathbf{y}_T^s\}\right)$;
while $|A| < k$ **do**
  Sample $\mathbf{z}_A \sim p(\mathbf{z}_A)$, $\mathbf{z}_{\overline{A}} \sim p(\mathbf{z}_{\overline{A}})$;
  for $t \in T \setminus A$ **do**
    Compute with forward particles: $p(\mathbf{z}_t|\mathbf{z}_A) \leftarrow \text{VI}_{\mathcal{Z}}(p(\theta|\mathbf{z}_A, \mathbf{y}_A^s))$;
    With backward particles:
      $p(\mathbf{z}_t|\mathbf{z}_{\overline{A\cup t}}) \leftarrow \text{VI}_{\mathcal{Z}}\left(\text{SVGD}_\Theta^{-1}\left(p\Big(\theta|\mathbf{z}_{\overline{A}}, \mathbf{y}_{\overline{A}}^s\Big), \{\mathbf{z}_t, \mathbf{y}_t^s\}\right)\right)$;
    Estimate $H(\mathcal{Z}_t|\mathcal{Z}_A) - H(\mathcal{Z}_t|\mathcal{Z}_{\overline{A\cup t}})$;
  end for
  Select $t^\star = \arg\max_t H(\mathcal{Z}_t|\mathcal{Z}_A) - H(\mathcal{Z}_t|\mathcal{Z}_{\overline{A\cup t}})$;
  Update forward particles:
    $p\big(\theta|\mathbf{z}_{A\cup t^\star}, \mathbf{y}_{A\cup t^\star}^s\big) \leftarrow \text{SVGD}_\Theta\left(p\big(\theta|\mathbf{z}_A, \mathbf{y}_A^s\big), \{\mathbf{z}_{t^\star}, \mathbf{y}_{t^\star}^s\}\right)$;
  Update backward particles:
    $p\Big(\theta|\mathbf{z}_{\overline{A\cup t^\star}}, \mathbf{y}_{\overline{A\cup t^\star}}^s\Big) \leftarrow \text{SVGD}_\Theta^{-1}\left(p\Big(\theta|\mathbf{z}_{\overline{A}}, \mathbf{y}_{\overline{A}}^s\Big), \{\mathbf{z}_{t^\star}, \mathbf{y}_{t^\star}^s\}\right)$;
  Update $A = A \cup t^\star$;
end while
return A

in evaluating (5.6). When we proceed from round $(i-1)$ to i, we update the set $A_i = A_{i-1} \cup t_i^\star$ of selected tasks. Then, $p(\theta|\mathbf{z}_{A_i}, \mathbf{y}^s_{A_i})$ can be correspondingly updated from $p(\theta|\mathbf{z}_{A_{i-1}}, \mathbf{y}^s_{A_{i-1}})$ by learning from the (sample dataset of) newly added task $t_i^\star$ in an online manner:[5] $p(\theta|\mathbf{z}_{A_i}, \mathbf{y}^s_{A_i}) \leftarrow \text{SVGD}_\Theta(p(\theta|\mathbf{z}_{A_{i-1}}, \mathbf{y}^s_{A_{i-1}}), \{\mathbf{z}_{t_i^\star}, \mathbf{y}^s_{t_i^\star}\})$. To this end, we only maintain a single set of particles and update it in place. We refer to the particles in this set as *forward particles*. Note that a significant advantage here is that in practice, by learning only from the newly added task, we need much fewer SVGD iterations (around 5) to converge to $p(\theta|\mathbf{z}_{A_i}, \mathbf{y}^s_{A_i})$ compared with naively learning $p(\theta|\mathbf{z}_{A_i}, \mathbf{y}^s_{A_i})$ from scratch (i.e., from $p(\theta)$).

The same trick applies when we compute $H(\mathcal{Z}_t|\mathcal{Z}_{\overline{A_{i-1} \cup t}})$ in (5.6). Firstly, we obtain $p(\theta|\mathbf{z}_{\overline{A_i}}, \mathbf{y}^s_{\overline{A_i}})$ which can be updated from $p(\theta|\mathbf{z}_{\overline{A_{i-1}}}, \mathbf{y}^s_{\overline{A_{i-1}}})$ by *unlearning* from the task $t_i^\star$. As shown in [Nguyen et al., 2020b], unlearning in the variational Bayes setting can be cast exactly as a minimization of an *evidence upper bound* (EUBO)[6], which is also applicable to SVGD, as stated below:

Proposition 5.1 (**Online SVGD for unlearning**). *Suppose that $p(\theta)$ is an uninformative prior (e.g., $\nabla_\theta \log p(\theta) = 0$). With $\{\theta^m\}_{m=1}^M$ initially sampled from $p(\theta|\mathbf{z}_A, \mathbf{y}^s_A)$, the SVGD operation for obtaining $p(\theta|\mathbf{z}_{A\setminus t}, \mathbf{y}^s_{A\setminus t})$ (i.e., unlearning from a task $t \in A$) is*

$$\theta^m \leftarrow \theta^m + \frac{\eta}{M} \sum_{\theta \in \{\theta^m\}_{m=1}^M} \Big[-k(\theta, \theta^m) \times \nabla_\theta \log p(\theta|\mathbf{z}_t, \mathbf{y}^s_t) + \nabla_\theta k(\theta, \theta^m) \Big] . \tag{5.9}$$

We denote such an unlearning process (containing multiple SVGD iterations till convergence) as $p(\theta|\mathbf{z}_{A\setminus t}, \mathbf{y}^s_{A\setminus t}) \leftarrow \textit{SVGD}^{-1}_\Theta(p(\theta|\mathbf{z}_A, \mathbf{y}^s_A), \{\mathbf{z}_t, \mathbf{y}^s_t\})$.

Note that SVGD for unlearning (5.9) differs from that for learning (5.8): The sign of the likelihood gradient is reversed due to unlearning, while the sign of the kernel gradient is not as it corresponds to a repulsive force term which remains the same in both Stein operators of learning and unlearning. The proof of Proposition 5.1 (Appendix D.3.4)

[5] A relevant proposition that formally describes this online SVGD is provided in Appendix D.3.3.
[6] See Appendix D.3.4 for more details.

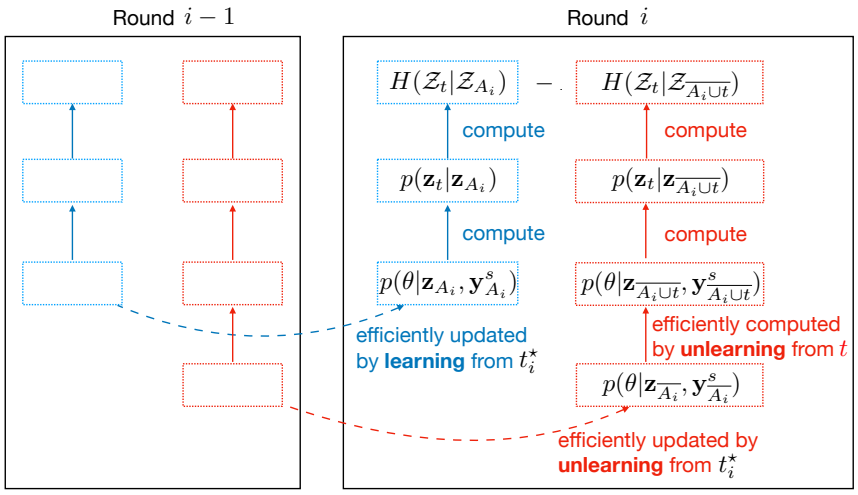

Figure 5.2: Computational graph of evaluating (5.6) with the forward-backward method in Chapter 5.4.3. The blue segments are computed using forward particles while the red segments are computed using backward particles.

is obtained by deriving the Stein operator [Liu and Wang, 2016] for the minimization of EUBO (i.e., unlearning). We make use of the unlearning variant of online SVGD by maintaining another set of *backward particles*, which is initialized as $p(\theta|\mathbf{z}_T, \mathbf{y}^s_T)$. We then update it in place in every round of task selection through unlearning from task $t_i^\star$. Note that unlearning from a single task needs much fewer SVGD iterations (around 5) to converge to $p(\theta|\mathbf{z}_{\overline{A_i}}, \mathbf{y}^s_{\overline{A_i}})$ compared with learning it from scratch (i.e., from $p(\theta)$). Unlearning can also be applied to obtain $p(\theta|\mathbf{z}_{\overline{A_i \cup t}}, \mathbf{y}^s_{\overline{A_i \cup t}})$ (i.e., for $H(\mathcal{Z}_t|\mathcal{Z}_{\overline{A_{i-1} \cup t}})$ in (5.6)) such that $p(\theta|\mathbf{z}_{\overline{A_i \cup t}}, \mathbf{y}^s_{\overline{A_i \cup t}})$ is obtained by unlearning $p(\theta|\mathbf{z}_{\overline{A_i}}, \mathbf{y}^s_{\overline{A_i}})$ from task t.

Time complexity. Fig. 5.2 shows a computational graph of evaluating (5.6). Algo. 4 describes a detailed variant of our near-optimal active task selection algorithm that utilizes the forward-backward method. We show in Appendix D.1 that its computational cost scales linearly in $|T|$. A detailed variant of the algorithm that uses the *naive* method (without the forward-backward method) is included in Appendix D.1 where we show that its computational cost scales quadratically in $|T|$.

5.5 Experiments and Discussion

In this section, we will empirically compare the performance of three active task selection criteria listed in Table 5.1: (a) greedy algorithm based on MILT (5.4), (b) greedy algorithm based on ELT (Chapter 5.3.2), (c) greedy algorithm based on an improved variance criterion (Chapter 5.3.2) over the method in Chapter 4.4, and also random task selection using three benchmark datasets. Note that the comparison does not include MIRD due to its potentially high computational cost (Chapter 5.3.2). We have also included a comparison with [Luna and Leonetti, 2020] in Appendix D.4.3.[7] We have also adapted the greedy class-pair based sampling (GCP) proposed in [Liu et al., 2020], and the probabilistic active meta-learning algorithm (PAML) proposed in [Kaddour et al., 2020] which have different problem settings, and compared with them in Appendix D.4.2.

Sinusoid regression. In this setting, the data of each regression task is sampled from a sinusoid wave where the amplitude and phase vary between tasks. Following the experimental setting of [Finn et al., 2017], the amplitude varies within $[0.1, 5]$, the phase varies within $[0, \pi]$, and the input $\mathbf{x}$ is sampled uniformly from $[-5, 5]$. We perform experiments in both the 5-shot setting (i.e., $|X_t^s| = |X_t^r| = 5$) and the 10-shot setting (i.e., $|X_t^s| = |X_t^r| = 10$). We randomly sample 1000 tasks as T.

Omniglot classification. Omniglot [Lake et al., 2011] is a benchmark few-shot image classification dataset consisting of 20 instances of 1623 characters from 50 different alphabets. Our experiment adopts the same task generation process as that in [Finn et al., 2017] (i.e., downsampling to 28×28 and applying random rotations). To further accelerate computation, we select tasks in a *batch* manner: Each task consists of 32 sub-tasks such that each sub-task is a 1-shot 5-way (i.e., $|X_t^s| = |X_t^r| = 5$) or 1-shot 20-way (i.e., $|X_t^s| = |X_t^r| = 20$) classification.[8] We randomly generate 2000 tasks as T.

MiniImageNet classification. The MiniImageNet [Ravi and Larochelle, 2017] dataset involves 64 training classes, 12 validation classes, and 24 test classes of 84×84 RGB im-

[7] We use regression benchmark datasets as surrogate examples of real-time data like finance market data, and classification benchmark datasets as surrogate examples of healthcare radiology image data.

[8] The active task selection criterion is a sum of the criterion over all sub-tasks.

ages. Our experiment adopts the same task generation process as that in [Finn et al., 2017] (i.e., applying random rotations). Similarly, we select tasks in a *batch* manner: Each task consists of 32 sub-tasks such that each sub-task is a 1-shot 5-way (i.e., $|X_t^s| = |X_t^r| = 5$) classification.[8] We randomly generate 2000 tasks as T.

The IP model (Chapter 5.2) is a fully-connected neural network with 2 hidden layers of size 40 with ReLU nonlinearities for Sinusoid, and a convolutional neural network with 4 modules of 3×3 convolutions and 64 filters, followed by batch normalization, ReLU nonlinearities, and strided convolutions for Omniglot or 2×2 max-pooling for MiniImageNet. We use $M = 5$ particles, and set the step size η as 0.05 for sinusoid and MiniImageNet and 0.5 for Omniglot.

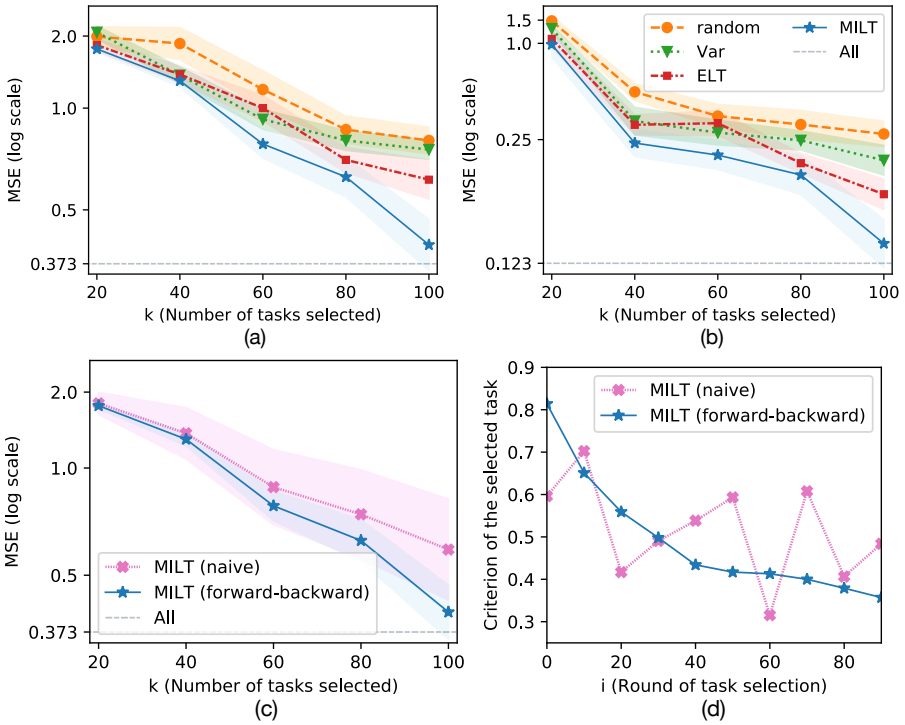

Figure 5.3: Meta-test *mean squared error* (MSE) and standard error over 5 runs vs. no. k of selected tasks on (a) 5-shot Sinusoid and (b) 10-shot Sinusoid. (c) Comparison of meta-test MSE between forward-backward method and naive method on 5-shot Sinusoid. *All* means meta-test MSE of the IP model trained on T (i.e., all 1000 candidate tasks). (d) plots the greedy criterion value $\text{MILT}(A_{i-1} \cup t_i^\star) - \text{MILT}(A_{i-1})$ (5.4) corresponding to selected task $t_i^\star$ vs. round i of selection.

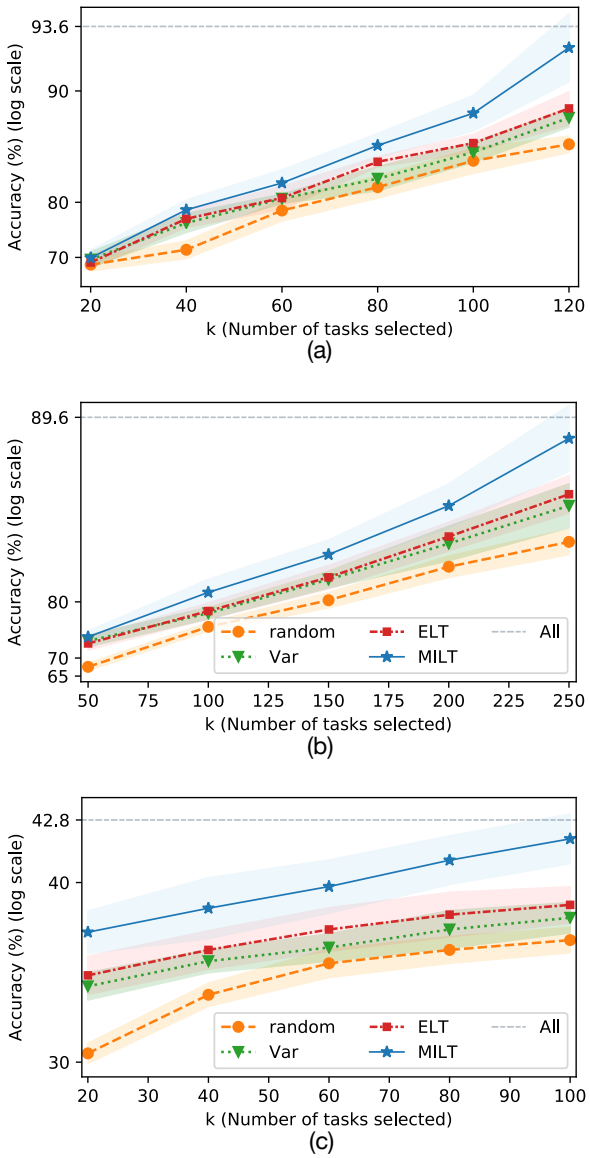

Figure 5.4: Meta-test accuracy (%) and standard error over 5 runs vs. no. k of selected tasks on (a) 1-shot 5-way Omniglot, (b) 1-shot 20-way Omniglot, and (c) 1-shot 5-way MiniImageNet. *All* means meta-test accuracy of the IP model trained on T (i.e., all 2000 candidate tasks).

5.5.1 Discussion of Baseline Comparisons

It can be observed from Figs. 5.3 and 5.4 that MILT outperforms all other baselines, which demonstrates the effectiveness of our proposed algorithm (Algo. 4). Random task selection performs the worst among all baselines, which is expected. ELT slightly outperforms the Variance baseline in nearly all cases, likely due to the entropy being able to better capture the uncertainty in $p(\mathbf{z}_t|\mathbf{z}_A)$ (5.7) which follows a mixture of Gaussians instead of a Gaussian.

Fig. 5.3 shows results of Sinusoid regression. Fig. 5.3b shows that actively selecting $k = 40$ tasks with MILT can already achieve a lower MSE of 0.241 than randomly selecting $k = 100$ tasks (MSE of 0.266). For both 5-shot (Fig. 5.3a) and 10-shot (Fig. 5.3b) settings, MILT achieves comparable performance to the IP model trained with all 1000 candidate tasks in T when actively selecting only 10% from T (i.e., $k = 100$ tasks).

Fig. 5.4 shows results of Omniglot and MiniImageNet classifications. For both Omniglot and MiniImageNet, we have generated 2000 candidate tasks in T. Meta-training with all these candidate tasks achieves a meta-test accuracy of 93.6% for 1-shot 5-way Omniglot, 89.6% for 1-shot 20-way Omniglot, and 42.8% for 1-shot 5-way MiniImageNet (Fig. 5.4). Previous works [Chen et al., 2021, Finn et al., 2017, Finn et al., 2018, Yoon et al., 2018] have generated 200000 tasks (batches) for meta-training and hence achieve higher meta-test accuracy on these cases. Nevertheless, our ablation study in Appendix E.3 shows that when selecting ($\sim$250) tasks, MILT does not need such a massive number (≈ 200000) of candidate tasks to achieve competitive meta-test performance.

Fig. 5.4a shows that actively selecting only 120 tasks with MILT can achieve a meta-test accuracy of 91.9% for 1-shot 5-way Omniglot, which is within one standard error from that achieved by the IP model trained on all 2000 candidate tasks. Fig. 5.4b&c also show that for 1-shot 20-way Omniglot and 1-shot 5-way MiniImageNet, similar observations hold when selecting only 250 tasks with MILT. All these observations further demonstrate the effectiveness of our proposed algorithm (Algo. 4).

5.5.2 Ablation Study

Effect of using particles. Table 5.2 shows results of meta-test performance of MILT with varying number M of particles (Chapter 5.4.1) from 1 (point estimate) to 5. It can be observed that increasing M consistently yields better performance for both 5-shot and 10-shot Sinusoid regression, thus indicating that our Bayesian treatment of the meta-parameters can improve the meta-test performance.

Forward-backward method based on online SVGD. Here, we compare the performance of the forward-backward method (Algo. 4) with the naive method (Chapter 5.4.3). Table 5.3 presents a comparison of their runtime, which meets our theoretical analysis. Fig. 5.3c presents a comparison of their meta-test performance: It may be surprising to observe that the forward-backward method clearly outperforms the naive method. In theory, we should expect them to perform similarly since the forward-backward method only seems to improve the efficiency. However, in practice, the forward-backward method performs better because it does not introduce *extra randomness* which may potentially violate the submodularity property of MILT. To see this, Fig. 5.3d plots the greedy criterion value $\text{MILT}(A_{i-1} \cup t_i^\star) - \text{MILT}(A_{i-1})$ (5.4) corresponding to selected task $t_i^\star$ vs. round i of active task selection. Since MILT is theoretically submodular, we expect to see a monotonically decreasing curve in Fig. 5.3d. However, we only observe such a monotonicity with the forward-backward method. This is because the evaluation of (5.4) involves approximation: It approximates the belief of the meta-parameters through SVGD. However, the naive method performs SVGD by initializing the particles randomly, which introduces randomness such that its approximation of the belief can be inconsistent between successive rounds. In contrast, the forward-backward method always performs belief updates with only one task and thus maintains a consistent belief because the belief updates are highly correlated between successive rounds. Such a "consistency" retains the submodularity of (our approximation of) MILT, which allows our algorithm (Algo. 4) to perform satisfactorily, as implied by its near-optimal performance guarantee.

Table 5.2: Meta-test *mean squared error* (MSE) over 5 runs with varying no. M of particles on 5-shot Sinusoid regression. MILT is used to select $k = 100$ tasks from $|T| = 1000$ candidate tasks.

	$M = 1$	$M = 3$	$M = 5$
5-shot	0.490	0.454	0.430
10-shot	0.144	0.137	0.129

Table 5.3: Mean runtime (seconds) on 5-shot Sinusoid regression. MILT is used to select from $|T| = 1000$ candidate tasks.

	forward-backward	naive
$k = 20$	24.5	153
$k = 40$	38.9	293
$k = 60$	54.6	425

5.5.2.1 Performance Sensitivity to Number of Sub-tasks in a Task

In Chapter 5.5, we have mentioned that for Omniglot and MiniImageNet classifications, we have selected tasks in a *batch* manner: Each task consists of 32 sub-tasks such that each sub-task is a 1-shot 5-way or 1-shot 20-way classification. We refer to the number of sub-tasks in a task as the *batch size*. For this *batch* setting, the active task selection criterion is the sum of the criterion over all sub-tasks (see footnote 8). Note that such a summation is exact if we know *a priori* that all the sub-tasks in task t correspond to the same latent task vector $\mathcal{Z}_t$. Nevertheless, we will investigate here whether the meta-test performance is sensitive to varying batch sizes.

For Sinusoid regression, we fix the total number of (5-shot or 10-shot) sub-tasks to be selected as 100 (e.g., when the batch size is 10, $k = 10$) and the total number of sub-tasks in the candidate tasks to be 1000.[9] For 1-shot 20-way Omniglot classification, we fix the total number of sub-tasks to be selected as 4000 and the total number of sub-tasks in the candidate tasks to be 64000.

Tables 5.4 and 5.5 show results of Sinusoid regression and Omniglot classification, respectively. It can be observed that the meta-test performance of MILT is similar across (and hence not so sensitive to) varying batch sizes which are small compared to the total number of sub-tasks to be selected.

[9]Doing so fixes the number of 5-shot or 10-shot sub-tasks that are available for selection.

Table 5.4: Meta-test *mean squared error* (MSE) and standard deviation over 5 runs on Sinusoid regression.

Batch size:	1	5	10
5-shot	0.39 ± 0.16	0.46 ± 0.14	0.43 ± 0.10
10-shot	0.12 ± 0.03	0.12 ± 0.03	0.13 ± 0.03

Table 5.5: Meta-test accuracy ($\%$) and standard deviation over 5 runs on Omniglot classification.

Batch size:	16	32	64
1-shot 20-way	89.1 ± 2.9	88.6 ± 3.0	88.8 ± 3.4

Table 5.6: Meta-test *mean squared error* (MSE) and standard deviation over 5 runs on Sinusoid regression.

No. of candidate tasks:	100	500	1000	2000
5-shot Sinusoid, $k = 100$	0.77 ± 0.19	0.48 ± 0.09	0.43 ± 0.10	0.40 ± 0.07
10-shot Sinusoid, $k = 100$	0.27 ± 0.10	0.19 ± 0.03	0.13 ± 0.03	0.11 ± 0.03

Table 5.7: Meta-test accuracy ($\%$) and standard deviation over 5 runs.

No. of candidate tasks:	1000	2000	4000
1-shot 5-way Omniglot, $k = 120$	91.0 ± 3.3	91.9 ± 3.0	91.8 ± 3.0
1-shot 20-way Omniglot, $k = 250$	87.5 ± 2.6	88.6 ± 3.0	88.6 ± 2.6

5.5.2.2 Performance Sensitivity to Number of Candidate Tasks

In Chapter 5.5, we have mentioned that we have generated a limited number of candidate tasks (i.e., 1000 for Sinusoid regression and 2000 for Omniglot classification). We have argued in Chapter 5.5 that we did not generate a massive number (≈ 200000) of candidate tasks since (a) it will require much more computation (linear in $|T|$ as shown in Appendix D.1) to perform task selection and (b) only a limited number (≤ 250) of tasks will be selected. We will investigate here whether the meta-test performance is sensitive to varying numbers of candidate tasks.

Tables 5.6 and 5.7 show results of Sinusoid regression and Omniglot classification, respectively. It can be observed that increasing the number of candidate tasks tends to improve the meta-test performance. However, when the number of candidate tasks becomes relatively large (respectively, 1000 and 2000 for Sinusoid and Omniglot), the improvement in meta-test performance by further increasing the number of candidate tasks is marginal. This empirically supports our argument that our active task selection

algorithm may not need such a massive number of candidate tasks, like in previous works [Chen et al., 2021, Finn et al., 2017, Finn et al., 2018, Yoon et al., 2018], in order to achieve reasonably competitive meta-test performance.

5.5.3 A Generalization to Adaptive Task Selection

This work considers the case of *non-adaptive* task selection s.t. k tasks are selected greedily (*one per iteration/round*) and their remaining datasets are acquired/observed only after all the k tasks are selected. For the case of **adaptive task selection** s.t. the remaining dataset $(X_t^r, \mathbf{y}_t^r)$ is acquired immediately after each task t is selected, an immediate improvement we can make to our algorithm is to additionally use this remaining dataset to update the forward particles to obtain a more accurate posterior belief of Θ in line 12 of Algorithm 1. As a result, the meta-test performance can be improved, as shown in Table 5.8. Note, however, that no performance guarantee is available for the adaptive case. Supposing the meta-learner's data need is time-critical (e.g., budget is available for a limited time) and each acquired remaining dataset can only be released after some time due to regulations, non-adaptive task selection may be preferred.

Table 5.8: Meta-test MSE or accuracy ($\%$) over 5 runs comparing adaptive vs. non-adaptive task selection.

	non-adaptive	adaptive
5-shot Sinusoid ($k = 100$)	0.430	**0.417**
10-shot Sinusoid ($k = 100$)	0.129	**0.124**
1-shot 5-way Omniglot ($k = 120$)	91.9	**93.2**
1-shot 20-way Omniglot ($k = 250$)	88.6	**89.2**
1-shot 5-way MiniImageNet ($k = 100$)	39.5	**41.1**

5.5.4 A Generalization to Larger Remaining Sets

We have also performed experiments on using a larger remaining dataset to evaluate the meta-test performance with varying ratios of remaining vs. sample dataset sizes from 1 to 20 given a fixed number R of data points in the remaining datasets of all selected tasks.

Table 5.9 shows that the meta-test performance degrades gracefully with an increasing ratio due to a reduced diversity of selected tasks. This reveals that in the setting of meta-learning, the meta-training task diversity plays an important role on the final meta-test performance.

Table 5.9: Meta-test MSE over 5 runs with varying ratios of remaining vs. sample dataset sizes given a fixed number R of data points in the remaining datasets of all selected tasks.

Ratio	1	5	8	10	20
5-shot Sinusoid ($R = 1000$)	**0.294**	0.315	0.349	0.375	0.398
10-shot Sinusoid ($R = 2000$)	**0.068**	0.070	0.070	0.108	0.129

5.6 Conclusion

This chapter describes a novel active task selection algorithm based on MILT for meta-learning with a near-optimal performance guarantee. A forward-backward method based on our proposed online SVGD (for learning/unlearning) is also designed to improve our efficiency. Empirical evaluations have demonstrated the state-of-the-art performance of our algorithm.

Chapter 6

Related Works

In this chapter, we give a review of related works for each of the four works included in this thesis, to elucidate the position of our contributions within the literature.

6.1 Recursive Reasoning-Based Training-Time Adversarial Machine Learning

Our *recursive reasoning-based training-time adversarial machine learning* (R2T2) framework introduces the concept of reasoning levels of the attacker and the defender, as illustrated in Fig. 6.1, where a level-k player best-responds to a level-$(k-1)$ opponent. During our introduction of the related works below, we show that our R2T2 frame-

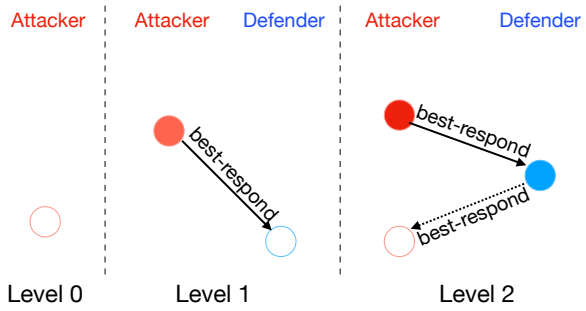

Figure 6.1: Illustration of how a level-$k = 0, 1, 2$ attack strategy is computed under the recursive reasoning model. A defense strategy of a different level is computed vice versa.

work encompasses a variety of existing adversarial ML methods which correspond to attackers/defenders with different reasoning levels.

Most works on adversarial *machine learning* (ML) have focused on the *test-time attacks* [Carlini and Wagner, 2017, Goodfellow et al., 2015, Madry et al., 2018, Zhang et al., 2019] where the attacker tries to fool an already trained ML model into an incorrect prediction by perturbing the test input. Several works aim at defending against such test-time attacks through data sanitization by correcting the perturbed test input [Li and Ji, 2019, Meng and Chen, 2017, Samangouei et al., 2018]. Note that all these works have studied either attacks or defenses solely. Their straightforward application to training-time attacks/defenses do not yield satisfactory performances, as demonstrated in our experiments (Chapter 2.5).

Another line of work called *adversarial training* [Szegedy et al., 2014] has attempted to perform test-time defense by including a training-time attacker during the training of the ML model. As a result, these attacks during training can act as a "vaccine" to ensure that the resulting model is robust against similar test-time attacks [Goodfellow et al., 2015, Kurakin et al., 2017, Madry et al., 2018, Singla and Feizi, 2020, Szegedy et al., 2014, Tramèr et al., 2018]. In adversarial training, the attacker is playing against a level-0 defender under our *Recursive Reasoning-based Training-Time adversarial ML* (R2T2) framework (Chapter 2.2.1).

Some other works have studied the training of an ML model under *dataset poisoning*, that is, by modifying the training set *before* the training starts and only allowing a few data points to be changed [Kearns and Li, 1993, Muñoz-González et al., 2017]. In comparison, we assume that the attacker can access and modify the minibatch data *during* model training. The dataset poisoning method of [Koh and Liang, 2017] makes use of influence function and can be considered a variant of our level-2 attack strategy (Chapter 2.3.3). A number of works have been proposed to defend against a dataset poisoning attack based on outlier removal [Barreno et al., 2010, Jagielski et al., 2018, Steinhardt et al., 2017], but they do not fit into our setting where an attack can corrupt all inputs of a minibatch.

A body of works called *backdoor attacks* [Bagdasaryan et al., 2020, Gu et al., 2019]

have tried injecting a backdoor into a model during training to produce wrong predictions if a specific trigger is added to an input at test time. Such attacks, which require the test set to be chosen/manipulated to be backdoored tasks, do not fit into our setting that does not allow modifying the test set. Backdoor defenses like backdoor removal [Wang et al., 2019] are ineffective in our setting where an attack does not inject any backdoor.

The work of [Dai et al., 2020] has proposed to use recursive reasoning in multi-agent Bayesian optimization but did not associate the level of recursive reasoning with any specific information (e.g., order of gradient in our case). This work has illustrated its application to test-time attacks, while we focus on training-time adversarial ML here.

Most relevant to our work here is that of [Feng et al., 2019] learning an auto-encoder-like network to generate training-time attacks, which essentially amounts to learning a variant of our level-1 attack strategy (Chapter 2.3.2).

Lastly, we show in Chapter 2.3.5 how our R2T2 framework encompasses a variety of existing adversarial ML methods which correspond to attackers/defenders with different reasoning levels.

6.2 Implicit Posterior Variational Inference for Deep Gaussian Processes

The *Gaussian process* (GP) [Rasmussen and Williams, 2006] can be composed hierarchically into a *deep GP* (DGP) model, which is proposed by [Damianou and Lawrence, 2013]. The DGP model usually exploits the notion of inducing variables [Quiñonero-Candela and Rasmussen, 2005] to improve its scalability to large datasets. The deterministic and stochastic approximation methods are two recent development for the inference of DGP. The former have imposed varying structural assumptions across the DGP hidden layers and assumed a Gaussian posterior belief of the inducing variables [Bui et al., 2016, Dai et al., 2016, Damianou and Lawrence, 2013, Hensman and Lawrence, 2014, Salimbeni and Deisenroth, 2017]. Particularly, to compute the posterior, the work of [Salimbeni and Deisenroth, 2017] has

proposed the use of the reparameterization trick [Kingma and Welling, 2013] and Monte Carlo sampling. The stochastic approximation methods like the work of [Havasi et al., 2018] has demonstrated that with at least one DGP hidden layer, the posterior belief of the inducing variables is usually non-Gaussian, hence potentially compromising the performance of the deterministic approximation methods due to their biased posterior belief. To resolve this, the stochastic approximation method of [Havasi et al., 2018] utilizes *stochastic gradient Hamiltonian Monte Carlo* (SGHMC) sampling to draw unbiased samples from the posterior belief in a computationally costly manner.

Regarding the advanced architecture design of the generators and the discriminators, there have been few literature trying to integrate them with DGP. Most relevant to our work is that of [Mirza and Osindero, 2014] coming up with the idea of using the known image label as the additional contextual input (for both generators and discriminators) in image generation. As a comparison, we are using the learnable inducing inputs as the contextual input to generate (or discriminate) the inducing output, which not only captures the intermediate dependency but also allows efficient joint optimization with DGP.

6.3 Meta-Learning with Implicit Processes

A number of meta-learning algorithms [Finn et al., 2018, Ravi and Beatson, 2018, Yoon et al., 2018] have proposed a Bayesian extension of the *model-agnostic meta-learning* (MAML) framework [Finn et al., 2017]. Their difference with our proposed *implicit process-based meta-learning* (IPML) is that they model the uncertainty in the predictions with a set of particles [Yoon et al., 2018] or a variational distribution [Finn et al., 2018, Ravi and Beatson, 2018], which does not allow latent task modeling, as explained before. The work of [Rusu et al., 2019] introduces a generative model that decodes latent vectors into the meta-parameters, but does not scale well in the dimension of meta-parameters. In comparison, IPML explicitly represents each task as a latent continuous vector and models its probabilistic belief and is hence scalable in the dimension

of meta-parameters. Moreover, MAML-based algorithms usually require evaluating computationally-intensive second-order derivatives of the meta-parameters during meta-training because they approximate the Bayesian inference through an inner loop of gradient descent. Although this issue can be addressed by methods such as first-order approximations (e.g., first-order MAML [Finn et al., 2017], Reptile [Nichol et al., 2018]) or implicit MAML [Rajeswaran et al., 2019] using implicit gradient, these works are not Bayesian. In contrast, our IPML algorithm naturally utilizes Bayes' rule to perform sampling during Bayesian inference and does not need second-order derivatives.

The work of [Kaddour et al., 2020] uses latent information to perform active task selection, but assumes known task-descriptor (task context) which is usually unknown. The work of [Garnelo et al., 2018b], which is the generalization of a previous work [Garnelo et al., 2018a], introduces the first use of stochastic processes (i.e., neural processes) in meta-learning and learns a heavily parameterized encoder to encode a dataset into its latent representation, which might introduce optimization difficulties and overfitting and can only output Gaussian posterior beliefs. The work of [Harrison et al., 2018] is a special case of our framework that considers only Bayesian linear regression. Under some assumptions, they perform tractable updates to obtain Gaussian posterior beliefs. In comparison, our IPML algorithm is the first to consider *stochastic gradient Hamiltonian Monte Carlo* (SGHMC) in task adaptation/inference of meta-learning, which can capture a non-Gaussian posterior belief to achieve a better performance (Appendix C.1.3). Our IPML algorithm is also the first to explicitly model task-dependent input distributions, which is lacking in the literature. Such a modeling enables synthetic task generation of complex image classification tasks for the first time.

6.4 Near-Optimal Task Selection with Mutual Information for Meta-Learning

Several works have aimed at combining active learning with meta-learning. The work of [Pang et al., 2018] has used meta-learning to learn the best active learning criterion for querying different datasets, which differs from our aim of using active learning to select tasks for meta-learning. The algorithms of [Finn et al., 2018, Yoon et al., 2018] have actively selected data points in each task but are not capable of selecting tasks directly. The greedy class-pair based sampling proposed in [Liu et al., 2020] and the probabilistic active meta-learning algorithm proposed in [Kaddour et al., 2020] have different problem settings. In comparison, our problem setting is more general. The work of [Luna and Leonetti, 2020] has proposed an information-theoretic task selection algorithm for meta-reinforcement learning, which assumes the availability of a validation set that can accurately represent the entire task distribution. Such a strong assumption contradicts the motivation of our active task selection problem.

Chapter 7

Conclusion, Limitations and Potential Future Work

This thesis has presented four pieces of works which have designed practical algorithms or network architectures to utilize the gradient information for effective optimization in deep learning scenarios. Each of the new algorithms or network architectures presented here has been shown to perform effectively in real-world experiments.

The training process of a machine learning (ML) model may be subject to adversarial attacks from an attacker who attempts to undermine the test performance of the ML model by perturbing the training minibatches, and thus needs to be protected by a defender. In Chapter 2, we describes the R2T2 framework which allows us to derive competitive level-k attack and defense strategies using gradient for the game of training-time adversarial ML. We empirically demonstrate that such strategies can achieve state-of-the-art performances on various benchmark image datasets. A limitation of this work is that we have only considered a myopic utility function. For our future work, we will investigate non-myopic approaches to solve this extensive-form game by considering a few steps lookahead. Another limitation is that the cost of computing a level-k strategy with our NPGD could be potentially high when k is large. Previous works have utilized neural networks to learn the attack and defense strategies. We plan to exploit similar ideas to learn all level-k strategies

with a recurrent neural network (RNN) to amortize the cost of computing the strategies such that a higher-level strategy is computed with more forward passes through the RNN.

Chapter 3 describes novel architecture designs for our proposed IPVI framework for DGPs. In our proposed IPVI framework, we cast the DGP inference problem as a two-player game and search for an unbiased posterior belief. However, due to the introduction of generators and discriminators into the DGP, the training becomes unstable and the test performances are unsatisfactory. We identified the optimization difficulties of the inducing inputs as the main reason for the unsatisfactory test performances. We thus propose a novel gradient-bridging architecture of the generator and discriminator in our IPVI framework for DGPs to alleviate the optimization difficulties and speed up training. Empirical evaluation on real-world datasets shows that IPVI with the gradient-bridging architecture outperforms the state-of-the-art parameter-tying architecture. A potential future improvement in the DGP inference is to tune the number of inducing points in an automatic way during model training, which address one limitation of this work that the number of inducing points is fixed (manually defined) before training.

The goal of few-shot learning (also known as *meta-learning*) is to leverage the experiences from previous tasks to form a model (represented by meta-parameters) that can rapidly adapt to a new task using only a limited quantity of its training data. However, it is not clear how or whether the recent meta-learning algorithms are naturally amenable to the characterization of a principled similarity/distance measure between tasks and various consequent applications. In Chapter 4, we describes a novel IPML algorithm that, in contrast to existing works, explicitly represents each task as a continuous latent vector and models its probabilistic belief within the highly expressive IP framework. Unlike existing works, IPML offers the benefits of being amenable to (a) the characterization of a principled distance measure between tasks using MMD, (b) active task selection without needing the assumption of known task contexts in [Kaddour et al., 2020], and (c) synthetic task generation of complicated image classifications via modeling of task-dependent input distributions using our X-Net. Empirical evaluation on benchmark datasets shows that

IPML outperforms existing Bayesian meta-learning algorithms. We have also empirically demonstrated on an anonymous e-commerce company's real-world dataset that IPML outperforms the multi-task learning baseline and identifies "outlier"/dissimilar tasks which can degrade meta-testing performance (Appendix C.1.2).

Finally, Chapter 5 describes a novel active task selection algorithm based on MILT for meta-learning with a near-optimal performance guarantee. A forward-backward method based on our proposed online SVGD is also designed to improve our efficiency. Empirical evaluation on several benchmark datasets have demonstrated the state-of-the-art performance of our algorithm. Chapter 5 successfully addresses the limitations of previous Chapter 4 by including a fully Bayesian treatment of the meta-parameters and proposing an improved active task selection algorithm with near-optimal performance guarantee. For future work, two potential improvements that can be investigated are: (i) the performance guarantee for the adaptive task selection such that the remaining dataset is acquired immediately after task t is selected instead of waiting for the budget of k tasks to be expended; (ii) the generalization to the case when different tasks (data) come with different costs, and we are given a budget of total cost instead of number of total tasks we can select.

Through the chapters, we have shown that gradient can play a series of different yet important roles in the modern machine learning problem. In Chapter 2, gradient first serves as a regularization which constraints the sophistication of the strategy associated with a player. Then the first-order gradient serves as a tool to efficiently and accurately approximate the optimal strategies in our proposed framework. In Chapter 3, allowing more efficient gradient back-propagation is the main idea on the design of the proposed model architecture. And in Chapter 3, 4, and 5, all the frameworks adopts implicitly represented distributions which can be efficiently optimized through gradient-based methods (gradient back-propagation of the model (meta) parameters, SGHMC of the posterior samples, and SVGD for the particles representing posterior distribution). Without using gradient, the problems would have been formulated differently and require other techniques, which may also bring extra difficulties. For example, without accessing the gradient information, the

strategy search in Chapter 2 becomes a zero-order black-box optimization problem. But the well-studied methods for such setting, like Bayesian optimization, could not scale to the high-dimensional input in the image datasets we considered. Without optimization of the gradient back-propagation architecture, the resulting model suffers from optimization difficulties as shown in Chapter 3. And in Chapter 4, without using gradient-based sampling method (SGHMC), the cost of running vanilla Markov Chain Monte Carlo sampling algorithms (e.g., Metropolis–Hastings algorithm) in such a high dimensional space of the latent vector could be formidable.

As we introduced in Chapter 1, many of the machine learning achievements are attributed to the back-propagation (BP) algorithm, which exploits gradient information of the deep neural network (DNN) models. For a long time, we see that the utilization of the gradient information are mainly considered by frequentists. However, this thesis investigates the exploitation of gradient information beyond the use of BP in conventional deep neural networks, and extends naturally to a series of Bayesian models/settings (including DGP, IP, and inference of intractable posteriors in various models). It is thus revealed that those area of studies prone to the Bayesian world can also benefits largely from exploitation of gradient information, which should not be considered sorely as a frequentist approach anymore. This thesis is not the end of the road in utilizing the gradient information, but shows possibilities to benefit from it in various aspects of machine learning.

Bibliography

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. In *Proc. OSDI*, pages 265–283.

[Bagdasaryan et al., 2020] Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., and Shmatikov, V. (2020). How to backdoor federated learning. In *Proc. AISTATS*, pages 2938–2948.

[Baluja and Fischer, 2018] Baluja, S. and Fischer, I. (2018). Learning to attack: Adversarial transformation networks. In *Proc. AAAI*, pages 2687–2695.

[Barreno et al., 2010] Barreno, M., Nelson, B., Joseph, A. D., and Tygar, J. D. (2010). The security of machine learning. *Machine Learning*, 81(2):121–148.

[Brooks et al., 2011] Brooks, S., Gelman, A., Jones, G., and Meng, X. (2011). *Handbook of Markov chain Monte Carlo*. CRC Press.

[Bui et al., 2016] Bui, T., Hernández-Lobato, D., Hernandez-Lobato, J., Li, Y., and Turner, R. (2016). Deep Gaussian processes for regression using approximate expectation propagation. In *Proc. ICML*, pages 1472–1481.

[Camerer et al., 2004] Camerer, C. F., Ho, T.-H., and Chong, J.-K. (2004). A cognitive hierarchy model of games. *Quarterly J. Economics*, 119(3):861–898.

[Carlini and Wagner, 2017] Carlini, N. and Wagner, D. (2017). Towards evaluating the robustness of neural networks. In *Proc. IEEE S&P*, pages 39–57.

[Caruana, 1997] Caruana, R. (1997). Multitask learning. *Machine learning*, 28(1):41–75.

[Chen et al., 2014] Chen, T., Fox, E., and Guestrin, C. (2014). Stochastic gradient Hamiltonian monte carlo. In *Proc. ICML*, pages 1683–1691.

[Chen et al., 2021] Chen, Y., Li, D., Li, N., Liang, T., Zhang, S., and Low, B. K. H. (2021). Meta-learning with implicit processes. `https://openreview.net/forum?id=m2ZxDprKYlO`.

[Dai et al., 2020] Dai, Z., Chen, Y., Low, B. K. H., Jaillet, P., and Ho, T.-H. (2020). R2-B2: Recursive reasoning-based Bayesian optimization for no-regret learning in games. In *Proc. ICML*, pages 2291–2301.

[Dai et al., 2016] Dai, Z., Damianou, A., González, J., and Lawrence, N. (2016). Variational auto-encoded deep Gaussian processes. In *Proc. ICLR*.

[Damianou and Lawrence, 2013] Damianou, A. and Lawrence, N. (2013). Deep Gaussian processes. In *Proc. AISTATS*, pages 207–215.

[Deisenroth and Ng, 2015] Deisenroth, M. P. and Ng, J. W. (2015). Distributed Gaussian processes. In *Proc. ICML*, pages 1481–1490.

[Duvenaud et al., 2014] Duvenaud, D., Rippel, O., Adams, R., and Ghahramani, Z. (2014). Avoiding pathologies in very deep networks. In *Proc. AISTATS*, pages 202–210.

[Eberhard and Paul, 2019] Eberhard, S. and Paul, R. (2019). A community-owned exchange for health information powered by blockchain technology. HIT Whitepaper version 4.01, Health Information Traceability Foundation.

[Feng et al., 2019] Feng, J., Cai, Q. Z., and Zhou, Z. H. (2019). Learning to confuse: Generating training time adversarial data with auto-encoder. In *Proc. NeurIPS*, pages 11971–11981.

[Finn et al., 2017] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proc. ICML*, pages 1126–1135.

[Finn et al., 2018] Finn, C., Xu, K., and Levine, S. (2018). Probabilistic model-agnostic meta-learning. In *Proc. NeurIPS*, pages 9516–9527.

[Gal et al., 2014] Gal, Y., van der Wilk, M., and Rasmussen, C. E. (2014). Distributed variational inference in sparse Gaussian process regression and latent variable models. In *Proc. NeurIPS*, pages 3257–3265.

[Garnelo et al., 2018a] Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. (2018a). Conditional neural processes. In *Proc. ICML*, pages 1704–1713.

[Garnelo et al., 2018b] Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. (2018b). Neural processes. arXiv:1807.01622.

[Gill and Prowse, 2016] Gill, D. and Prowse, V. (2016). Cognitive ability, character skills, and learning to play equilibrium: A level-k analysis. *J. Political Economy*, 124(6):1619–1676.

[Goodfellow et al., 2015] Goodfellow, I., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *Proc. ICLR*.

[Gordon et al., 2019] Gordon, J., Bronskill, J., Bauer, M., Nowozin, S., and Turner, R. E. (2019). Meta-learning probabilistic inference for prediction. In *Proc. ICLR*.

[Gretton et al., 2012] Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773.

[Gu et al., 2019] Gu, T., Liu, K., Dolan-Gavitt, B., and Garg, S. (2019). Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244.

[Harrison et al., 2018] Harrison, J., Sharma, A., and Pavone, M. (2018). Meta-learning priors for efficient online Bayesian regression. In *Proc. WAFR*, pages 318–337.

[Havasi et al., 2018] Havasi, M., Hernández-Lobato, J. M., and Murillo-Fuentes, J. J. (2018). Inference in deep Gaussian processes using stochastic gradient Hamiltonian Monte Carlo. In *Proc. NeurIPS*, pages 7517–7527.

[Hensman et al., 2013] Hensman, J., Fusi, N., and Lawrence, N. (2013). Gaussian processes for big data. In *Proc. UAI*, pages 282–290.

[Hensman and Lawrence, 2014] Hensman, J. and Lawrence, N. D. (2014). Nested variational compression in deep Gaussian processes. arXiv:1412.1370.

[Hernández-Lobato et al., 2011] Hernández-Lobato, D., Hernández-Lobato, J. M., and Dupont, P. (2011). Robust multi-class Gaussian process classification. In *Proc. NeurIPS*, pages 280–288.

[Ho et al., 1998] Ho, T.-H., Camerer, C., and Weigelt, K. (1998). Iterated dominance and iterated best response in experimental "p-beauty contests". *American Economic Review*, 88(4):947–969.

[Hoang et al., 2015] Hoang, T. N., Hoang, Q. M., and Low, K. H. (2015). A unifying framework of anytime sparse Gaussian process regression models with stochastic variational inference for big data. In *Proc. ICML*, pages 569–578.

[Hoang et al., 2016] Hoang, T. N., Hoang, Q. M., and Low, K. H. (2016). A distributed variational inference framework for unifying parallel sparse Gaussian process regression models. In *Proc. ICML*, pages 382–391.

[Jagielski et al., 2018] Jagielski, M., Oprea, A., Biggio, B., Liu, C., Nita-Rotaru, C., and Li, B. (2018). Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *Proc. IEEE S&P*, pages 19–35.

[Jerfel et al., 2019] Jerfel, G., Grant, E., Griffiths, T., and Heller, K. A. (2019). Reconciling meta-learning and continual learning with online mixtures of tasks. In *Proc. NeurIPS*, pages 9119–9130.

[Jin, 2018] Jin, Y. (2018). Does level-k behavior imply level-k thinking? Available at SSRN: https://ssrn.com/abstract=3138321.

[Jones, 2014] Jones, N. (2014). Computer science: The learning machines. *Nature News*, 505(7482):146.

[Kaddour et al., 2020] Kaddour, J., Sæmundsson, S., and Deisenroth, M. P. (2020). Probabilistic active meta-learning. In *Proc. NeurIPS*.

[Kearns and Li, 1993] Kearns, M. and Li, M. (1993). Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. In *Proc. ICLR*.

[Ko et al., 1995] Ko, C., Lee, J., and Queyranne, M. (1995). An exact algorithm for maximum entropy sampling. *Operations Research*, 43(4):684–691.

[Koh and Liang, 2017] Koh, P. W. and Liang, P. (2017). Understanding black-box predictions via influence functions. In *Proc. ICML*, pages 1885–1894.

[Krause and Golovin, 2014] Krause, A. and Golovin, D. (2014). Submodular function maximization. *Tractability*, 3:71–104.

[Krause et al., 2008] Krause, A., Singh, A., and Guestrin, C. (2008). Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *JMLR*, 9(8):235–284.

[Kurakin et al., 2017] Kurakin, A., Goodfellow, I., and Bengio, S. (2017). Adversarial machine learning at scale. In *Proc. ICLR*.

[Lake et al., 2011] Lake, B., Salakhutdinov, R., Gross, J., and Tenenbaum, J. (2011). One shot learning of simple visual concepts. In *Proc. CogSci*.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[Li et al., 2018] Li, B., Chen, C., Wang, W., and Carin, L. (2018). Second-order adversarial attack and certifiable robustness. arXiv:1809.03113.

[Li and Ji, 2019] Li, X. and Ji, S. (2019). Defense-VAE: A fast and accurate defense against adversarial attacks. In *Proc. ECML/PKDD*, pages 191–207.

[Liu et al., 2020] Liu, C., Wang, Z., Sahoo, D., Fang, Y., Zhang, K., and Hoi, S. (2020). Adaptive task sampling for meta-learning. In *Proc. ECCV*, pages 752–769.

[Liu and Wang, 2016] Liu, Q. and Wang, D. (2016). Stein variational gradient descent: A general purpose Bayesian inference algorithm. In *Proc. NeurIPS*, pages 2378–2386.

[Luna and Leonetti, 2020] Luna, G. R. and Leonetti, M. (2020). Information-theoretic task selection for meta-reinforcement learning. In *Proc. NeurIPS*.

[Ma et al., 2019] Ma, C., Li, Y., and Hernández-Lobato, J. M. (2019). Variational implicit processes. In *Proc. ICML*, pages 4222–4233.

[Madry et al., 2018] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. In *Proc. ICLR*.

[Matthews et al., 2017] Matthews, A. G. d. G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. (2017). GPflow: A Gaussian process library using TensorFlow. *JMLR*, 18:1–6.

[McMahan et al., 2013] McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., Chikkerur, S., Liu, D., Wattenberg, M., Hrafnkelsson, A. M., Boulos, T., and Kubica, J. (2013). Ad click prediction: A view from the trenches. In *Proc. KDD*, pages 1222–1230.

[Meng and Chen, 2017] Meng, D. and Chen, H. (2017). MagNet: A two-pronged defense against adversarial examples. In *Proc. CCS*, pages 135–147.

[Mirza and Osindero, 2014] Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. arXiv:1411.1784.

[Miyato et al., 2019] Miyato, T., Maeda, S., Koyama, M., and Ishii, S. (2019). Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(8):1979–1993.

[Muñoz-González et al., 2017] Muñoz-González, L., Biggio, B., Demontis, A., Paudice, A., Wongrassamee, V., Lupu, E. C., and Roli, F. (2017). Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proc. AISec*, pages 27–38.

[Nagel, 1995] Nagel, R. (1995). Unraveling in guessing games: An experimental study. *American Economic Review*, 85(5):1313–1326.

[Neal, 1993] Neal, R. M. (1993). Bayesian learning via stochastic dynamics. In *Proc. NeurIPS*, pages 475–482.

[Nesterov, 2018] Nesterov, Y. (2018). *Lectures on Convex Optimization*. Springer.

[Nguyen et al., 2020a] Nguyen, C., Do, T., and Carneiro, G. (2020a). Uncertainty in model-agnostic meta-learning using variational inference. In *Proc. WACV*, pages 3090–3100.

[Nguyen et al., 2020b] Nguyen, Q. P., Low, B. K. H., and Jaillet, P. (2020b). Variational Bayesian unlearning. In *Proc NeurIPS*.

[Nichol et al., 2018] Nichol, A., Achiam, J., and Schulman, J. (2018). On first-order meta-learning algorithms. arXiv:1803.02999.

[Pal and Vidal, 2020] Pal, A. and Vidal, R. (2020). A game theoretic analysis of additive adversarial attacks and defenses. In *Proc. NeurIPS*.

[Pang et al., 2018] Pang, K., Dong, M., and Hospedales, T. (2018). Meta-learning transferable active learning policies by deep reinforcement learning. `https://openreview.net/forum?id=HJ4IhxZAb`.

[Papernot et al., 2016] Papernot, N., McDaniel, P., and Goodfellow, I. (2016). Transferability in machine learning: From phenomena to black-box attacks using adversarial samples. arXiv:1605.07277.

[Qi, 2006] Qi, L. (2006). Rank and eigenvalues of a supersymmetric tensor, the multivariate homogeneous polynomial and the algebraic hypersurface it defines. *J. Symbolic Computation*, 41(12):1309–1327.

[Qi et al., 2018] Qi, L., Chen, H., and Chen, Y. (2018). *Tensor Eigenvalues and their Applications*. Springer.

[Quiñonero-Candela and Rasmussen, 2005] Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *JMLR*, 6:1939–1959.

[Rajeswaran et al., 2019] Rajeswaran, A., Finn, C., Kakade, S. M., and Levine, S. (2019). Meta-learning with implicit gradients. In *Proc. NeurIPS*, pages 113–124.

[Rasmussen and Williams, 2006] Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian processes for machine learning*. MIT Press.

[Ravi and Beatson, 2018] Ravi, S. and Beatson, A. (2018). Amortized Bayesian meta-learning. In *Proc. ICLR*.

[Ravi and Larochelle, 2017] Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. In *Proc. ICLR*.

[Rusu et al., 2019] Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. (2019). Meta-learning with latent embedding optimization. In *Proc. ICLR*.

[Salimbeni and Deisenroth, 2017] Salimbeni, H. and Deisenroth, M. (2017). Doubly stochastic variational inference for deep Gaussian processes. In *Proc. NeurIPS*, pages 4588–4599.

[Samangouei et al., 2018] Samangouei, P., Kabkab, M., and Chellappa, R. (2018). Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *Proc. ICLR*.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[Singla and Feizi, 2020] Singla, S. and Feizi, S. (2020). Second-order provable defenses against adversarial attacks. In *Proc. ICML*.

[Snell et al., 2017] Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In *Proc. NeurIPS*, pages 4077–4087.

[Sohn et al., 2015] Sohn, K., Lee, H., and Yan, X. (2015). Learning structured output representation using deep conditional generative models. In *Proc. NeurIPS*, pages 3483–3491.

[Springenberg et al., 2016] Springenberg, J. T., Klein, A., Falkner, S., and Hutter, F. (2016). Bayesian optimization with robust Bayesian neural networks. In *Proc. NeurIPS*, pages 4134–4142.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

[Stahl and Wilson, 1994] Stahl, D. O. and Wilson, P. W. (1994). Experimental evidence on players' models of other players. *J. Economic Behavior & Organization*, 25(3):309–327.

[Stahl and Wilson, 1995] Stahl, D. O. and Wilson, P. W. (1995). On players' models of other players: Theory and experimental evidence. *Games and Economic Behavior*, 10(1):218–254.

[Steinhardt et al., 2017] Steinhardt, J., Koh, P. W., and Liang, P. S. (2017). Certified defenses for data poisoning attacks. In *Proc. NeurIPS*, pages 3517–3529.

[Szegedy et al., 2014] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks. In *Proc. ICLR*.

[Titsias and Lázaro-Gredilla, 2014] Titsias, M. and Lázaro-Gredilla, M. (2014). Doubly stochastic variational Bayes for non-conjugate inference. In *Proc. ICML*, pages 1971–1979.

[Titsias, 2009a] Titsias, M. K. (2009a). Variational learning of inducing variables in sparse Gaussian processes. In *Proc. AISTATS*, pages 567–574.

[Titsias, 2009b] Titsias, M. K. (2009b). Variational model selection for sparse Gaussian process regression. Technical report, School of Computer Science, University of Manchester.

[Tramèr et al., 2018] Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. (2018). Ensemble adversarial training: Attacks and defenses. In *Proc. ICLR*.

[van der Maaten and Hinton, 2008] van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Proc. NeurIPS*, pages 5998–6008.

[Wang et al., 2019] Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., and Zhao, B. Y. (2019). Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *Proc. IEEE S&P*, pages 707–723.

[Wei and Tanner, 1990] Wei, G. C. and Tanner, M. A. (1990). A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association*, 85(411):699–704.

[Yin et al., 2020] Yin, M., Tucker, G., Zhou, M., Levine, S., and Finn, C. (2020). Meta-learning without memorization. In *Proc. ICLR*.

[Yoon et al., 2018] Yoon, J., Kim, T., Dia, O., Kim, S., Bengio, Y., and Ahn, S. (2018). Bayesian model-agnostic meta-learning. In *Proc. NeurIPS*, pages 7332–7342.

[Yu et al., 2019] Yu, H., Chen, Y., Low, B. K. H., Jaillet, P., and Dai, Z. (2019). Implicit posterior variational inference for deep gaussian processes. *Proc. NeurIPS*, 32.

[Zhang et al., 2019] Zhang, D., Zhang, T., Lu, Y., Zhu, Z., and Dong, B. (2019). You only propagate once: Accelerating adversarial training via maximal principle. In *Proc. NeurIPS*, pages 227–238.

Appendix A

Appendix for Chapter 2

A.1 Appendix: Supplementary Information and More Experimental Results

A.1.1 Additional Information for Experiments

A.1.1.1 Stabilizing Training with Momentum

As discovered by [Feng et al., 2019], a naive implementation of training-time attacks may keep the model training from convergence: In two consecutive iterations, the perturbations added by the attacker can be significantly different, which may cause dramatic differences in the modified data distributions, thus rendering the model training difficult. This is undesirable since the attacker's goal is to fool the target ML model into performing badly during test time while ensuring its convergence during training.

In order to tackle this instability during training (i.e., sometimes also observed in our experiments), we adopt a *momentum-based* variant of the attack strategy that keeps track of the attacker's own modification history of an input x_t as $\delta_{t-1}^H(x_t)$. The history starts from $\delta_0^H = 0$ and is updated when the attacker performs attacks; if an input is not in $\mathcal{D}_t^X$, then the history of this input will not be updated in iteration t. Then, the momentum-based

variant of the optimal level-k attack strategy is

$$\overline{\delta_t^{k^*}}(x_t) = \text{Proj}\left(\beta\ \delta_{t-1}^{H}(x_t) + (1-\beta)\ \delta_t^{k^*}(x_t)\right)$$

where $\beta \in [0, 1]$ is set to 0.95 in our experiments. We have observed in our experiments that with this momentum-based attack strategy, model training is significantly stabilized and thus always converges. Meanwhile, the defender does not follow such a momentum-based strategy.[1] Moreover, we have also observed that replacing SGD with more advanced momentum-based optimization techniques (e.g., Adam [Kingma and Ba, 2014]) also improves the stability of training. The experiments are thus conducted with Adam optimizers.

A.1.1.2 Network Architectures and Training Settings

For MNIST, the target model is a *convolutional neural network* (CNN) with two convolutional layers (of 16 and 64 channels) and a fully-connected layer (of 64 hidden units). For CIFAR-10, the target model is a CNN with three convolutional layers (of 128, 128, and 512 channels) along with an average pooling layer (3×3) and a fully-connected layer (of 256 hidden units). For ImageNet, the target model is a CNN with four convolutional layers (of 64, 64, 64, and 512 channels and corresponding strides of 4, 3, 2, 1) along with an average pooling layer (7×7) and a fully-connected layer (of 128 hidden units).

For defense-GAN, defense-VAE, and defense-CVAE, the decoders/generative models are CNNs of 2 layers (of 32 and 16 channels). The encoders are CNNs of 2 layers (of 16 and 32 channels). The dimensions of the latent variables in these models are set to be 20. We have implemented a defence-CVAE baseline because this improved baseline will not only use the information of clean inputs, but also use the information of the labels of the inputs, which is also given in the known public validation set $\mathcal{D}_{\text{val}}$. The implementation of DC uses an auto-encoder architecture which is the same as the auto-decoder of the

[1]The defender cannot follow such a momentum-based strategy because the attacks are adaptive in t. So, the defender cannot identify the same data inputs.

defense-VAE.

In Chapter 2.5.2, the subsampled MNIST dataset of digits 1 vs. 7 contains 40 training examples and 400 validation examples. The binary logistic regression classifier contains only a single linear layer. The CNN has two convolutional layers (of 16 and 16 channels) and a fully-connected layer (with 32 hidden units). We use such a small CNN because the calculation of Hessian is time-consuming if the target ML model has a large number of parameters.

The learning rate is 0.0008 for the Adam optimizer. The batch size is 500 for MNIST, 400 for CIFAR-10, and 100 for 2-class ImageNet. The test performances are recorded after 120 epochs for MNIST, 240 epochs for CIFAR-10, and 40 epochs for ImageNet.

A.1.2 More Experimental Results

A.1.2.1 More Analysis of Synthetic Experiment

From Theorem 2.1, the optimal R2T2 level-1 attack and defense strategies can be computed tractably as follows:

$$\delta_t^{1^*}(x_t) = \vec{\mathbf{e}}\left(-8\eta(\theta^* - \theta_t)^2\|z\|^2 x_t\right) = \vec{\mathbf{e}}\left(-x_t\right) \ ,$$
$$\sigma_t^{1^*}(x_t') = \vec{\mathbf{e}}\left(8\eta(\theta^* - \theta_t)^2\|z\|^2 x_t'\right) = \vec{\mathbf{e}}\left(x_t'\right) \ .$$

From Theorem 2.2, the optimal R2T2 level-2 attack and defense strategies can be computed tractably as ϵ approaches 0:

$$\lim_{\epsilon\to 0}\delta_t^{2^*}(x_t) = \vec{\mathbf{e}}\left(-8\eta(\theta^* - \theta_t)^2\|z\|^2(1-2\eta\|x_t\|^2)x_t\right) = \vec{\mathbf{e}}\left(-(1-2\eta\|x_t\|^2)x_t\right) \ ,$$
$$\lim_{\epsilon\to 0}\sigma_t^{2^*}(x_t') = \vec{\mathbf{e}}\left(8\eta(\theta^* - \theta_t)^2\|z\|^2(1-2\eta\|x_t'\|^2)x_t'\right) = \vec{\mathbf{e}}\left((1-2\eta\|x_t'\|^2)x_t'\right) \ .$$

Note that a perturbation along x_t will increase the model update step while a perturbation along $-x_t$ will decrease the model update step since $\|\theta_{t+1} - \theta_t\| \propto \|x_t''\|^2$. The difference from the level-1 players is that a level-2 player is aware of the case that a very large learning rate will cause the training to wiggle around the optimal parameter value. To see

this, observe that an additional term appears in the level-2 strategies: $(1 - 2\eta\|x'_t\|^2)$. If $1 < 2\eta\|x'_t\|^2$, then this term will reverse the direction of the perturbation. This condition corresponds exactly to the situation where the SGD step will "overshoot" the optimal parameters θ^* (Fig. 2.3b). So, the level-2 strategy differs from the level-1 strategy in that the level-2 defense strategy will correctly reduce such overshooting while the level-2 attack strategy will correctly encourage it.

A.1.2.1.1 Nash Equilibrium. Since

$$\mathcal{U}_t(x''_t) = \left| \left[\theta_t + 2\eta(\theta^* - \theta_t)\|x''_t\|^2\right] z - \theta^* z \right|^2$$

is a polynomial of $\|x''_t\|$,

$$\lim_{\epsilon\to 0} \delta_t^{2^*}(x_t) = \vec{\mathbf{e}}\left(-8\eta(\theta^* - \theta_t)^2\|z\|^2(1 - 2\eta\|x_t\|^2)x_t\right) ,$$
$$\lim_{\epsilon\to 0} \sigma_t^{2^*}(x'_t) = \vec{\mathbf{e}}\left(8\eta(\theta^* - \theta_t)^2\|z\|^2(1 - 2\eta\|x'_t\|^2)x'_t\right) .$$

Also, as ϵ approaches 0, $x'_t = x_t$. Then, from the above, we know that $\lim_{\epsilon\to 0} \sigma_t^{2^*}(x'_t) = \vec{\mathbf{e}}((1-2\eta\|x'_t\|^2)x'_t) = \vec{\mathbf{e}}((1-2\eta\|x_t\|^2)x_t)$ is independent of the attack strategy; the special case of $x_t = 0$ is omitted from discussion since it leads to the same outcome. Therefore, as ϵ approaches 0, the optimal strategies of both players are independent of their opponents' strategies. Thus, the above optimal strategies form a Nash equilibrium. Note that the Nash equilibrium strategies are equal to the optimal level-2 strategies, which indicates that Nash equilibrium is attained in the case of a level-2 attacker against a level-2 defender; nevertheless, only level-$k \geq 3$ players are able to recognize such a Nash equilibrium when they find that their level-$(k-1)$ opponents are also playing Nash equilibrium strategies. Such a Nash equilibrium implies that all level-k ($k \geq 2$) strategies equal to the level-2 strategies.

With regards to the experimental settings, we set $\eta = 0.2$ and $\epsilon = 0.1$, and sample minibatch input x_t (of minibatch size 1) uniformly from the interval $(0, 2]$. We set $\theta^* = 20$

and $\theta_0 = -10$. We consider a single fixed point $z = 50$ as (the input of) the validation set.

A.1.2.2 Visualization of Training-Time Adversarial Examples

Fig. A.1 shows the visualization. The way we visualize a perturbation (possibly with negative value in it) is to normalize it to $[0, 1]$ by the scale of difference between its minimum and maximum pixel values.

On MNIST, it can be observed that the defender tends to perform defenses by removing attacker's perturbations (e.g., first two rows of Fig. A.1a) or mending the attacker's removals (e.g., digit '3' in the last row of Fig. A.1a). On CIFAR-10, the attacks and defenses, although still effective, are harder to interpret as they are abstract and result in visually indistinguishable adversarial examples/perturbed inputs x'_t and transformed inputs x''_t (Fig. A.1b). On 2-class ImageNet, although the attacks and defenses result in indistinguishable adversarial examples/perturbed inputs x'_t and transformed inputs x''_t, it can be observed that the defender tends to perform defenses by offsetting the adversarial attacks as the color contrasts between the attacker's perturbations and the defender's transformations in the first three rows of Fig. A.1c (respectively, purple vs. green, skyblue vs. red, blue vs. orange) correspond to negations of pixel values.

A.1.2.3 Evaluation of Transferability: No Known Validation Set for the Attacker

We additionally examine a scenario when there is no known public validation set for the attacker. Note that the attacker can observe the clean minibatches and the attack strategy can be computed in a similar way by setting $\mathcal{D}_{\text{val}} = \mathcal{D}_t$ for the attacker.

The results in Table A.1 reveal a slight degradation in performance, possibly due to less training examples in a minibatch than in the validation set, hence yielding suboptimal attacks with large variances. However, the performance is not significantly different from that in the case of known validation set (Table 2.2), which means that the attacker may not need a known validation set to perform effective attacks.

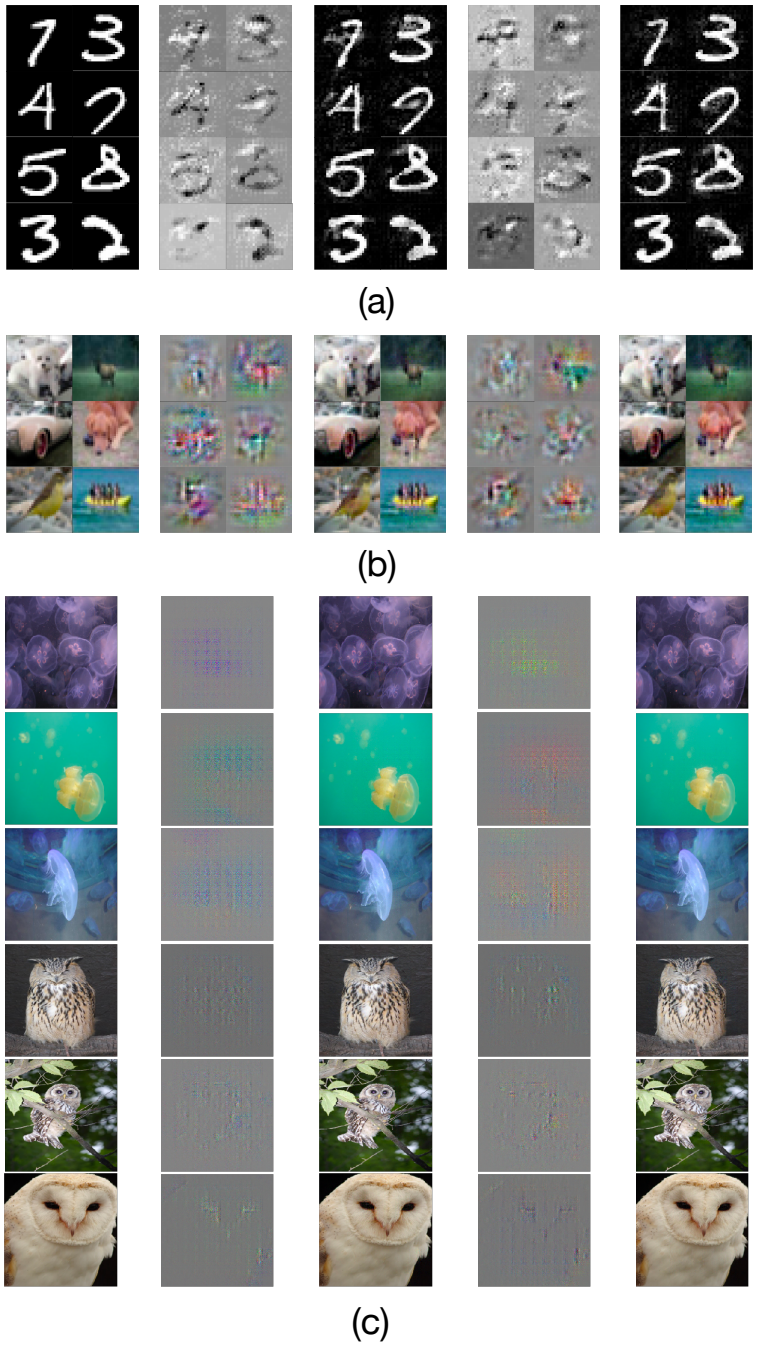

Figure A.1: From left to right are, respectively, visualizations of the original sampled images x_t, perturbations $\delta_t^{1^*}(x_t)$ from level-1 attacks, adversarial examples/perturbed inputs x'_t, transformations $\sigma_t^{1^*}(x'_t)$ from level-1 defenses, and finally the transformed inputs x''_t for model training on (a) MNIST, (b) CIFAR-10, and (c) ImageNet (color rescaled for viewing).

Table A.1: Mean test accuracy (%) $\pm$ 1 standard deviation (i.e., over 5 runs) for attacks (without a known validation set) against level-0 and level-1 defenders (with a known validation set).

	MNIST		CIFAR-10	
Attacker \ Defender	Level-0	Level-1	Level-0	Level-1
Level-1	14.0±4.6	96.0±1.2	18.9±1.7	50.1±7.2

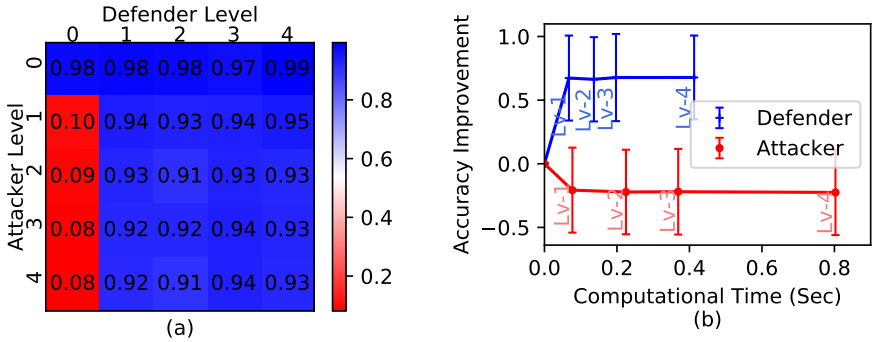

Figure A.2: (a) Mean test accuracy (%) on MNIST. (b) Graph of mean accuracy improvement (over null strategy) vs. time needed to compute the strategies given a minibatch of MNIST.

A.1.2.4 Level-k Strategies on MNIST

Fig. A.2a shows that when both players reason at level $k \geq 1$, the test accuracy is quite stable and does not change much with the players' reasoning levels. When both players reason beyond level $k \geq 1$, the change in the resulting test accuracy with respect to the players' reasoning levels is small. The marginal benefit of reasoning at a higher level vs. time needed to compute the higher-level strategy in Fig. A.2b is similar to that in Fig. 2.7b on CIFAR-10.

A.1.2.5 Level-k Strategies on 2-Class ImageNet

Fig. A.3a (i.e., same as Fig. 2.7c) shows that level-$k \geq 1$ defenses are quite effective. Interestingly, as analyzed in Chapter 2.5.3, even when defending against a level-0 attacker (i.e., no attack), higher-level defenses still improves test accuracy noticeably.

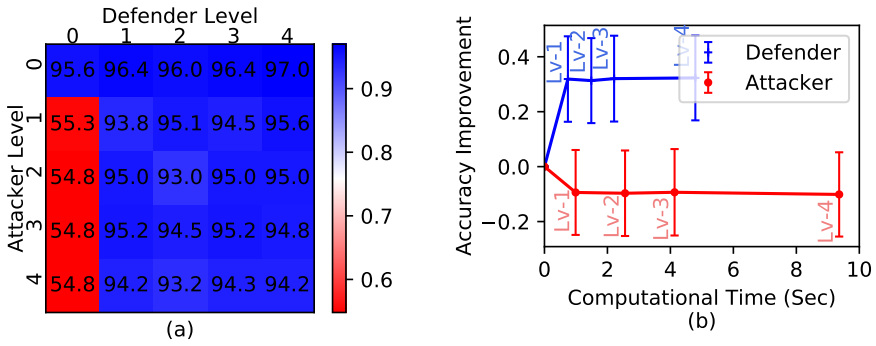

Figure A.3: (a) Mean test accuracy on 2-class ImageNet (i.e., same as Fig. 2.7c in Chapter 2.5.3). (b) Graph of mean accuracy improvement (over null strategy) vs. time needed to compute strategies given a minibatch of 2-class ImageNet.

A.2 Appendix: Proofs

A.2.1 Proof of Theorem 2.1

At level-1 the attacker best-responds to level-0 defender, with only first-order gradient information (Assumption 2.1). While assuming higher-order gradients $\nabla_\theta^k \sum_{z\in\mathcal{D}_{\text{val}}^X} \mathcal{L}_{\theta_t}(z)$ are all zero for $k \geq 1$, the attacker's evaluation of the goal following a Taylor-series expansion is $\mathcal{U}_t = (\theta_{t+1} - \theta_t)^\top \nabla_\theta \sum_{z\in\mathcal{D}_{\text{val}}^X} \mathcal{L}_{\theta_t}(z)$.

Note that for level-1 attacker, $\mathcal{L}_{\theta_t}(x)$ is a linear function of x (Assumption 2.1) and thus $\nabla_\theta \mathcal{L}_{\theta_t}(x)$ is also a linear function of x. The attacker then adopts strategy

$$\begin{aligned}
\delta_t^{1^*}(x_t) &= \underset{\delta_t(x_t):\|\delta(x_t)\|\leq 1}{\arg\max} \; -[\eta \nabla_\theta \mathcal{L}_{\theta_t}(x_t + \epsilon\, \delta_t(x_t) + \epsilon\, \sigma_t^{0^*}(x_t'))]^\top \sum_{z\in\mathcal{D}_{\text{val}}^X} [\nabla_\theta \mathcal{L}_{\theta_t}(z)] \\
&= \underset{\delta_t(x_t):\|\delta_t(x_t)\|\leq 1}{\arg\max} \; -[\eta \nabla_\theta \mathcal{L}_{\theta_t}(x_t + \epsilon\, \delta_t(x_t))]^\top [\sum_{z\in\mathcal{D}_{\text{val}}^X} \nabla_\theta \mathcal{L}_{\theta_t}(z)] \\
&= -\vec{\mathrm{e}} \left(\eta \nabla_x [\nabla_\theta \mathcal{L}_{\theta_t}(x_t)]^\top [\sum_{z\in\mathcal{D}_{\text{val}}^X} \nabla_\theta \mathcal{L}_{\theta_t}(z)] \right).
\end{aligned} \tag{A.1}$$

where $\vec{\mathrm{e}}(\cdot)$ means taking the unit vector. A similar reasoning on the defender gives the optimal defense strategy on the theorem. **End of the proof.**

Note that a level-0 strategy does not affect the functional form of level-1 strategy even if it is not null (e.g., $\sigma_t^{0^*}(x_t') \neq 0$): Assumption 2.1 has constrained that $\nabla_\theta \mathcal{L}_{\theta_t}(x_t)$ is a

linear function of x_t, thus for the attacker

$$\nabla_x \nabla_\theta \mathcal{L}_{\theta_t}(x_t + \epsilon\, \sigma_t^{0^*}(x_t')) = \nabla_x \nabla_\theta \mathcal{L}_{\theta_t}(x_t).$$

And vice versa for the defender. Consequently, level-0 strategy does not affect the functional form of level-k strategy for all $k \geq 1$.

A.2.2 Proof of Theorem 2.2

We first consider the attack strategy. Given Assumption 2.1, the second-order derivatives are available. We simplify the notation by denoting the optimal level-1 defense strategy $\sigma_t^{1^*}$ as σ, and denoting the optimal level-2 attack strategy $\delta_t^{2^*}$ as δ. $\delta(x_t)$ will result in the change of the stochastic gradient descent direction which can be expressed as a second order polynomial of the strategy $\delta(x_t)$,

$$\begin{aligned} \delta\theta &= -\eta \nabla_\theta [\mathcal{L}_{\theta_t}(x_t + \epsilon\, \delta(x_t) + \epsilon\, \sigma(x_t')) - \mathcal{L}_{\theta_t}(x_t + \epsilon\, \sigma(x_t))] \\ &= -\eta \nabla_\theta [\epsilon\, \delta(x_t)^\top \nabla_x \mathcal{L}_{\theta_t}(x_t)] + \mathcal{O}(\epsilon^2)\, \mathbf{u}_1, \end{aligned} \tag{A.2}$$

where $\mathbf{u}_1$ denotes a unit vector indicating the direction of residual term that we are not interested in. Suppose that we denote the domain of $\delta\theta$ as Θ ($\delta\theta \in \Theta$). As $\epsilon \to 0$, $\delta\theta$ is bounded and vary within a line segment Θ^0 where the domain Θ^0 is defined as

$$\Theta^0 \triangleq \{c\, \eta\epsilon \nabla_\theta \|\nabla_x \mathcal{L}_{\theta_t}(x_t)\|_2, \text{ where } c \in [-1, 1]\}.$$

Denote $\Delta\theta = -\eta \nabla_\theta \mathcal{L}_{\theta_t}(x_t + \epsilon\, \sigma(x_t))$. The optimal gradient descent direction for the attacker can then be obtained by solving a constraint quadratic optimization problem since

$\mathcal{L}_{\theta_{t+!}}(z)$ can be expanded as a quadratic polynomial of θ_{t+1} (Assumption 2.1):

$$\begin{aligned}\max_{\delta(x_t):\|\delta(x_t)\|_2\leq 1} \mathcal{U}_t &= \max_{\delta\theta\in\Theta} \mathcal{U}_t \\ &= \max_{\delta\theta\in\Theta} \sum_{z\in\mathcal{D}_{\text{val}}^X} [\mathcal{L}_{\theta_t+\Delta\theta+\delta\theta}(z) - \mathcal{L}_{\theta_t}(z)] \\ &= \max_{\delta\theta\in\Theta} \sum_{z\in\mathcal{D}_{\text{val}}^X} \left[\delta\theta^\top \nabla_\theta \mathcal{L}_{\theta_t}(z) + \frac{1}{2}(\delta\theta+\Delta\theta)^\top H_{\theta_t|z}(\delta\theta+\Delta\theta)\right],\end{aligned} \tag{A.3}$$

where $H_{\theta_t|z} \triangleq \nabla_\theta^2 \mathcal{L}_{\theta_t}(z)$, and $H_{\theta_t} \triangleq \sum_{z\in\mathcal{D}_{\text{val}}^X} H_{\theta_t|z}$. Given that Θ^0 is a linear constraint, we can transform the constrained quadratic optimization problem on $\delta\theta$ (A.3) into a constrained linear optimization problem on $\delta\theta$ as Θ approaches Θ^0 (ϵ approaches zero) since

$$\begin{aligned}&\lim_{\Theta\to\Theta^0} \arg\max_{\delta\theta\in\Theta} \sum_{z\in\mathcal{D}_{\text{val}}^X} [\delta\theta^\top \nabla_\theta \mathcal{L}_{\theta_t}(z) + \frac{1}{2}(\delta\theta+\Delta\theta)^\top H_{\theta_t|z}(\delta\theta+\Delta\theta)] \\ =& \lim_{\Theta\to\Theta^0} \arg\max_{\delta\theta\in\Theta} \delta\theta^\top \left\{ \left[(\sum_{z\in\mathcal{D}_{\text{val}}^X} H_{\theta_t|z})\Delta\theta + \sum_{z\in\mathcal{D}_{\text{val}}^X} [\nabla_\theta \mathcal{L}_{\theta_t}(z)] \right] + \mathcal{O}(\|\delta\theta\|^2) \right\} \\ =& \lim_{\Theta\to\Theta^0} \arg\max_{\delta\theta\in\Theta} \delta\theta^\top \left[(\sum_{z\in\mathcal{D}_{\text{val}}^X} H_{\theta_t|z})\Delta\theta + \sum_{z\in\mathcal{D}_{\text{val}}^X} [\nabla_\theta \mathcal{L}_{\theta_t}(z)] \right] \\ =& \lim_{\Theta\to\Theta^0} \arg\max_{\delta\theta\in\Theta} \delta\theta^\top \mathcal{G}(x_t + \epsilon\,\sigma(x_t), \theta_t)\end{aligned} \tag{A.4}$$

where we define the function

$$\mathcal{G}(x, \theta_t) \triangleq -\eta(\sum_{z\in\mathcal{D}_{\text{val}}^X} H_{\theta_t|z}) \nabla_\theta \mathcal{L}_{\theta_t}(x) + \sum_{z\in\mathcal{D}_{\text{val}}^X} [\nabla_\theta \mathcal{L}_{\theta_t}(z)] \tag{A.5}$$

for simplicity of notations. Note that $\delta\theta$ is a second-order polynomial on $\delta(x_t)$ (A.2), the optimization problem (A.4) can now be turned into a quadratic constrained quadratic programming on $\delta(x_t)$. Now the attacker can solve for its optimal strategy following (A.2)

and (A.4),

$$
\begin{aligned}
&\lim_{\epsilon\to 0} \delta_t^{2^*}(x_t) \\
=&\lim_{\epsilon\to 0} \underset{\delta(x_t):\|\delta(x_t)\|\leq 1}{\arg\max} \Big(-\eta\epsilon\ \delta(x_t)^\top [\nabla_x \nabla_\theta \mathcal{L}_{\theta_t}(x_t)]^\top + \mathcal{O}(\epsilon^2)\ \mathbf{u}_1^\top \Big)\ \mathcal{G}(x_t + \epsilon\ \sigma(x_t), \theta_t) \\
=&\lim_{\epsilon\to 0} \underset{\delta(x_t):\|\delta(x_t)\|\leq 1}{\arg\min} \delta(x_t)^\top [\nabla_x \nabla_\theta \mathcal{L}_{\theta_t}(x_t)]^\top \mathcal{G}(x_t + \epsilon\ \sigma(x_t), \theta_t) \\
=&\underset{\delta(x_t):\|\delta(x_t)\|\leq 1}{\arg\min} \delta(x_t)^\top [\nabla_x \nabla_\theta \mathcal{L}_{\theta_t}(x_t)]^\top \mathcal{G}(x_t, \theta_t) \\
=&-\vec{\mathbf{e}}([\nabla_x \nabla_\theta \mathcal{L}_{\theta_t}(x_t)]^\top \mathcal{G}(x_t, \theta_t)).
\end{aligned}
\tag{A.6}
$$

A similar reasoning on the defender side gives the optimal defense strategy on the theorem.

End of the proof.

A.2.2.1 Proof of Corollary 2.1

Following the above proof step (A.3), $\sum_{z\in\mathcal{D}_{\text{val}}^X} \mathcal{L}_{\theta_t+\Delta\theta+\delta\theta}(z)$ can be expanded as a second-order polynomial of θ and is convex under the assumption of positive definite H_{θ_t}.

Note that OC attack solves

$$
\begin{aligned}
&\lim_{\epsilon\to 0} \min_{\delta(x_t), \|\delta(x_t)\|_2\leq 1} \|\theta_{t+1} - \theta^\star\| \\
&= \lim_{\epsilon\to 0} \min_{\delta\theta\in\Theta} \|\theta_{t+1} - \theta^\star\| \\
&= \lim_{\Theta\to\Theta^0} \min_{\delta\theta\in\Theta} \|\theta_{t+1} - \theta^\star\| \\
&= \min_{\delta\theta\in\Theta^0} \|\theta_{t+1} - \theta^\star\| \\
&= \max_{\delta\theta\in\Theta^0} \delta\theta^\top [\theta_{t+1} - \theta^\star] \\
&= \max_{\delta\theta\in\Theta^0} \delta\theta^\top \Big[H_{\theta_t}^{-1} \nabla_\theta [\sum_{z\in\mathcal{D}_{\text{val}}^X} \mathcal{L}_{\theta_t+\Delta\theta}(z)] \Big] \\
&= \max_{\delta\theta\in\Theta^0} \delta\theta^\top \Big[H_{\theta_t}^{-1} \mathcal{G}(x_t + \epsilon\ \sigma(x_t), \theta_t) \Big],
\end{aligned}
\tag{A.7}
$$

So, the final solution is

$$\begin{aligned}
&\lim_{\epsilon\to 0} \operatorname*{arg\,min}_{\delta(x_t):\|\delta(x_t)\|_2\leq 1} \|\theta_{t+1}-\theta^\star\| \\
=&\lim_{\epsilon\to 0} \operatorname*{arg\,max}_{\delta(x_t):\|\delta(x_t)\|_2\leq 1} \Big(-\eta\epsilon\, \delta(x_t)^\top [\nabla_x \nabla_\theta \mathcal{L}_{\theta_t}(x_t)]^\top + \mathcal{O}(\epsilon^2)\, \mathbf{u}_1^\top \Big)\, H_{\theta_t}^{-1} \mathcal{G}(x_t+\epsilon\,\sigma(x_t),\theta_t) \\
=&-\vec{\mathbf{e}}([\nabla_x \nabla_\theta \mathcal{L}_{\theta_t}(x_t)]^\top H_{\theta_t}^{-1} \mathcal{G}(x_t,\theta_t))
\end{aligned} \tag{A.8}$$

End of the proof.

A.2.3 Proof of Theorem 2.3

Note that for the problem of interest, input x varies within a closed domain bounded by the original input space and the perturbation constraint. Without loss of generality, we assume each dimension of x varies within $[0,1]$ (which is satisfied if we are dealing with normalized image input, otherwise, we normalize the bounded input space to $[0,1]$ and the analysis below will only differ by a normalizing constant).

Suppose the dimension of x is N. We denote the h-th order derivatives as $\mathcal{H}_{x_t}^h \triangleq \nabla_x^h \mathcal{U}_t(x_t)$, which is a h-th order N dimensional real *supersymmetric* tensor [Qi et al., 2018] for $h \geq 2$.

Lemma A.1. *There exists a constant M such that under any level-k reasoning, the largest Z-eigenvalue[2] of $\mathcal{H}_x^h$ ($h \leq k$) is not more than $M^h \frac{k!}{(k-h)!}$ for x in a closed domain $[0,1]^N$.*

Proof of Lemma A.1: First note that the Z-eigenvalue always exists [Qi et al., 2018]. Under any finite level of reasoning, the Taylor expansion of $\mathcal{U}_t(x)$ on $x_0 = \mathbf{0}$ can be expressed as finite polynomial on $x_{\{1\}}, \ldots, x_{\{N\}}$ where the bracket denotes the dimensions of x.

$$\mathcal{U}_t(x) = \sum_{v_1,\ldots,v_N} C_{v_1,\ldots,v_N} x_{\{1\}}^{v_1} \ldots x_{\{N\}}^{v_N}.$$

[2]Refer to [Qi et al., 2018] for detailed definition of Z-eigenvalue.

We then denote $C_1 \triangleq \sum_{v_1,\ldots,v_N} |C_{v_1,\ldots,v_N}|$. Under level-$k$ reasoning, the loss function is a k-th order polynomial of x (Assumption 2.1). As a result, the Taylor expansion of $\mathcal{U}_t(x)$ on $x = 0$ is cut off in order k (a k-th order polynomial), and the above summation over $v_1, \ldots, v_N$ is constrained by $v_1 + \ldots + v_N \leq k$. Note that the maximum absolute value of $x_{\{1\}}, \ldots, x_{\{N\}}$ is < 1 in such closed domain, thus the scale of every element in the supersymmetric tensor $\mathcal{H}_x^h$ is bounded by $C_1 \frac{k!}{(k-h)!}$. The supersymmetric tensor $\mathcal{H}_x^h$ contains N^h elements, the Frobenius norm of $\mathcal{H}_x^h$ is thus bounded: $\|\mathcal{H}_x^h\|_F \leq C_1 \frac{k!}{(k-h)!} N^{h/2}$. Since the largest absolute value of Z-eigenvalues of $\mathcal{H}_x^h$ is bounded by the Frobenius norm $\|\mathcal{H}_x^h\|_F$ (Theorem 9 in [Qi, 2006]), choose $M = C_1 N^{\frac{1}{2}}$ we can arrive at the result. **End of the proof**.

Lemma A.2. *Under level-k reasoning, function $\mathcal{U}_t(x)$ is L-smooth with respect to x in the closed domain bounded by the perturbation constraint $X_t'' = \{x : \|x - x_t'\|_2 \leq \epsilon\}$ where $L = M^2 k^2 (1 + 2M\epsilon)^{k-2}$ for some M.*

Proof of Lemma A.2: Note that the following optimization problem

$$\max_{\Delta, \|\Delta\|_2 \leq 1} A \odot \Delta^h, \text{ where } A \odot x^h \triangleq \sum_{i_1,\ldots,i_h=1}^{N} a_{i_1,\ldots,i_h} x_{\{i_1\}} \cdots x_{\{i_h\}}, \tag{A.9}$$

where A is a h-th order N dimensional real supersymmetric tensor, has solution equals to the largest Z-eigenvalue of A [Qi et al., 2018]. Since $\forall \{x_1, x_1 + \Delta\} \in X_t''$ we have (note that this implies $\|\Delta\| < \varepsilon \triangleq 2\epsilon$):

$$\begin{aligned}
&\max_{\{x_1, x_1+\Delta\} \in X_t''} \frac{2}{\|\Delta\|^2} \left[\mathcal{U}_t(x_1 + \Delta) - \mathcal{U}_t(x_1) - (\nabla_x \mathcal{U}_t(x_1))^\top \Delta\right] \\
&\leq \frac{2}{\|\Delta\|^2} \max_{\{x_1, x_1+\Delta\} \in X_t''} \sum_{h=2}^{k} \frac{1}{h!} \mathcal{H}_{x_1}^h \odot \Delta^h \\
&\leq \sum_{h=2}^{k} \frac{2\, \varepsilon^{h-2} k!}{(k-h)! h!} M^h \\
&\leq M^2 k^2 (1 + 2M\epsilon)^{k-2},
\end{aligned} \tag{A.10}$$

where the first inequality follows from Taylor expansion under level-k reasoning, the

second inequality follows from Lemma A.1, the third inequality holds since $\varepsilon \leq 2\epsilon$ in the closed domain. The above inequality indicates that $\mathcal{U}_t(x)$ is L-smooth in the closed domain bounded by the perturbation constraint (X_t'') where $L = M^2k^2(1+2M\epsilon)^{k-2}$, according to the definition of L-smooth. **End of the proof**.

Lemma A.3. *For a convex function $\mathcal{U}$ that is L-smooth, the iterates given by the projected gradient descent with step size $\Gamma = 1/L$ starting from x_0 satisfy for every step i*

$$|\mathcal{U}(x_{[i]}) - \mathcal{U}(x^*)| \leq (2L/i)\|x_0 - x^*\|^2. \tag{A.11}$$

If $\mathcal{U}$ is further μ-strongly convex, we have

$$\|x_{[i]} - x^*\|^2 \leq (1-\mu/L)^i\|x_0 - x^*\|^2. \tag{A.12}$$

Proof of Lemma A.3: The lemma is a direct result in [Nesterov, 2018]. The proof is given in [Nesterov, 2018]. **End of the proof**.

Proof of the main theorem. Given Lemma A.2 and Lemma A.3, we know that

$$|\mathcal{U}_t(x' + \epsilon\,\sigma_{[i]}) - \mathcal{U}_t(x_t' + \epsilon\,\sigma^{k^*}(x'))| \leq (2L\epsilon^2/i)\|\sigma_{[0]} - \sigma_t^{k^*}(x_t')\|^2 \leq 2\epsilon^2/(\Gamma_k i). \tag{A.13}$$

If $\mathcal{U}_t(x)$ is $\mu-$strongly convex, then

$$\|\sigma_{[i]} - \sigma_t^{k^*}(x_t')\|^2 \leq \frac{1}{\epsilon^2}(1-\mu/L)^j\epsilon^2\|\sigma_{[0]} - \sigma_t^{k^*}(x_t')\|^2 \leq (1-\Gamma_k\mu)^j \leq \exp(-\mu\Gamma_k j). \tag{A.14}$$

End of the proof.

A.2.3.1 Attacker PGD: The Naive Version

For the attacker, we define an auxiliary function $\mathcal{U}_t^\star(x) \triangleq \mathcal{U}_t\big(x + \epsilon\,\sigma_t^{(k-1)^*}(x)\big)$.

Corollary A.1. ***(Convergence of naive attacker PGD)*** *Assume that $-\mathcal{U}_t^\star(x)$ is a convex function of x and suppose the 1-st to k-th gradients of $\sigma_t^{(k-1)^*}(x)$ with respect to x is finite. Then there exists a constant M, such that by setting the step size $\Gamma_k \triangleq 1/[M^2k^2(1+2M\epsilon)^{k-2}]$, to approximate level-$k$ strategy for input x_t, the PGD of $\delta_{[i]}$ starting from PGD step $i=0$ with $\delta_{[0]}=0$:*

$$\overline{\delta}_{[i+1]} = \text{Proj}\big(\delta_{[i]} + (\Gamma_k/\epsilon)\nabla_x \mathcal{U}_t^\star(x'_{[i]})\big) \tag{A.15}$$

where $x'_{[i]} \triangleq x_t + \epsilon\,\delta_{[i]}$, has following convergence guarantee in any PGD step i:

$$\big|\mathcal{U}_t^\star\big(x_t+\epsilon\,\delta_{[i]}\big) - \mathcal{U}_t^\star\big(x_t+\epsilon\,\delta_t^{k^*}(x_t)\big)\big| \le 2\epsilon^2/(\Gamma_k i). \tag{A.16}$$

If $-\mathcal{U}_t^\star(x)$ is further $\mu-$strongly convex, we have

$$\big\|\delta_{[i]} - \delta_t^{k^*}(x_t)\big\| \le \exp(-\mu\Gamma_k i). \tag{A.17}$$

Its proof is similar to the proof of Theorem 2.3. **End of the proof.**

Convergence of attacker's NPGD: The naive attacker PGD (A.15) cannot be computed in practice because we cannot compute $\sigma_t^{(k-1)^*}$ tractably. However, if $\sigma_{[i]}$ in the attacker NPGD (2.9) approximates $\sigma_t^{(k-1)^*}(x'_{[i]})$ perfectly without error in every PGD step i, the NPGD is then identical to the naive attacker PGD, thus $\delta_{[i]}$ of the NPGD enjoy the convergence guarantee on Corollary A.1 above.

A.2.3.2 Estimate M and Γ_k

To estimate M from training, note that from the above proof of Theorem 2.3 we can see that $N^{\frac{1}{2}}|\mathcal{U}_t([1,\ldots,1])| = N^{\frac{1}{2}}|\sum_{v_1,\ldots,v_N} C_{v_1,\ldots,v_N}| \le C_1 N^{\frac{1}{2}} = M$ provides a lower bound of M. Although in theory this lower bound could be loose, we found that estimating M with this lower bound works well in practice, such that we can always observe improvements

when reasoning into a higher-level. To calculate this lower bound, we train the ML model on a white image ($[1, \ldots, 1]$) for 1 iteration and calculate the change of validation loss. As a result, the estimation of M is roughly 0.203, 0.351, 1.04 for MNIST, CIFAR-10 and 2-class ImageNet respectively.

The step size Γ_k can thus be calculated correspondingly. Note that in practice it is always the case that $(\Gamma_k/\epsilon)|\nabla_x \mathcal{U}_t(x)| > 1$, thus we can safely set the step size to positive infinity and the resulting attacker NPGD/ defender PGD still performs well. To estimate a reasonable number of total PGD steps, we first ensure that approximating level-k strategies at least needs to perform k steps of PGD. We then set the total number of steps i as $2\Gamma_k$, $0.5\Gamma_k$ and $0.01\Gamma_k$ for MNIST, CIFAR-10 and 2-class ImageNet respectively, with the estimation of M above.

A.2.4 Generalize To More Than One Example in $\mathcal{D}_t$

We extend the notation of $\mathcal{U}_t$ such that $\mathcal{D}_t^{''} \triangleq \{[x_{jt}'', y(x_{jt})] : j = 1, \ldots\}$ is the minibatch after the perturbations of the attacker and the defender, $\mathcal{D}_t^{''X} \triangleq \{[x_{jt}''] : j = 1, \ldots\}$ is the collection of the inputs of $\mathcal{D}_t^{''}$, and $\mathcal{U}_t(\mathcal{D}_t^{''X})$ represents the utility function if current training in conducted on the (perturbed) minibatch $\mathcal{D}_t^{''}$; and $\mathcal{U}_t(x_t^{''})$ still represents the utility function if current training in conducted on the (perturbed) single data input x_t''. As a result, the base game can be written as:

$$\begin{aligned} \textbf{Attacker:} \quad & \max_{\delta_t(x_t), \|\delta_t(x_t)\| \leq 1} \mathcal{U}_t(\mathcal{D}_t^{''X}), \\ \textbf{Defender:} \quad & \max_{\sigma_t(x_t'), \|\sigma_t(x_t')\| \leq 1} -\mathcal{U}_t(\mathcal{D}_t^{''X}). \end{aligned} \tag{A.18}$$

Suppose now the minibatch data $\mathcal{D}_t$ can contain more than one training examples, such that $\mathcal{D}_t^X = \{[x_{jt}] : j = 1, 2, \ldots\}$. We have

$$\begin{aligned}\frac{\partial \mathcal{U}_t(\mathcal{D}_t''^X)}{\partial x_j} =& -\eta[\frac{\partial}{\partial x_j}\nabla_\theta \sum_{x''_{lt}\in\mathcal{D}_t''^X} \mathcal{L}_{\theta_t}(x''_{lt})]^\top \sum_{z\in\mathcal{D}_{\text{val}}^X}[\nabla_\theta \mathcal{L}_{\theta_t}(z)] \\ =& -\eta[\frac{\partial}{\partial x_j}\nabla_\theta \mathcal{L}_{\theta_t}(x''_{jt})]^\top \sum_{z\in\mathcal{D}_{\text{val}}^X}[\nabla_\theta \mathcal{L}_{\theta_t}(z)] \\ =& \nabla_x \mathcal{U}_t(x''_{jt}),\end{aligned} \tag{A.19}$$

where the last line follows from (2.7).

A.2.4.1 Level-1 strategies

Due to (A.19), follow the proof in A.2.1 we can show that the level-1 strategies still follow (2.6) and remain unchanged.

A.2.4.2 Level-2 strategies

Because of (A.19), it is not hard to follow the proof in A.2.2 and show that the level-2 strategies in the limit of small ϵ should be changed to below.

Corollary A.2. ***(Level-2 strategies)*** *Given a training minibatch $\mathcal{D}_t^X = \{[x_{jt}] : j = 0, 1, \ldots\}$, the optimal level-2 attack and defense strategies in the limit of small ϵ are*

$$\begin{aligned}\lim_{\epsilon\to 0}\delta_t^{2^*}(x_t) &= -\vec{\mathbf{e}}\big([\nabla_x\nabla_\theta \mathcal{L}_{\theta_t}(x_t)]^\top\big[\sum_{z\in\mathcal{D}_{val}^X}\nabla_\theta \mathcal{L}_{\theta_t}(z) - \eta H_{\theta_t}\nabla_\theta \sum_{x_{jt}\in\mathcal{D}_t^X}\mathcal{L}_{\theta_t}(x_{jt})\big]\big), \\ \lim_{\epsilon\to 0}\sigma_t^{2^*}(x'_t) &= \vec{\mathbf{e}}\big([\nabla_x\nabla_\theta \mathcal{L}_{\theta_t}(x'_t)]^\top\big[\sum_{z\in\mathcal{D}_{val}^X}\nabla_\theta \mathcal{L}_{\theta_t}(z) - \eta H_{\theta_t}\nabla_\theta \sum_{x'_{jt}\in\mathcal{D}'^X_t}\mathcal{L}_{\theta_t}(x'_{jt})\big]\big).\end{aligned} \tag{A.20}$$

A.2.4.3 Level-k strategies

Because of (A.19), both defender PGD (2.10) and attacker NPGD (2.9) remain unchanged.

Appendix B

Appendix for Chapter 3

B.1 Proof of Nash Equilibrium Coincides with the Global Maximizer of the ELBO

If $(\{\theta_D{}^*\}, \{\theta^*, \theta_G{}^*\})$ is a Nash equilibrium, then under the assumption that $T_{\theta_D{}^*}$ is expressive enough, we know that **Player 1** is playing its optimal strategy $\theta_D{}^*$ such that

$$T_{\theta_D{}^*}(\mathbf{U}) = \log q_{\theta_G{}^*}(\mathbf{U}) - \log p(\mathbf{U}) \; . \tag{B.1}$$

Substituting (B.1) into (3.5) reveals that **Player 2**'s strategy $\{\theta^*, \theta_G{}^*\}$ maximizes its payoff which is a function of $\{\theta, \theta_G\}$:

$$\begin{aligned} \mathcal{F}(\theta, \theta_G) \triangleq \; & \mathbb{E}_{q_{\theta_G}(\mathbf{U})}[\mathcal{L}(\theta, \mathbf{X}, \mathbf{y}, \mathbf{U}) + \log p(\mathbf{U}) - \log q_{\theta_G{}^*}(\mathbf{U})] \\ = \; & \mathbb{E}_{q_{\theta_G}(\mathbf{U})}[\mathcal{L}(\theta, \mathbf{X}, \mathbf{y}, \mathbf{U}) + \log p(\mathbf{U}) - \log q_{\theta_G}(\mathbf{U}) + \log q_{\theta_G}(\mathbf{U}) - \log q_{\theta_G{}^*}(\mathbf{U})] \\ = \; & \mathcal{EL}(\theta, \theta_G) + \mathrm{KL}[q_{\theta_G}(\mathbf{U}) \| q_{\theta_G{}^*}(\mathbf{U})] \end{aligned} \tag{B.2}$$

where $\mathcal{EL}(\theta, \theta_G)$ is the ELBO in (3.3).

Now, suppose that $\{\theta^*, \theta_G{}^*\}$ does not maximize the ELBO. Then, there exists some

$\{\theta', {\theta_G}'\}$ such that $\mathcal{EL}(\theta', {\theta_G}') > \mathcal{EL}(\theta^*, {\theta_G}^*)$. By substituting $\{\theta', {\theta_G}'\}$ into (B.2),

$$\mathcal{F}(\theta', {\theta_G}') = \mathcal{EL}(\theta', {\theta_G}') + \mathrm{KL}[q_{\theta_G{}'}(\mathbf{U}) \| q_{\theta_G{}^*}(\mathbf{U})] > \mathcal{F}(\theta^*, {\theta_G}^*) \ ,$$

which contradicts the fact that $\{\theta^*, {\theta_G}^*\}$ maximizes (B.2). Hence, $\{\theta^*, {\theta_G}^*\}$ maximizes the ELBO, which is equal to the log-marginal likelihood $\log p_{\theta^*}(\mathbf{y})$ with θ^* being the maximum likelihood assignment and $q_{\theta_G{}^*}(\mathbf{U})$ being equal to the true posterior belief $p(\mathbf{U}|\mathbf{y})$.

B.1.1 Discussion on the Existence of Nash Equilibrium

Statement: Suppose that the parametric representations of T_{θ_D} and g_{θ_G} are expressive enough to represent any function and the DGP model hyperparameters are fixed to be $\theta_\circ$. Then, the two-player pure-strategy game in (3.6) for the case of fixed $\theta_\circ$ has a Nash equilibrium. Furthermore, if $(\{{\theta_D}^*\}, \{\theta_\circ, {\theta_G}^*\})$ is a Nash equilibrium, then $\{{\theta_G}^*\}$ is a global maximizer of the ELBO for the case of fixed $\theta_\circ$ such that $q_{\theta_G{}^*}(\mathbf{U})$ is equal to the true posterior belief $p_{\theta_\circ}(\mathbf{U}|\mathbf{y})$.

Proof. Since we assume the parametric representation of g_{θ_G} to be expressive enough to represent any function, we can find some $\{\theta_{G\circ}\}$ such that $q_{\theta_{G\circ}}(\mathbf{U})$ is equal to the true posterior belief $p_{\theta_\circ}(\mathbf{U}|\mathbf{y})$. We now know that $\{\theta_{G\circ}\}$ maximizes the ELBO in (3.3) for the case of fixed DGP model hyperparameters $\theta_\circ$, which we denote by $\mathcal{EL}(\theta_\circ, \theta_{G\circ})$.

Since we assume the parametric representation of T_{θ_D} to be expressive enough to represent any function, we can further obtain some $\{\theta_{D\circ}\}$ such that $T_{\theta_{D\circ}}(\mathbf{U}) = \log q_{\theta_{G\circ}}(\mathbf{U}) - \log p(\mathbf{U})$. As a result, $\{\theta_{D\circ}\}$ maximizes the payoff to **player 1**. Hence, **player 1** cannot improve its strategy to achieve a better payoff.

Given that **player 1** plays strategy $\{\theta_{D\circ}\}$ for the case of fixed $\theta_\circ$, the payoff to **player 2**

playing strategy $\{\theta_\circ, \theta_G\}$ is

$$\begin{aligned}
\mathcal{F}(\theta_\circ, \theta_G) \triangleq\ & \mathbb{E}_{q_{\theta_G}(\mathbf{U})}[\mathcal{L}(\theta_\circ, \mathbf{X}, \mathbf{y}, \mathbf{U}) + \log p(\mathbf{U}) - \log q_{\theta_{G\circ}}(\mathbf{U})] \\
=\ & \mathbb{E}_{q_{\theta_G}(\mathbf{U})}[\mathcal{L}(\theta_\circ, \mathbf{X}, \mathbf{y}, \mathbf{U}) + \log p(\mathbf{U}) - \log q_{\theta_G}(\mathbf{U}) + \log q_{\theta_G}(\mathbf{U}) - \log q_{\theta_{G\circ}}(\mathbf{U})] \\
=\ & \mathcal{EL}(\theta_\circ, \theta_G) + \mathrm{KL}[q_{\theta_G}(\mathbf{U}) \| q_{\theta_{G\circ}}(\mathbf{U})] \\
=\ & \log p_{\theta_\circ}(\mathbf{y}) - \mathrm{KL}[q_{\theta_G}(\mathbf{U}) \| p_{\theta_\circ}(\mathbf{U}|\mathbf{y})] + \mathrm{KL}[q_{\theta_G}(\mathbf{U}) \| q_{\theta_{G\circ}}(\mathbf{U})] \\
=\ & \log p_{\theta_\circ}(\mathbf{y}) \,.
\end{aligned}$$

So, **player 2** receives a constant payoff (i.e., independent of $\{\theta_G, \theta_\circ\}$) and cannot improve its strategy to achieve a better payoff. Since every player cannot improve strategy to achieve a better payoff, $(\{\theta_{D\circ}\}, \{\theta_\circ, \theta_{G\circ}\})$ is a Nash Equilibrium.

The rest of the proof is similar to that above. □

B.1.2 Regression and Classification Details

In this subsection, we provide additional details for our experiments in the supervised learning tasks.

Learning Rates. We adopt the default settings of the learning rates of the tested methods from their publicly available implementations. The learning rates and maximum iteration for IPVI are tuned through grid search and cross validation with a default setting of $\alpha_{\theta_D} = 0.05$, $\alpha_{\theta_G} = 0.001$, $\alpha_\theta = 0.025$ and cut-off at a maximum of 20000 iterations. The learning rates for classification is simply set to be 0.02 for all parameters.

Hidden Dimensions. The dimension of inducing variables for all implementations are set to be (i) the same as input dimension for the UCI benchmark regression and Airline datasets, (ii) 16 for the YearMSD dataset, and (iii) 98 for the classification tasks.

Mini-Batch Sizes. The mini-batch sizes for all implementations are set to be (i) 10000 for the UCI benchmark regression tasks, (ii) 20000 for the large-scale regression tasks, and (iii) 256 for the classification tasks.

Mean Function of DGP. The 'skip-layer' connections are implemented in both

SGHMC [Havasi et al., 2018] and DSVI [Salimbeni and Deisenroth, 2017] for DGPs and in our IPVI framework as well. The work of [Duvenaud et al., 2014] has analyzed that using a zero mean function in the DGP prior causes some difficulty as each GP mapping is highly non-injective. To mitigate this issue, the work of [Salimbeni and Deisenroth, 2017] has proposed to include a linear mean function $m(\mathbf{X}) = \mathbf{W}\mathbf{X}$ for all hidden layers. The 'skip-layer' connection $\mathbf{W}$ is set to be an identity matrix if the input dimension equals to the output dimension. Otherwise, $\mathbf{W}$ is computed from the top H eigenvectors of the data under SVD. We follow the same setting as this 'skip-layer' mean function. Note that this 'skip-layer' mean function contains no trainable parameters.

Likelihood. For the classification tasks, we use the robust-max multiclass likelihood [Hernández-Lobato et al., 2011] (see Chapter 3.1.1). Tricks like data augmentation are not applied, which means that the accuracy can still be improved further with those additional tricks.

Appendix C

Appendix for Chapter 4

C.1 Appendix: Additional Details, Experimental Settings, Results, and Analysis

C.1.1 Details of Experimental Settings

C.1.1.1 Sampler Hyperparameters

In practice, running the SGHMC sampling process has two phases: The first is the burn-in phase used to determine the suitable sampler hyperparameters, while the second is the sampling phase which is run using the fixed sampler hyperparameters. In the burn-in phase, the hyperparameters of the sampler are selected using a heuristic auto-tuning approach following that of [Springenberg et al., 2016] with initial value of $\epsilon = 0.03$, $C = 0.05$, and $M = \mathbf{I}$.

Since consecutive samples are highly correlated, we do not choose consecutive samples to perform the M step to prevent the risk of overfitting to those samples. Instead, we randomly select one sample of $\mathbf{z}$ per task from the sampling phase to carry out optimization in the M step. In the sampling phase, we acquire a total of 40 samples for sinusoid regression, and 5 samples for Omniglot and mini-Imagenet classification.

C.1.1.2 Meta-training Settings

For sinusoid regression, we train for a total of 15000 iterations with an Adam optimizer of meta-learning rate (i.e., learning rate of the outer gradient step) 0.001 and meta-batch size 25. For Omniglot and mini-Imagenet classification, we train for a total of 60000 iterations with an Adam optimizer of meta-learning rate 0.001, and meta-batch size is 16 and 4 for Omniglot and mini-Imagenet, respectively. Gradient clipping of ± 10 is applied in both the SGHMC step and the outer gradient step.

Although IPML does not require any inner gradient step, we have found that adding such inner optimization further improves performance. Thus, in our implementation, we perform inner gradient optimization of the whole model's parameter after E step, and the M step will differentiate through those inner gradient steps (only in a first-order manner). Note that this does not violate our analysis in Chapter 4.3.1. Our analysis on the E step is accurate given the current best estimation of the meta-parameter θ. However, after knowing the support set, we are able to optimize our current estimation of θ. The inner gradient optimization corresponds to such a step of refining our estimation of θ and can be interpreted as an inner M step nested in the outer E step. We perform 5 inner gradient steps by default.

C.1.1.3 Active Task Selection Settings

For sinusoid, we consider 25 meta-tasks per iteration and IPML will choose 1 of them according to our proposed active task selection criterion (i.e., variance in samples of $\mathbf{z}$). For mini-Imagenet, we consider 16 meta-tasks per iteration and IPML will choose 1 of them. IPML will store these selected tasks in a buffer and train on them with meta-batch size of 5 and 4 for sinusoid and mini-Imagenet, respectively. Without active task selection, the model will just train on all tasks received.

Motivating active task selection with a real-world use case. In our real-world risk detection experiment (Chapter 4.4), a considerable fraction of black and white labels (i.e., risk or no risk) are labeled manually due to the rapidly evolving nature of risks or fast

emergence of new variants of risks. However, for the anonymous e-commerce company, the labeling budget is limited (e.g., a human can only label a fixed amount of items per day). This will require us to select the best sequence of tasks to be labeled to best improve the model. Such problem is crucial and is essentially an active task selection problem that can be tackled using our proposed IPML.

C.1.2 Experiments on an Anonymous E-Commerce Company's Risk Detection Dataset

In an anonymous company's online e-shop, the items advertised by the sellers may contain risks (e.g., fraud, pornography, contraband). Such risks appear in different forms and in different categories (of items). It is hard to detect risks in different categories by training models separately for each category because some categories have only very limited amounts of black samples (i.e., < 50). The similarities of the detected risks in different categories, if discovered, can help improve the performance and the model can quickly adapt to detect risks on new categories of items with only few-shot data given. Meta-learning is thus a suitable algorithm for its ability to perform (a) detectionof risks across different categories of items and (b) adaptation to new categories.

Data preparation and training settings. We first perform self-supervised learning on the texts of an item using all the raw data available (including items with risk labels or without risk labels). The texts we use are concatenations of an item's title and descriptions. This allows us to obtain an embedding for each item, which has a dimension of 32768. The embedding will serve as the input $\mathbf{x}$ while its label is a binary variable indicating whether it contains risks ($y_{\mathbf{x}} = 1$ for black samples and $y_{\mathbf{x}} = 0$ for white samples). The data containing all items are separated by categories of items yielding 47 categories in total. Initially, 10 held-out categories are used for meta-testing (see Table C.1) while the rest are used for meta-training. Some of the categories have nearly 200000 items while some only have less than 100 items. The fractions of black samples are typically small (average of

14% across 47 categories). We train with a meta-batch size of 10 (each subsampled from a category) and a batch size of 32 (16 support items and 16 query items). During training, we ensure that a task in a minibatch should contain > 2 black samples; otherwise, we skip that task.

Table C.1: Ten held-out categories for meta-testing.

Category ID	Category Name
3	Personal nursing and Cosmetics
9	Antique collection
19	Domestic and Daily-use
21	Cellular
23	Costume and accessories
36	Network equipment
39	Watches and glasses
44	In-game currencies
46	Gaming accounts
47	Gaming items

We use a 2-layer neural network with the first layer having 1024 hidden neurons, and the second layer outputs a binary classification. A leaky ReLU activation of slope 0.1 is used in the hidden layer. We choose a multi-task learning baseline for performance comparison, which explicitly ties the parameters on the first layer of neural networks, and extends a separate second layer for binary classification (i.e., one for each category). When testing on an unseen category, multi-task learning performs adaptation by randomly initializing its second layer's parameters for the new category (while tying the first layer's parameters) and optimizing them on the few-shot support data. For both methods, we train with a learning rate of 0.01 for a total of 8000 iterations. At 8000 iterations, we have observed that the mean meta-training accuracy of IPML on 37 categories is stabilized at around 96.33%. In meta-testing for a particular category, we sampled 200 items (150 white samples and 50 black samples) and split them into a support set of size 100 and a query set of size 100 in every cross-validation trial. The results are averaged over 2 cross-validation trials.

More discussion on the latent task representation. As can be seen from Tables C.2

Table C.2: Meta-testing performance on 10 held-out meta-testing categories for IPML.

Category ID	Accuracy (%)	F1	Recall	Precision
3	85.3	76.1	88.8	66.6
9	83.0	73.8	88.8	63.1
19	90.2	81.4	88.0	75.8
21	94.5	81.7	86.0	77.8
23	89.0	81.3	88.9	75.0
36	79.4	63.1	66.6	60.0
39	93.6	85.7	100.0	75.0
44	84.0	71.4	80.0	64.5
46	72.1	43.9	40.7	47.8
47	74.0	45.8	40.8	52.3
Average	84.5	70.5	76.8	65.8

and C.3, IPML outperforms multi-task learning, which indicates its stronger ability to generalize to unseen categories. Fig. 4.5 visualizes the latent task embedding of the 10 meta-testing categories for analysis. IPML learns useful latent task representations: For example, from Fig. 4.5a, gaming-related categories with IDs 46 and 47 are mapped closely in the latent task space/embedding, which coincides with the fact that they both represent risks on categories involving gaming. Another category with ID 44, which is also gaming-related, is mapped closest to them among the rest of 8 categories. IDs 21, 23, and 36 are also mapped closely in the latent space, but a clear interpretation cannot be obtained as they represent quite different categories (i.e., cellular, costume and accessories, and network equipment). Although the white samples of these 3 categories may be considerably dissimilar, we have found that their black samples bear a striking similarity after digging into their raw text (title and descriptions): There is a huge fraction of overlap between their black samples because certain types of malicious sellers (e.g., on fraud and forging of certificates and diplomas) are more likely to advertise their duplicated risk items (black samples) on these three categories, while they are less likely to advertise on other categories such as gaming (e.g., IDs 46 and 47). This shows that our latent task representation and embedding can aid our understanding and instruct us to identify categories that are influenced more by a certain type of malicious sellers.

More results on learning with or without the "outlier"/dissimilar tasks. From

Table C.3: Meta-testing performance on 10 held-out meta-testing categories for multi-task learning baseline.

Category ID	Accuracy (%)	F1	Recall	Precision
3	90.2	79.1	70.3	90.4
9	87.0	75.4	74.4	76.9
19	82.2	57.1	48.1	70.5
21	87.3	69.7	60.3	83.3
23	78.7	55.9	51.8	60.8
36	80.0	52.3	40.7	73.3
39	93.6	83.3	83.3	83.3
44	79.0	53.3	48.2	60.0
46	78.1	38.8	25.9	77.7
47	77.5	41.0	29.6	66.6
Average	84.1	60.5	54.1	73.9

our analysis in Chapter 4.4, we conjecture that when a meta-learning model is trained to perform well (during meta-testing) on the desired categories/tasks with IDs 19, 21, 23, 36, and 44 which are shown to be similar to previous meta-training tasks, training alongside with dissimilar ones (i.e., with IDs 3, 9, 39, 46, and 47) can compromise its performance. To verify our conjecture, we compare meta-learning on (A) the same setting as before by holding out the 10 meta-testing categories vs. (B) training on all categories in setting A as well as the dissimilar ones with IDs 3, 9, 39, 46, and 47. Table C.4 shows detailed results on the individual meta-testing performance on the 10 meta-testing categories. The experimental results agree with our conjecture: By looking at the F1 score, meta-training that additionally includes dissimilar tasks/categories results in degradation of performance in 4 of the 5 desired categories. Thus, to achieve competitive meta-testing performance, one should consider not to include dissimilar tasks during meta-training.

C.1.3 Ablation Study of the Effectiveness of IPML Components

This subsection empirically evaluates the effectiveness of each component of the IPML algorithm.

Table C.4: Meta-testing performance on desired categories with IDs 19, 21, 23, 36, and 44 trained without (setting A) or with (setting B) dissimilar tasks.

Category ID	Accuracy (%)		F1		Recall		Precision	
Setting	A	B	A	B	A	B	A	B
19	**90.2**	89.0	**81.4**	79.9	88.0	88.0	**75.8**	73.3
21	**94.5**	90.4	**81.7**	80.7	**86.0**	84.0	**77.8**	77.7
23	89.0	**90.1**	81.3	**83.2**	88.9	**92.5**	75.0	**75.7**
36	**79.4**	76.5	**63.1**	58.6	**66.6**	62.9	**60.0**	54.8
44	84.0	**85.0**	**71.4**	69.3	**80.0**	68.0	64.5	**70.8**
Average	**87.4**	86.4	**75.8**	74.4	**81.9**	79.8	**70.6**	70.5

Table C.5: Comparison between IPML-VI vs. IPML on benchmark meta-learning datasets.

	Sinusoid 10-shot (MSE)	Omniglot 1-shot 5-way (Accuracy %)	Omniglot 1-shot 20-way (Accuracy %)	mini-Imagenet 1-shot 5-way (Accuracy %)	mini-Imagenet 5-shot 5-way (Accuracy %)
IPML-VI	0.160	98.7	94.3	50.2	66.6
IPML	0.123	98.8	94.0	50.5	67.6

C.1.3.1 SGHMC vs. VI and Amortized VI

To obtain the posterior samples from $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$, we use SGHMC. Other methods exist to obtain posterior samples from $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$. The first method is *variational inference* (VI) that directly constructs a variational distribution for $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$. The resulting approach, which we call *IPML with VI* (IPML-VI), replaces SGHMC with VI in our IPML algorithm while keeping its other components unchanged. Unfortunately, such a variational approximation usually yields a biased latent task posterior belief while SGHMC can ideally recover (samples from) the true latent task posterior belief $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$. It can be observed from our visualization of the latent task embeddings in Figs. A.1 and 4.4b that the distribution of a cluster of tasks is highly non-Gaussian. We thus conjecture that even for a single task, its distribution is not likely to be Gaussian. So, IPML-VI is limited in its ability to recover the true latent task posterior belief $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$. This is empirically supported by the results in Table C.5 showing that our IPML algorithm with SGHMC outperforms IPML-VI in 4 of the 5 test cases.

The other method that can obtain posterior samples from $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$ is amortized VI which is used by the *neural process* (NP) [Garnelo et al., 2018b]: It learns an encoder that

can take in the support set and output the variational distribution for $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$. As analyzed in Chapter 4.4, NP utilizing amortized VI performs unsatisfactorily as compared to IPML for both sinusoid regression (Table 4.1) and Omniglot (Table 4.2), likely because (a) it employs a heavily parameterized encoder which may introduce optimization difficulties and overfitting during meta-training, and (b) the encoder of NP takes in the simple concatenation of $(\mathbf{x}, y_\mathbf{x})$ and thus does not explicitly capture the $\mathbf{x} \rightarrow y_\mathbf{x}$ relationship in the support set. An ablation study of the limitations of NP is presented in Appendix C.1.7.

C.1.3.2 EM Algorithm vs. Variational Gaussian Process Framework

Table C.6: Few-shot classification accuracy (%) on held-out Omniglot characters.

	Omniglot 1-shot 5-way	Omniglot 5-shot 5-way
VGPML	62.7	80.4
IPML	98.8	99.5

Using SGHMC, our EM algorithm can directly maximize the original meta-learning objective $\mathcal{J}_{\text{meta}}$ (D.1). In contrast, the VI framework directly models $p(\mathbf{f}_{\mathcal{X}_t^q}|\mathbf{y}_{\mathcal{X}_t^s})$ and maximizes the variational/*evidence lower bound* (ELBO) of $\mathcal{J}_{\text{meta}}$ (D.1). Such a framework typically assumes f to be distributed by a *Gaussian process* (GP) [Ma et al., 2019] instead of an IP (Def. 5.1) and we call the resulting framework *variational GP-based meta-learning* (VGPML). Its flexibility is then largely constrained by the GP and its chosen kernel. Table C.6 shows that VGPML using the widely used radial basis function kernel performs unsatisfactorily as compared to IPML on few-shot classification, hence reflecting the effectiveness of our proposed EM algorithm for IPML over VGPML.

C.1.3.3 Coupling of Latent Task Vector $\mathbf{z}$ with DNN Generator g_θ

Table C.7: *Mean square error* (MSE) on few-shot sinusoid regression.

	Sinusoid 5-shot	Sinusoid 10-shot
Concatenation	0.817	0.525
Soft mask	0.373	0.123

As mentioned in Chapter 4.3.2, the design of the coupling of $\mathbf{z}$ with the DNN $g_\theta(\mathbf{x}, \cdot)$ is crucial to achieving competitive performance of our IPML algorithm. Table C.7 shows results comparing (a) the naive design of concatenating $\mathbf{z}$ with $\mathbf{x}$ as a contextual input during forward passes vs. (b) our delicate design of applying $\mathbf{z}$ as a continuous-valued (soft) mask to the last DNN layer's parameters (Chapter 4.3.2), the latter of which is used by our IPML algorithm to produce the experimental results in Chapter 4.4. It can be observed from Table C.7 that our delicate design using $\mathbf{z}$ as a soft mask outperforms the naive design concatenating $\mathbf{z}$ with $\mathbf{x}$ by a considerable margin.

C.1.4 Doubly-Contextual X-Net

In Chapter 4.3.3, we have introduced X-Net that can generate both synthetic regression or classification tasks. For the more specific case of balanced N-way classification tasks, we have prior knowledge that $y_\mathbf{x}$ is uniformly distributed over $[1, \ldots, N]$ in a sampled task. Thus, when performing synthetic task generation, our X-Net can be additionally conditioned on $y_\mathbf{x}$ uniformly sampled from $[1, \ldots, N]$ when generating $\mathbf{x}$. We refer to such a design as *doubly-contextual X-generative network* (DC X-Net) since it now takes in both $\mathbf{z}$ and $y_\mathbf{x}$ as contexts. Note that with DC X-Net, we no longer need the IP to generate labels as the labels are sampled in the first place. Although there is now no explicit forward pass in IP during synthetic task generation, this does not make the DC X-Net an independent model of IP. On the contrary, since $\mathbf{z}$ (i.e., samples obtained from IP) is taken in as a context during training, the learning of DC X-Net can be interpreted as a knowledge distillation process of the IP.

Now, let the DC X-Net: $\mathbf{x} \triangleq h_\phi(y_\mathbf{x}, \mathbf{z}, \boldsymbol{\omega})$ learn to generate an input vector $\mathbf{x}$ given a sample of the latent task vector $\mathbf{z}$, label $y_\mathbf{x}$ (i.e., converted into one-hot encoding), and a sample of the random vector $\boldsymbol{\omega} \sim p(\boldsymbol{\omega}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ where $\boldsymbol{\omega}$ models the diversity of the input distribution given a fixed class $y_\mathbf{x}$ in a fixed task represented by the sample of $\mathbf{z}$. As before, we use (the decoder of) CVAE to implement DC X-Net. The training objective for synthetic task generation is then the empirical lower bound [Sohn et al., 2015] of VI on

$p(\boldsymbol{\omega}|\mathbf{x}, y, \mathbf{z})$:

$$\mathcal{J}_{\mathrm{X}} \triangleq \sum_{t\in\mathcal{T}} \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})} \left[|\mathcal{X}_t|^{-1} \sum_{\mathbf{x}\in\mathcal{X}_t} (\mathbb{E}_{q_\psi(\boldsymbol{\omega}|\mathbf{x},y_\mathbf{x},\mathbf{z})}[\log p_\phi(\mathbf{x}|y_\mathbf{x},\mathbf{z},\boldsymbol{\omega})] - D_{\mathrm{KL}}[q_\psi(\boldsymbol{\omega}|\mathbf{x},y_\mathbf{x},\mathbf{z})\|p(\boldsymbol{\omega})]) \right]$$

where ϕ is the parameter of the DC X-Net (decoder neural network), ψ is the parameter of the encoder neural network, and D_{KL} denotes the KL distance. In the training of DC X-Net, we sample one $\mathbf{z}$ per update. We also sample one $\boldsymbol{\omega}$ per update to train with reparameterization tricks.

C.1.5 Evaluation of Distance Measure between Tasks for Settings B to E in Chapter 4.4

For setting A, the subsampled Omniglot contains 15 characters, each of which has around 20 images. For settings B to E, the subsampled mini-Imagenet contains 50 classes. For each setting, we evaluate the distance measure between different types of tasks using *maximum mean discrepancy* (MMD) metric with radial basis function kernels on the $\mathbf{z}$ samples, and show the results in Table C.8. By comparing with Fig. A.1, the distances evaluated using the MMD metric are in accordance with our visualization of the latent task embeddings.

To illustrate that the latent task representation produced by IPML captures the semantic difference between tasks instead of the input-level modification, we consider the latent input representation learned by an ordinary classifier which has the same architecture as g_θ. To compute the latent input embedding of a task, we average the last layer's outputs over all data in this task. Fig. C.1 shows a TSNE visualization of latent input embeddings from settings A to E based on the ordinary classifier. A comparison of Fig. C.1 vs. Fig. A.1 shows that except for setting C (i.e., different types of hue), the ordinary classifier cannot produce a latent input representation that captures the semantic difference between tasks. In contrast, IPML is capable of producing a latent task representation that captures such

Table C.8: Values of MMD metric between different types of tasks for mini-Imagenet (settings B to E). Larger value means larger dissimilarity.

Setting B	brightness -0.5	normal	brightness $+0.5$
brightness -0.5	0	0.69	2.02
normal	0.69	0	1.26
brightness $+0.5$	2.02	1.26	0

Setting C	hue1 (red)	hue2 (green)	hue3 (blue)
hue1 (red)	0	0.70	1.97
hue2 (green)	0.70	0	1.69
hue3 (blue)	1.97	1.69	0

Setting D	original	$3\times$zoom-in	$10\times$zoom-in
original	0	0.58	2.06
$3\times$zoom-in	0.58	0	1.48
$10\times$zoom-in	2.06	1.48	0

Setting E	contrast=$1/10$	normal	contrast=10
contrast=$1/10$	0	0.73	0.95
normal	0.73	0	0.15
contrast=10	0.95	0.15	0

semantic difference between tasks. Worth mentioning, the latent task representation produced by NP encoder is similar to that of Fig. C.1 and failed to adequately capture the semantic difference between tasks. The reason is that the NP encoder has no explicit modeling of $\mathbf{x} \rightarrow y_{\mathbf{x}}$ relationship (which is further explained in Appendix C.1.7), and thus may mainly capture the semantic difference in the inputs.

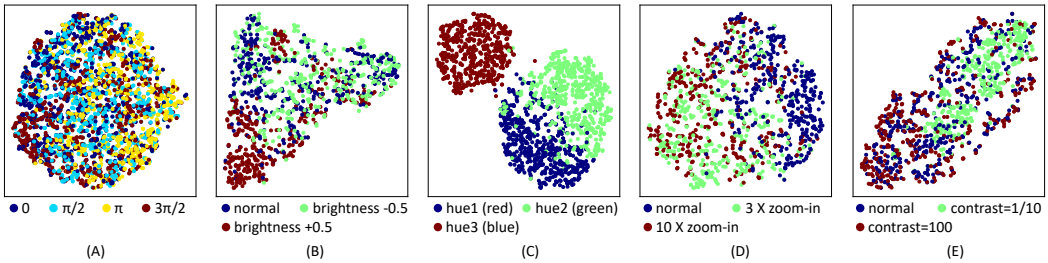

Figure C.1: TSNE visualization of latent input embeddings from settings A to E based on an ordinary classifier.

C.1.6 Empirical Evaluation of Time Efficiency of IPML on the Anonymous E-Commerce Company's Risk Detection Dataset

This subsection empirically compares the time efficiency of different tested meta-learning algorithms on the aforementioned anonymous e-commerce company's risk detection dataset which contains around 1 million entries. The training setting is the same as that in Appendix C.1.2 with a meta-batch size of 10. One training epoch spans approximately 950 iterations. Table C.9 reports the results during meta-training: *Model forward* corresponds to the E step/SGHMC of IPML or the inner optimization loop of MAML-based algorithms, while *model backward* corresponds to the M step of IPML or the outer optimization loop of MAML-based algorithms. Our implementation of IPML obtains a total of 30 SGHMC samples per iteration for each task and MAML-based algorithms have 20 gradient steps in the inner optimization loop. The convergence is checked by inspecting the difference in the validation losses every 500 iterations.

The results in Table C.9 show that the time efficiency of IPML is comparable to that of the first-order MAML baseline since (a) the forward passes are not that slow compared with the inner optimization loops of first-order MAML, and (b) the cost of backward passes of IPML is similar to that of first-order MAML. The time efficiency of BMAML baseline [Yoon et al., 2018] is lower than that of both IPML and vanilla MAML because BMAML maintains a set of particles (i.e., 5 particles in this setting), each representing a distinct network parameter. Unless efficient parallel optimization of those particles are implemented, BMAML or other particle-based Bayesian variant of MAML (e.g., [Jerfel et al., 2019]) can be a few times slower than vanilla MAML. In comparison, IPML does not maintain a set of particles of network weights and is thus more time-efficient.

Table C.10 reports results during meta-testing (i.e., adaptation to a new task): *Model adaptation* corresponds to the E step/SGHMC of IPML or the inner optimization loop of MAML-based algorithms. Similar to the case during meta-training, IPML is comparable to first-order MAML in time efficiency during model adaptation, and BMAML has the

worst time efficiency due to the use of particles. When performing prediction of new inputs, the time efficiency of all tested meta-learning algorithms are nearly the same. Both IPML and BMAML can estimate uncertainty by using more samples/particles in their predictions. For IPML, we have observed that using only 1 SGHMC sample in its prediction can already give satisfactory results.

The *neural process* (NP) uses amortized VI which introduces an additional encoder. Thus, it is faster in model forward but slower in model backward.

Table C.9: Time efficiency of tested meta-learning algorithms during meta-training on the anonymous e-commerce company's risk detection dataset.

	Total Iterations until Convergence	Model Forward	Model Backward
NP	8000	0.74s/iteration	0.14s/iteration
MAML (first-order)	8000	0.99s/iteration	0.024s/iteration
BMAML (5 particles)	7000	3.7s/iteration	0.045s/iteration
IPML	8000	1.24s/iteration	0.026s/iteration

Table C.10: Time efficiency of tested meta-learning algorithms during meta-testing (i.e., adaptation to a new task) on the anonymous e-commerce company's risk detection dataset.

	Support Set Size	Model Adaptation	Model Prediction (100 entries)
NP	100	0.098s	0.009s/sample
MAML (first-order)		0.112s	0.009s
BMAML (5 particles)		0.308s	0.009s/particle
IPML		0.132s	0.009s/SGHMC sample

C.1.7 Ablation Study of the Limitations of Neural Process

In Chapter 4.4, we have empirically compared the performance of IPML with the *neural process* (NP) [Garnelo et al., 2018b]. For the implementation of NP, the default encoder is a 5-layer feedforward neural network (i.e., with 100 hidden neurons) whose output is $\mathbf{z}$. The aggregator is the mean function, as proposed in the original work of [Garnelo et al., 2018b]. The decoder has the same architecture as our IPML by adopting the same coupling of $\mathbf{z}$ with the DNN g_θ (i.e., by applying $\mathbf{z}$ as a mask to the last DNN layer's parameters).

When performing classification tasks, we use the aforementioned robust-max likelihood (Appendix 3.1.1).

We have observed that for both sinusoid regression (Table 4.1) and Omniglot (Table 4.2), NP performs unsatisfactorily as compared to IPML, likely due to the causes mentioned below.

Optimization difficulties and overfitting. NP performs amortized variational inference of $\mathbf{z}$ through a heavily parameterized encoder which may introduce optimization difficulties and overfitting during meta-training. This is supported by the empirical evidence in Table C.11: With an increasing depth of the encoder from the default of 2 layers, its performance improves and then deteriorates.

Table C.11: *Mean square error* (MSE) on few-shot sinusoid regression.

Encoder depth	Sinusoid 5-shot	Sinusoid 10-shot
2 layers	0.780	0.417
3 layers	0.521	0.328
5 layers	0.460	0.264
7 layers	0.477	0.289

No explicit modeling of $\mathbf{x} \rightarrow y_{\mathbf{x}}$ relationship in the support set. Our IPML algorithm performs SGHMC to obtain posterior samples of $\mathbf{z}$ from $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$. SGHMC relies on our IP framework (Def. 5.1) which explicitly defines the $\mathbf{x} \rightarrow y_{\mathbf{x}}$ relationship. Thus, the $\mathbf{x} \rightarrow y_{\mathbf{x}}$ relationship in the support set is well captured by IPML with SGHMC. On the other hand, the encoder of NP takes in the simple concatenation of $(\mathbf{x}, y_{\mathbf{x}})$ and thus does not explicitly capture the $\mathbf{x} \rightarrow y_{\mathbf{x}}$ relationship in the support set, which we think is crucial to accurately recovering the true latent task posterior belief $p(\mathbf{z}|\mathbf{y}_{\mathcal{X}_t^s})$.

Appendix D

Appendix for Chapter 5

D.1 Computational Cost of Forward-Backward Method and Naive Method

We will analyze here the computational cost of both the forward-backward method and the naive method to understand the improvement by adopting the former. Algo. 4 describes the forward-backward method, while Algo. 5 presents the naive method. The difference lies mainly in lines 7 and 8 of Algo. 4 where the forward-backward method uses online SVGD to perform fast belief updates for the selected task. On the other hand, the naive method computes such a belief of the meta-parameters from scratch using A (and $T \setminus (A \cup t)$) in lines 5 and 6 of Algo. 5.

Though the computational cost of VI (containing multiple SVGD iterations till convergence) on n tasks (specifically, the n sample datasets) is not necessarily n times the computational cost of VI on 1 task, we have observed in practice that VI on a *minibatch* of tasks converges within (slightly more but) nearly the same number of iterations as VI on one task; both require around 5 SVGD iterations. So, we have considered the following assumption:

Assumption D.1 (informal). *Suppose that the GPU can afford to process a minibatch of $n \leq B$ tasks in parallel. VI (using SVGD to compute posterior belief of the meta-*

Algorithm 5 Near-Optimal Active Task Selection based on MILT (Naive Method without Forward-Backward Method)

1: Set $A = \emptyset$;
2: **while** $|A| < k$ **do**
3: Sample $\mathbf{z}_A \sim p(\mathbf{z}_A)$, $\mathbf{z}_{\overline{A}} \sim p(\mathbf{z}_{\overline{A}})$;
4: **for** $t \in T \setminus A$ **do**
5: Compute $p\big(\theta|\mathbf{z}_A, \mathbf{y}_A^s\big) \leftarrow \text{SVGD}_\Theta\left(p(\theta)\,, \{\mathbf{z}_A, \mathbf{y}_A^s\}\right)$;
6: Compute $p\Big(\theta|\mathbf{z}_{\overline{A\cup t}}, \mathbf{y}_{\overline{A\cup t}}^s\Big) \leftarrow \text{SVGD}_\Theta\left(p(\theta)\,, \{\mathbf{z}_{\overline{A\cup t}}, \mathbf{y}_{\overline{A\cup t}}^s\}\right)$;
7: Compute with $p\big(\theta|\mathbf{z}_A, \mathbf{y}_A^s\big) : p(\mathbf{z}_t|\mathbf{z}_A) \leftarrow \text{VI}_{\mathcal{Z}}(p(\theta|\mathbf{z}_A, \mathbf{y}_A^s))$;
8: Compute with $p\Big(\theta|\mathbf{z}_{\overline{A\cup t}}, \mathbf{y}_{\overline{A\cup t}}^s\Big) : p(\mathbf{z}_t|\mathbf{z}_{\overline{A\cup t}}) \leftarrow \text{VI}_{\mathcal{Z}}\left(p\Big(\theta|\mathbf{z}_{\overline{A\cup t}}, \mathbf{y}_{\overline{A\cup t}}^s\Big)\right)$;
9: Estimate $H(\mathcal{Z}_t|\mathcal{Z}_A) - H(\mathcal{Z}_t|\mathcal{Z}_{\overline{A\cup t}})$;
10: **end for**
11: Select $t^\star = \arg\max_t H(\mathcal{Z}_t|\mathcal{Z}_A) - H(\mathcal{Z}_t|\mathcal{Z}_{\overline{A\cup t}})$;
12: Update $A = A \cup t^\star$;
13: **end while**
14: **return** A

parameters) on a minibatch of $n \leq B$ tasks converges in the same number of iterations as VI on 1 task.

Note that line 3 of Algo. 4 and lines 5 and 6 of Algo. 5 can enjoy computational benefits from such parallel processing of each minibatch of tasks. Since (a) the overall computational cost is dominated by the number of SVGD iterations in the algorithms and (b) the computational cost of one iteration of online SVGD (for both learning or unlearning) is the same as that of one SVGD iteration per task, the following results ensue:

Remark D.1. *The computational cost measured by the number of floating point operations is $\mathcal{O}(k|T|)$ for the forward-backward method and $\mathcal{O}(k|T|^2)$ for the naive method.*

Remark D.2. *The computational cost measured by the runtime is $\mathcal{O}(k|T|)$ for the forward-backward method and $\mathcal{O}(k|T|^2/B)$ for the naive method.*

The above results arise from iterating through all candidate tasks in T in each of the k rounds of Algo. 4 or Algo. 5. For every candidate task, the naive method requires $\mathcal{O}(|T|)$ floating point operations in SVGD, while the forward-backward method requires only $\mathcal{O}(1)$ floating point operations in online SVGD.

As can be concluded from the above remarks, the forward-backward method improves time efficiency by $|T|$ times over the naive method because its online SVGD performs fast belief updates for only 1 selected task instead of using A (and $T \setminus (A \cup t)$). The parallel processing of each minibatch of tasks can accelerate the naive method by B times, but the naive method is still much slower than the forward-backward method since usually, $B \ll |T|$.

D.2 EM Algorithm for Meta-Learning and Modified Variant

D.2.1 Probabilistic Meta-Learning with IP

Meta-learning on a set A of tasks, which adopt a split of their corresponding datasets according to Chapter 5.2, can be defined as the inference of meta-parameters Θ with the following joint likelihood to be maximized [Chen et al., 2014, Finn et al., 2017, Finn et al., 2018]:

$$p(\mathbf{y}_A^r, \theta | \mathbf{y}_A^s) = p(\theta) \prod_{t \in A} p(\mathbf{y}_t^r | \theta, \mathbf{y}_t^s) = p(\theta) \prod_{t \in A} \int_{\mathbf{f}_t^r} p(\mathbf{y}_t^r | \mathbf{f}_t^r)\, p(\mathbf{f}_t^r | \theta, \mathbf{y}_t^s)\, \mathrm{d}\mathbf{f}_t^r . \tag{D.1}$$

Task *adaptation* $p(\mathbf{f}_t^r | \theta, \mathbf{y}_t^s)$ is performed via IP inference [Chen et al., 2021] given the sample dataset $(X_t^s, \mathbf{y}_t^s)$:

$$p(\mathbf{f}_t^r | \theta, \mathbf{y}_t^s) = \int_{\mathbf{z}} p(\mathbf{f}_t^r | \mathbf{z}, \theta)\, p(\mathbf{z} | \theta, \mathbf{y}_t^s)\, \mathrm{d}\mathbf{z} . \tag{D.2}$$

Fig. 5.1 shows a graphical model of the IP for meta-learning. Note that in meta-learning, we have to model the belief of $\mathcal{Y}_t^r$ and use both the sample and remaining datasets to perform meta-training. On the other hand, we have to model the belief of $\mathcal{Y}_t^s$ to perform task selection (Chapter 5.4).

[Chen et al., 2021] have used an *expectation maximization* (EM) algorithm to perform meta-training such that the E step carries out the IP inference of $\mathcal{Z}$ (and $f(\cdot)$) in (D.2)

and the M step maximizes the joint likelihood (D.1) w.r.t. a point estimate of Θ. Note that with our additional Bayesian treatment of Θ, meta-training returns a posterior belief $p(\theta|\mathbf{y}_A^s, \mathbf{y}_A^r)$ of the meta-parameters instead of a point estimate, which is empirically shown to improve performance (Chapter 5.5). While [Chen et al., 2021] are mainly interested in deriving such an EM algorithm for meta-training, the motivation of our work here is to derive an efficient active task selection algorithm for meta-learning with a near-optimal performance guarantee. Nevertheless, we have modified their EM algorithm to work in our case with the Bayesian treatment of Θ.

D.2.2 Our modification

We will provide details of the EM algorithm proposed by [Chen et al., 2021] and describe our modifications here.

Expectation (E) step. The aim of the E step is to obtain (samples of) $p(\mathbf{f}_t^r|\theta, \mathbf{y}_t^s)$. Note that (samples of) $p(\mathbf{f}_t^r|\theta, \mathbf{y}_t^s)$ can be obtained using the generator g_Θ (5.1) and the *latent task posterior belief* $p(\mathbf{z}|\theta, \mathbf{y}_t^s)$, as follows: First draw samples of $\mathbf{z}$ from $p(\mathbf{z}|\theta, \mathbf{y}_t^s)$, and then passing them and X_t^r as inputs to generator g_Θ to obtain samples from $p(\mathbf{f}_t^r|\theta, \mathbf{y}_t^s)$. Hence, for a task t, performing its adaptation $p(\mathbf{f}_t^r|\theta, \mathbf{y}_t^s)$ (D.2) reduces to obtaining the latent task posterior belief $p(\mathbf{z}|\theta, \mathbf{y}_t^s)$.

In general, $p(\mathbf{z}|\theta, \mathbf{y}_t^s)$ cannot be evaluated in closed form. Instead of using *variational inference* (VI), the work of [Chen et al., 2021] has drawn samples from $p(\mathbf{z}|\theta, \mathbf{y}_t^s)$ using *stochastic gradient Hamiltonian Monte Carlo* (SGHMC), which introduces an auxiliary random vector $\mathbf{r}$ and samples from the joint distribution $p(\mathbf{z}, \mathbf{r}|\theta, \mathbf{y}_t^s)$ following the Hamiltonian dynamics [Brooks et al., 2011, Neal, 1993] and making use of the tractable gradient on $\mathcal{Z}$: $-\nabla_\mathbf{z} \log p(\mathbf{z}|\theta, \mathbf{y}_t^s) = -\nabla_\mathbf{z} \log p(\mathbf{z}, \mathbf{y}_t^s|\theta) = -\nabla_\mathbf{z}[\log p(\mathbf{y}_t^s|\mathbf{f}_t^s = (g_\theta(\mathbf{x}, \mathbf{z}))_{\mathbf{x}\in X_t^s}^\top) + \log p(\mathbf{z})]$.

Our modification of E step: We represent the belief of Θ as a set $\{\theta^m\}_{m=1}^M$ of M particles, which we denote as $q(\theta)$ without loss of generality. To accelerate computation,

we use VI in Chapter 5.4.2 to compute $p(\mathbf{z}|\theta^m, \mathbf{y}_t^s)$:

$$p(\mathbf{z}|\theta^m, \mathbf{y}_t^s) \leftarrow \text{VI}_{\mathcal{Z}}(\theta^m) \ .$$

We then pass the samples of $p(\mathbf{z}|\mathbf{y}_t^s, \theta^m)$ to the generator g_{θ^m} (i.e., corresponding to the particle θ^m) to obtain samples of $p(\mathbf{f}_t^r|\theta^m, \mathbf{y}_t^s)$.

Maximization (M) step. The aim of the M step is to optimize (D.1) w.r.t. a point estimate of θ using the samples from the E step. In particular, (D.1) is optimized through gradient descent w.r.t. θ using samples of $\mathbf{z}$ from the E step.

Our modification of M step: Instead of using gradient descent, we use SVGD (Chapter 5.4.1) on the datasets $(X_A, \mathbf{y}_A)$ of the subset A of tasks to compute $p(\theta|\mathbf{y}_A)$ such that in every SVGD iteration, each particle θ^m is updated as follows (i.e., similar to that in Chapter 5.4.1):

$$\theta^m \leftarrow \theta^m + \frac{\eta}{M} \sum_{\theta \in \{\theta^m\}_{m=1}^M} \big[k(\theta, \theta^m) \\ \times \nabla_\theta \log p(\mathbf{y}_A^r|\theta, \mathbf{y}_A^s) + \nabla_\theta k(\theta, \theta^m)\big]$$

where η is the step size, and $k(\cdot, \cdot)$ is a *radial basis function* kernel representing a repulsive force between particles to prevent them from collapsing. We denote the entire VI process (containing multiple SVGD iterations till convergence) to obtain $p(\theta|\mathbf{y}_A)$ as

$$p(\theta|\mathbf{y}_A) \leftarrow \text{SVGD}_\Theta(p(\theta), \mathbf{y}_A) \ .$$

D.3 Proofs and other Theoretical Results

D.3.1 Proof of Theorem 5.1

Firstly, note that the MILT$(\cdot)$ function (5.2) is submodular:

Lemma D.1. *The set function $A \mapsto$ MILT(A) is submodular.*

Its proof can be found in [Krause and Golovin, 2014]. We can also formally prove that the MILT$(\cdot)$ function (5.2) is approximately monotonic up to a constant error of C_0:

Lemma D.2. $\forall \mathcal{B} \subseteq T \setminus A$ MILT$(A \cup \mathcal{B}) \geq$ MILT$(A) - C_0$ *where* $C_0 \triangleq H(\Theta)$.

Proof.

$$
\begin{aligned}
&\text{MILT}(A \cup \mathcal{B}) - \text{MILT}(A) \\
&= H(\mathcal{Z}_{\mathcal{B}}|\mathcal{Z}_A) - H(\mathcal{Z}_{\mathcal{B}}|\mathcal{Z}_{T\setminus(A\cup\mathcal{B})}) \\
&\geq H(\mathcal{Z}_{\mathcal{B}}|\mathcal{Z}_A, \Theta) - H(\mathcal{Z}_{\mathcal{B}}|\mathcal{Z}_{T\setminus(A\cup\mathcal{B})}) \\
&\geq H(\mathcal{Z}_{\mathcal{B}}|\mathcal{Z}_A, \Theta) - H(\mathcal{Z}_{\mathcal{B}}, \Theta|\mathcal{Z}_{T\setminus(A\cup\mathcal{B})}) \\
&= H(\mathcal{Z}_{\mathcal{B}}|\Theta) - H(\mathcal{Z}_{\mathcal{B}}, \Theta|\mathcal{Z}_{T\setminus(A\cup\mathcal{B})}) \\
&\geq H(\mathcal{Z}_{\mathcal{B}}|\Theta) - (H(\mathcal{Z}_{\mathcal{B}}|\Theta, \mathcal{Z}_{T\setminus(A\cup\mathcal{B})}) + H(\Theta|\mathcal{Z}_{T\setminus(A\cup\mathcal{B})})) \\
&= -H(\Theta|\mathcal{Z}_{T\setminus(A\cup\mathcal{B})}) \\
&\geq -H(\Theta)
\end{aligned}
$$

where the first equality is by definition of MILT$(\cdot)$ function (5.2), the first and last inequalities are due to the "conditioning reduces entropy" property, the second and third inequalities are due to chain rule for entropy, and the second and last equalities follow from $\mathcal{Z}_t, \mathcal{Z}_{t'}, \forall t \neq t'$ being mutually independent given meta-parameters Θ. □

Proof of Theorem 5.1. Let $t_1^\star, \ldots, t_k^\star$ be the k tasks selected by Algo. 4. So, $A_i = \{t_1^\star, \ldots, t_i^\star\}$. Recall from (5.3) that $A^\star \triangleq \arg\max_{A\subseteq T, |A|=k}$ MILT(A). From Lemma D.2,

$$
\text{MILT}(A^\star \cup A_i) \geq \text{MILT}(A^\star) - C_0 \tag{D.3}
$$

for all rounds $i = 1, \ldots, k$. Let $\Delta_i \triangleq$ MILT$(A_i) -$ MILT(A_{i-1}). From Lemma D.1, we know that for any A and $t \notin (A \cup A_i)$,

$$
\begin{aligned}
&\text{MILT}((A \cup A_i) \cup t) - \text{MILT}(A \cup A_i) \\
&\leq \text{MILT}(A_i \cup t) - \text{MILT}(A_i) \leq \Delta_{i+1} \,.
\end{aligned}
$$

It follows that for any A s.t. $|A| = k$,

$$\text{MILT}(A \cup A_i) - \text{MILT}(A_i) \leq k\Delta_{i+1} \ .$$

Consequently,

$$\text{MILT}(A^\star \cup A_i) \leq \text{MILT}(A_i) + k\Delta_{i+1} = \sum_{j=1}^{i} \Delta_j + k\Delta_{i+1} \ . \tag{D.4}$$

From (D.3) and (D.4),

$$\text{MILT}(A^\star) - C_0 \leq \sum_{j=1}^{i} \Delta_j + k\Delta_{i+1} \tag{D.5}$$

for round $i = 0, \ldots, k-1$. We refer to (D.5) as the i-th inequality. Now, by multiplying both sides of the i-th inequality by a factor of $(1/k)(1-(1/k))^{k-i-1}$ and then summing both sides over all rounds $i = 0, \ldots, k-1$,

$$\begin{aligned} &\big(\text{MILT}(A^\star) - C_0\big) \sum_{i=0}^{k-1} \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-i-1} \\ &\leq \sum_{i=1}^{k} \Delta_i = \text{MILT}(A_k) \ . \end{aligned} \tag{D.6}$$

To understand why the coefficient of Δ_i in (D.6) reduces to 1 for $i = 1, \ldots, k$, the Δ_i term in the j-th inequality (D.5) for $j = i, \ldots, k-1$ has a coefficient of $(1/k)(1-(1/k))^{k-j-1}$, while the $k\Delta_i$ term in the $(i-1)$-th inequality (D.5) has a coefficient of $(1/k)(1-(1/k))^{k-(i-1)-1}$. Then,

$$\begin{aligned} &\text{coefficient of } \Delta_i \text{ in (D.6)} \\ &= 1 \times \sum_{j=i}^{k-1} \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-j-1} + k \times \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i-1)-1} \\ &= 1 \ . \end{aligned}$$

From (D.6),

$$\begin{aligned}\text{MILT}(A_k) &\geq \big(\text{MILT}(A^\star) - C_0\big)\left(1 - \left(1 - \frac{1}{k}\right)^k\right) \\ &\geq (\text{MILT}(A^\star) - C_0)(1 - 1/e)\ .\end{aligned}$$

D.3.2 Near-Optimal Performance Guarantee for MIRD

We will now describe a greedy algorithm similar to Algo. 1:

Algorithm 2. *Start with an empty set $A_0 = \emptyset$ of tasks. In each round $i = 1, \ldots, k$, greedily select the next task:*

$$t_i^\star \triangleq \arg\max_t \text{MIRD}(A_{i-1} \cup t) - \text{MIRD}(A_{i-1}) \tag{D.7}$$

and update the set $A_i = A_{i-1} \cup t_i^\star = \{t_1^\star, \ldots, t_i^\star\}$ of selected tasks.

Theorem D.1 (Near-optimal performance guarantee)**.** *Algorithm 2 is guaranteed to select a set A of k tasks s.t.*

$$\text{MIRD}(A) \geq (1 - 1/e)(\text{OPT} - C_0)$$

where $\text{OPT} \triangleq \max_{A \subseteq T, |A| = k} \text{MIRD}(A)$ *and the constant $C_0 = H(\Theta)$ is the entropy of meta-parameters Θ.*

Lemma D.3. $\forall \mathcal{B} \subset T \setminus A$ *MIRD*$(A \cup \mathcal{B}) \geq$ *MIRD*$(A) - C_0$ *where* $C_0 \triangleq H(\Theta)$.

Proof.

$$
\begin{aligned}
&\text{MIRD}(A \cup \mathcal{B}) - \text{MIRD}(A) \\
&= H(\mathcal{Y}^r_{\mathcal{B}}|\mathcal{Y}^r_A) - H(\mathcal{Y}^r_{\mathcal{B}}|\mathcal{Y}^r_{T\setminus(A\cup\mathcal{B})}) \\
&\geq H(\mathcal{Y}^r_{\mathcal{B}}|\mathcal{Y}^r_A, \Theta) - H(\mathcal{Y}^r_{\mathcal{B}}|\mathcal{Y}^r_{T\setminus(A\cup\mathcal{B})}) \\
&\geq H(\mathcal{Y}^r_{\mathcal{B}}|\mathcal{Y}^r_A, \Theta) - H(\mathcal{Y}^r_{\mathcal{B}}, \Theta|\mathcal{Y}^r_{T\setminus(A\cup\mathcal{B})}) \\
&= H(\mathcal{Y}^r_{\mathcal{B}}|\Theta) - H(\mathcal{Y}^r_{\mathcal{B}}, \Theta|\mathcal{Y}^r_{T\setminus(A\cup\mathcal{B})}) \\
&\geq H(\mathcal{Y}^r_{\mathcal{B}}|\Theta) - (H(\mathcal{Y}^r_{\mathcal{B}}|\Theta, \mathcal{Y}^r_{T\setminus(A\cup\mathcal{B})}) + H(\Theta|\mathcal{Y}^r_{T\setminus(A\cup\mathcal{B})})) \\
&= -H(\Theta|\mathcal{Y}^r_{T\setminus(A\cup\mathcal{B})}) \\
&\geq -H(\Theta)
\end{aligned}
$$

where the first equality is by definition of MILT$(\cdot)$ function (5.2), the first and last inequalities are due to the "conditioning reduces entropy" property, the second and third inequalities are due to chain rule for entropy, and the second and last equalities follow from $\mathcal{Y}^r_t, \mathcal{Y}^r_{t'}, \forall t \neq t'$ being mutually independent given meta-parameters Θ. □

Proof of Theorem D.1. Similar to MILT, MIRD is submodular since it is also a mutual information criterion. Lemma D.3 reveals that MIRD is approximately monotonic. Therefore, we can adopt the same proof as that for MILT (Appendix D.3.1) to derive the near-optimal performance guarantee for MIRD.

D.3.3 Online SVGD for Learning: Theoretical Result

Proposition D.1 (Online SVGD for learning). *Suppose that $p(\theta)$ is an uninformative prior (e.g., $\nabla_\theta \log p(\theta) = 0$). With $\{\theta^m\}_{m=1}^M$ initially sampled from $p\big(\theta|\mathbf{z}_A, \mathbf{y}^s_A\big)$, the SVGD operation for obtaining $p\big(\theta|\mathbf{z}_{A\cup t}, \mathbf{y}^s_{A\cup t}\big)$ (i.e., learning from a task $t \notin A$) is*

$$
\begin{aligned}
\theta^m \leftarrow \theta^m + \frac{\eta}{M} \sum_{\theta\in\{\theta^m\}_{m=1}^M} \big[&k(\theta, \theta^m) \\
&\times \nabla_\theta \log p(\theta|\mathbf{z}_t, \mathbf{y}^s_t) + \nabla_\theta k(\theta, \theta^m)\big] \,.
\end{aligned} \tag{D.8}
$$

We denote such a learning process (containing multiple SVGD iterations till convergence) as

$$p(\theta|\mathbf{z}_{A\cup t}, \mathbf{y}^s_{A\cup t}) \leftarrow \mathit{SVGD}_\Theta(p(\theta|\mathbf{z}_A, \mathbf{y}^s_A), \{\mathbf{z}_t, \mathbf{y}^s_t\}) \; .$$

Proof. With $\{\theta^m\}_{m=1}^M$ initially sampled from an arbitrary distribution $q(\theta)$, the SVGD operation for learning $p\big(\theta|\mathbf{z}_{A\cup t}, \mathbf{y}^s_{A\cup t}\big)$ is

$$\begin{aligned}
\theta^m \leftarrow \theta^m + \frac{\eta}{M} \sum_{\theta\in\{\theta^m\}_{m=1}^M} & \big[k(\theta, \theta^m) \\
& \times \nabla_\theta \log p(\theta|\mathbf{z}_{A\cup t}, \mathbf{y}^s_{A\cup t}) + \nabla_\theta k(\theta, \theta^m)\big] \\
= \theta^m + \frac{\eta}{M} \sum_{\theta\in\{\theta^m\}_{m=1}^M} & \big[k(\theta, \theta^m) \\
& \times \nabla_\theta (\log p(\theta|\mathbf{z}_t, \mathbf{y}^s_t) + \log p(\theta|\mathbf{z}_A, \mathbf{y}^s_A) - \log p(\theta)) \\
& + \nabla_\theta k(\theta, \theta^m)\big] \; .
\end{aligned}$$

Note that SVGD is a discretization of the underlying continuous functional gradient in the *reproducing kernel Hilbert space* (RKHS) of $k(\cdot, \cdot)$. The corresponding continuous functional gradient, which is proven to be the expectation of the Stein's operator [Liu and Wang, 2016], of the above SVGD is

$$\begin{aligned}
\mathbb{E}_{q(\theta)}[k(\theta, \cdot) \times \nabla_\theta \big(& \log p(\theta|\mathbf{z}_t, \mathbf{y}^s_t) \\
& + \log p(\theta|\mathbf{z}_A, \mathbf{y}^s_A) - \log p(\theta)\big) + \nabla_\theta k(\theta, \cdot)] \; .
\end{aligned}$$

Since we have assumed an uninformative prior $p(\theta)$, $\nabla_\theta \log p(\theta) = 0$ in the above SVGD. Also, using Stein's identity [Liu and Wang, 2016], $\mathbb{E}_{p(\theta|\mathbf{z}_A, \mathbf{y}^s_A)}[k(\theta, \cdot)\nabla_\theta \log p(\theta|\mathbf{z}_A, \mathbf{y}^s_A)] = 0$. Since $q(\theta) = p(\theta|\mathbf{z}_A, \mathbf{y}^s_A)$, the above functional gradient reduces to

$$\mathbb{E}_{q(\theta)}[k(\theta, \cdot)\nabla_\theta \log p(\theta|\mathbf{z}_t, \mathbf{y}^s_t) + \nabla_\theta k(\theta, \cdot)]$$

which results in the SVGD in (D.8). □

D.3.4 Proof of Proposition 5.1

Unlearning from a task $t \in A$ can be cast as a problem of minimizing the *evidence upper bound* (EUBO):

$$\text{EUBO} \triangleq \mathbb{E}_{q(\theta)}[\log p(\mathbf{y}_t^s|\mathbf{z}_t, \theta)] + \text{KL}(q(\theta)\|p(\theta|\mathbf{z}_A, \mathbf{y}_A^s))$$

which yields the exact solution to the original problem of maximizing the *evidence lower bound* (ELBO), which involves learning from the set $A \setminus t$ of tasks [Nguyen et al., 2020b].

The above problem can also be cast as one of maximizing the *negative EUBO* (NEUBO): w.r.t. $q(\theta)$:

$$\text{NEUBO} \triangleq \mathbb{E}_{q(\theta)}[-\log p(\mathbf{y}_t^s|\mathbf{z}_t, \theta)] - \text{KL}(q(\theta)\|p(\theta|\mathbf{z}_A, \mathbf{y}_A^s)) \ . \tag{D.9}$$

Note that SVGD is a discretization of the underlying continuous functional gradient in the *reproducing kernel Hilbert space* (RKHS) of $k(\cdot, \cdot)$. The corresponding continuous functional gradient, which is proven to be the expectation of the Stein's operator [Liu and Wang, 2016], of $-\text{KL}(q(\theta)\|p(\theta|\mathbf{y}_A^s, \mathbf{z}_A))$ is

$$\mathbb{E}_{q(\theta)}[k(\theta, \cdot)\nabla_\theta\big(\log p(\theta|\mathbf{z}_A, \mathbf{y}_A^s)\big) + \nabla_\theta k(\theta, \cdot)] \ .$$

Using Stein's identity [Liu and Wang, 2016], $\mathbb{E}_{p(\theta|\mathbf{z}_A, \mathbf{y}_A^s)}[k(\theta, \cdot)\nabla_\theta \log p(\theta|\mathbf{z}_A, \mathbf{y}_A^s)] = 0$. Since $q(\theta) = p(\theta|\mathbf{z}_A, \mathbf{y}_A^s)$, the above functional gradient reduces to $\mathbb{E}_{q(\theta)}[\nabla_\theta k(\theta, \cdot)]$.

On the other hand, the functional gradient of $\mathbb{E}_{q(\theta)}[-\log p(\mathbf{y}_t^s|\mathbf{z}_t, \theta)]$ [Liu and Wang, 2016] is

$$\begin{aligned}
&-\mathbb{E}_{q(\theta)}[k(\theta, \cdot)\nabla_\theta(\log p(\theta|\mathbf{z}_t, \mathbf{y}_t^s) - \log p(\theta))] \\
&= -\mathbb{E}_{q(\theta)}[k(\theta, \cdot)\nabla_\theta \log p(\theta|\mathbf{z}_t, \mathbf{y}_t^s)]
\end{aligned}$$

such that the equality follows from our assumption of an uninformative prior $p(\theta)$, that is, $\nabla_\theta \log p(\theta) = 0$. By summing the above two functional gradients, we obtain the functional

gradient of (D.9):

$$\mathbb{E}_{q(\theta)}[-k(\theta,\cdot)\nabla_\theta \log p(\theta|\mathbf{z}_t,\mathbf{y}_t^s) + \nabla_\theta k(\theta,\cdot)]$$

which results in the SVGD in (5.9).

D.4 More Experimental Results

D.4.1 Some Implementation Details

For Sinusoid regression, we execute 1 SVGD iteration to perform the adaptation of a task in meta-training and execute 10 SVGD iterations to perform the adaptation of a task in meta-test. The training is performed for a total of 15000 iterations with an Adam optimizer. For Omniglot and MiniImagenet classification, we execute 5 SVGD iterations to perform the adaptation of a task in meta-training and execute 10 SVGD iterations to perform the adaptation of a task in meta-test. The training is performed for a total of 60000 iterations with an Adam optimizer. For other meta-training settings, we adopt the same as that in [Chen et al., 2021] (Chapter 4).

D.4.2 Comparison with Two Recent Baselines: the Probabilistic Active Meta-learning algorithm and the Greedy Class-Pair Based Sampling

The following describes our implementation details of the two baselines which we compared with. Note that both works have different problem setting as ours. Thus we have to adapt their implementation to perform fair empirical comparison baseds on the experimental settings discussed in Chapter 5.5.

D.4.2.1 The probabilistic active meta-learning algorithm (PAML)

To describe our adapted implementation of PAML from [Kaddour et al., 2020], contextual information is omitted to ensure a fair comparison since all other baselines in our experiments do not need to assume the availability of such contextual information. To achieve this, we have removed the regularization term associated with the contextual information (i.e., the summation term in Equation (10) of [Kaddour et al., 2020]). Then, the meta-training objective becomes a typical meta-learning objective (i.e., Equation (5) of [Kaddour et al., 2020]) which is similar to (is the variational lower bound of) ours that explicitly represents the distributions via variational parameters. In our implementation of PAML, we have adopted the same Gaussian variational distribution of the latent vector in [Kaddour et al., 2020] and used the same model architectures as the ones in our baselines.

Results and discussion. The results on Sinusoid are presented on Figure D.1 and the results on Omniglot and MiniImageNet are presented on Figure D.2. The results show that our proposed MILT outperforms PAML. From our experiments, we have observed that PAML is more inclined to select "boundary" tasks: For example, in 10-shot Sinusoid where the amplitude amp is sampled from $[0.1, 5.0]$, PAML has a proportion of 28.9% and 13.3% of its selected tasks from amp > 4.0 and amp < 1.0, respectively. In contrast, our proposed MILT has a proportion of 17.5% and 7.5% of its selected tasks from amp > 4.0 and amp < 1.0, respectively. So, the PAML criterion tends to rank "boundary" tasks more highly due to their greater surprisals [R1]. However, selecting too large a proportion of such "boundary" tasks may not yield as much information on the unobserved tasks, which explains why PAML is outperformed by our proposed MILT.

D.4.2.2 The greedy class-pair based sampling (GCP)

The GCP proposed in [Liu et al., 2020] is an active task selection method for meta-classification tasks. GCP does not directly sample tasks. Instead, it samples classes and then generates the tasks by randomly picking images of the selected classes, which is fundamentally different from our setting. So, to establish a meaningful empirical

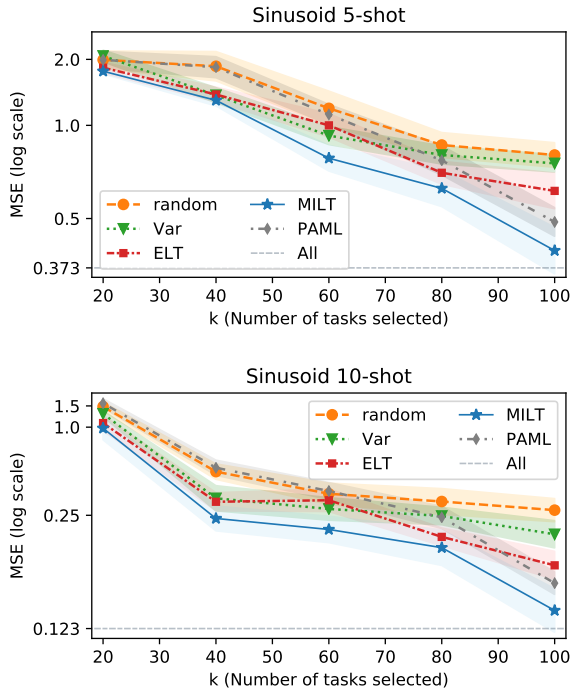

Figure D.1: Meta-test *mean squared error* (MSE) and standard error over 5 runs vs. no. k of selected tasks on (left) 5-shot Sinusoid and (right) 10-shot Sinusoid.

comparison with GCP, we have to adapt it such that it fits into the general problem setting where the candidate tasks are given/fixed prior to task selection.

In the original GCP setting involving a N-way classification problem, a N-clique of classes is sampled in each round such that the selection probability is formulated based on its potential which is the product of pairwise potentials between two classes.

In our adapted implementation of GCP, we use GCP to compute the potential of each candidate task where each task corresponds to a N-way classification problem. Then, we sample tasks (i.e., one task per round of task selection without replacement) according to the probability computed from the potential. By doing so, instead of allowing GCP to generate tasks, we use the GCP criterion to sample from the given candidate tasks. We adopt the default hyperparameters value (i.e., aggressiveness of 1 and discounting factor of 0.5) in the original GCP paper.

Results and discussion. The results on Omniglot and MiniImageNet are presented on Figure D.2. The results show that our proposed MILT outperforms GCP. We think

the reason is that GCP is designed to achieve the best generalization performance on new classes during test time. However, MILT is designed to achieve the best generalization performance on new tasks (but not necessarily new classes) during test time. MILT better fits our general problem setting and thus achieves a better performance than GCP.

D.4.3 Comparison with Information-Theoretic Task Selection

While it is complicated to generalize the existing active task selection algorithms to cater to our more general problem setting (i.e., without making strong assumptions of the availability of contextual information or a validation set, we have found a way to adapt the problem setting of the *information-theoretic task selection* (ITTS) algorithm [Luna and Leonetti, 2020] to do so. A core assumption of ITTS is the availability of a validation set that can accurately represent the entire (meta-test) task distribution. Such a strong assumption contradicts the motivation of our active task selection problem, as discussed in Chapter 5.1. However, we can first construct such a validation set by randomly selecting $k' < k$ candidate tasks as the validation set which we now denote as V.[1] Then, we actively select the rest of the $(k - k')$ tasks according to the ITTS criterion. In our experiments, we set $k' = k/2$.

Note that ITTS is conventionally designed for meta-reinforcement learning, but can be adapted to cater to supervised meta-learning in our problem setting. The two key components of ITTS are (a) the quantification of the *difference* between two tasks measured by KL divergence, and (b) the quantification of the *relevance* of a task t to a task t' measured by the difference in entropy. We adapt their meta-reinforcement learning framework into our (probabilistic) supervised meta-learning framework, as follows:

(a) In the work of [Luna and Leonetti, 2020], the *difference* between two tasks t and t' is measured as the averaged KL divergence between their respective policies over the states of the validation tasks in V. For supervised meta-learning, it is natural to consider

[1]To align with our problem setting where the remaining datasets for all tasks in A are acquired/observed for meta-training only after A is selected by an active task selection algorithm (Sections 5.3 and 5.2), only the *a priori* known sample datasets for all tasks in V are given during active task selection.

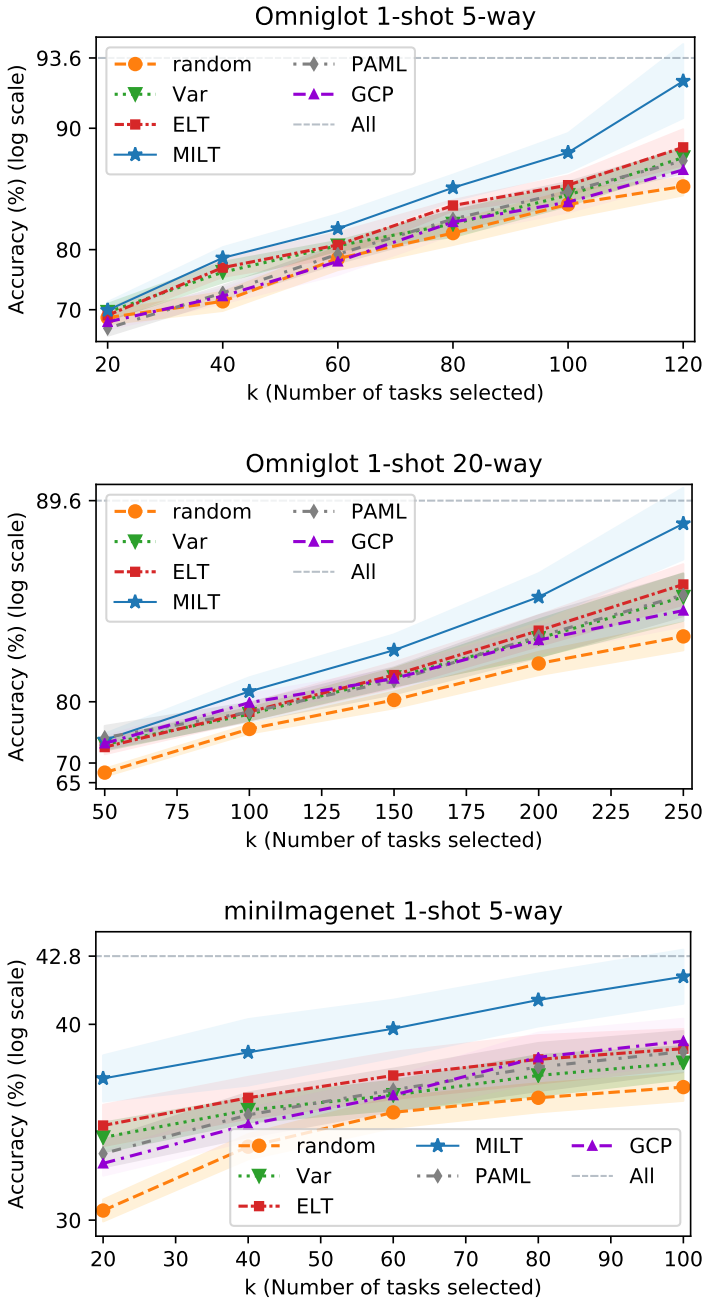

Figure D.2: Meta-test accuracy and standard error over 5 runs vs. no. k of selected tasks on (top) 1-shot 5-way Omniglot, (middle) 1-shot 20-way Omniglot, and (bottom) 1-shot 5-way MiniImageNet.

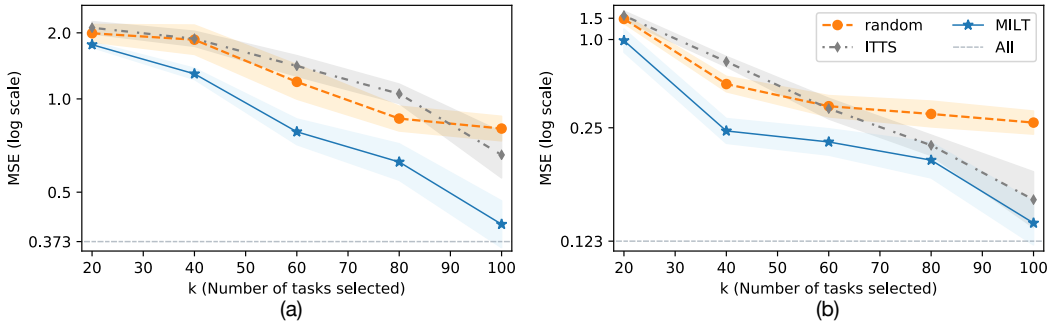

Figure D.3: Meta-test *mean squared error* (MSE) and standard error over 5 runs vs. no. k of selected tasks on (a) 5-shot Sinusoid and (b) 10-shot Sinusoid.

the averaged KL divergence between their latent task beliefs over the latent task vectors representing the tasks in V:

$$\delta(t,t') = \mathbb{E}_{\mathbf{z}_V \sim p(\mathbf{z}_V)}[\mathrm{KL}(p(\mathcal{Z}_t = \mathbf{z}|\mathbf{z}_V) \| p(\mathcal{Z}_{t'} = \mathbf{z}|\mathbf{z}_V))] \ .$$

(b) The work of [Luna and Leonetti, 2020] has defined the *relevance* of task t to task t' as the expected difference in the entropy of the policies before and after learning (over the states of t') w.r.t. the on-policy distribution before learning. For supervised meta-learning, such a relevance translates to

$$\rho(t,t') = H(\mathcal{Z}_{t'}|\mathcal{Z}_t) - H(\mathcal{Z}_t) \ .$$

Apart from these two components, the rest of the ITTS algorithm follows the original implementation.

Fig. D.3 shows results of Sinusoid regression comparing the meta-test performance of MILT vs. ITTS. It can be observed that MILT outperforms ITTS for both 5-shot and 10-shot Sinusoid regression. Note that ITTS does not outperform random selection when k is not sufficiently large. This may be due to its validation set (whose size $k' < k$) not being highly representative of the distribution of meta-test tasks. As a result, the calculation of the difference (see (a) above) using such a validation set can be misleading. When k is sufficiently large, the validation set becomes representative enough for ITTS to perform

better than random selection.