THE UNIVERSITY *of* EDINBURGH

Meta-learning to Optimise: Loss Functions and Update Rules

Boyan Gao

Doctor of Philosophy

Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh

2023

Abstract

Meta-learning, aka "learning to learn", aims to extract invariant meta-knowledge from a group of tasks in order to improve the generalisation of the base models in the novel tasks. The learned meta-knowledge takes various forms, such as neural architecture, network initialization, loss function and optimisers. In this thesis, we study learning to optimise through meta-learning with of main components, loss function learning and optimiser learning. At a high level, those two components play important roles where optimisers provide update rules to modify the model parameters through the gradient information generated from the loss function. We work on the meta-model's re-usability across tasks. In the ideal case, the learned meta-model should provide a "plug-and-play" drop-in which can be used without further modification or computational expense with any new dataset or even new model architecture. We apply these ideas to address three challenges in machine learning, namely improving the convergence rate of optimisers, learning with noisy labels, and learning models that are robust to domain shift.

We first study how to meta-learn loss functions. Unlike most prior work parameterising a loss function in a black-box fashion with neural networks, we meta-learn a Taylor polynomial loss and apply it to improve the robustness of the base model to label noise in the training data. The good performance of deep neural networks relies on gold-stand labelled data. However, in practice, wrongly labelled data is common due to human error and imperfect automatic annotation processes. We draw inspiration from hand-designed losses that modify the training dynamic to reduce the impact of noisy labels. Going beyond existing hand-designed robust losses, we develop a bi-level optimisation meta-learner Automated Robust Loss (ARL) that discovers novel robust losses that outperform the best prior hand-designed robust losses.

A second contribution, ITL, extends the loss function learning idea to the problem of Domain Generalisation (DG). DG is the challenging scenario of deploying a model trained on one data distribution to a novel data distribution. Compared to ARL where the target loss function is optimised by a genetic-based algorithm, ITL benefits from gradient-based optimisation of loss parameters. By leveraging the mathematical guarantee from the Implicit Function Theorem, the hypergradient required to update the loss can be efficiently computed without differentiating through the whole base model training trajectory. This reduces the computational cost dramatically in the meta-learning stage and accelerates the loss function learning process by providing a more accurate hypergradient. Applying our learned loss to the DG problem, we are able to learn base

models that exhibit increased robustness to domain shift compared to the state-of-the-art. Importantly, the modular plug-and-play nature of our learned loss means that it is simple to use, requiring just a few lines of code change to standard Empirical Risk Minimisation (ERM) learners.

We finally study accelerating the optimisation process itself by designing a meta-learning algorithm that searches for efficient optimisers, which is termed MetaMD. We tackle this problem by meta-learning Mirror Descent-based optimisers through learning the strongly convex function parameterizing a Bregman divergence. While standard meta-learners require a validation set to define a meta-objective for learning, MetaMD instead optimises the convergence rate bound. The resulting learned optimiser uniquely has mathematically guaranteed convergence and generalisation properties.

Acknowledgements

I would like to thank my supervisor Professor Timothy Hosepedales for his enthusiasm, support guidance and encouragement throughout the course of my PhD.

I am grateful for helpful comments, discussions and proofreads from many over the course of this work, particularly Hakan Bilen (my second supervisor), Yongxin Yang, Henry Gouk, Jan Stuhmer and Hae Beom Lee. I would also like to thank my colleagues in MIG, VICO and IPAB. It is lucky to be around such a group of great minds. Thank you to my friends in my office and my internship colleagues.

A big thank you goes to my parents and my aunt, Yiyang Xie, for their unwavering love and support and for celebrating all my success proudly. Thank you to my family and friends. Finally, a special thank you goes to Bingqing Guo for making my life better and supporting me throughout the years.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Boyan Gao*)

To my grandma Shizhi Yang.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contemporary deep neural networks have comparable performance with humans on a variety of challenging tasks, such as visual object classification, detection, and segmentation. These abilities are gained by learning to understand raw features with an iterative process using numerous training examples. However, despite the ability to solve specific tasks, humans are also good at adapting the learned knowledge and the learning experience to the process of quickly understanding new subjects and improving the understanding of previously learned concepts. i.e. the learned knowledge and learning experience from one field can be helpful in another stage of studying. In contrast to machine learning models, especially large neural networks which require huge amounts of training data and training time, Humans are better able to generalise the knowledge to novel concepts. The question then is can a machine learning system be equipped with similar abilities? Finn and Levine (2018) and their following research provide a seminal approach to handling such problems, where the deep neural networks are able to classify novel objects through a limited number of training steps on a few training samples and this ability is enabled by the pre-learned "learning to learn" knowledge, termed meta-knowledge in the form of network initialisation. This line of research is named meta-learning. Such learning models and algorithms are designed to absorb learning skills from multiple training episodes and utilise those skills to achieve faster learning and better performance than conventional non-meta models in a similar task regime.

With a long history and various extensions to multiple areas, it is difficult to give a clear definition to meta-learning (Thrun and Pratt, 1998; Schmidhuber et al., 1997;

Schmidhuber, 1987). Compared with the task-specific machine learning methods aiming to improve prediction in a single data distribution setting, meta-learning provides an alternative paradigm where it aims to improve the algorithm itself in terms of the generalisation ability over a task distribution, where the learned knowledge can be reused in the unseen tasks from the same distribution. Due to the large body of the historical context and the rapidly growing area, in this work, we mainly focus on meta-learning in the deep neural network context.

Deep meta-learning problems are usually constructed under bilevel optimisation, where the meta-representation, i.e. meta-knowledge, is captured by solving outer loop optimisation tasks and the corresponding base models, neural networks, are solved conditioned on outer loop statuses. The meta-learning algorithm design space is built by spanning three axes: meta-optimiser, meta-objective and meta-representation (Hospedales et al., 2020). Meta-optimiser and meta-objective focus on the choices of the optimisers and objectives used in the outer loop optimisations to learn the meta-representation, the instantiations of meta-representation includes hyperparameter optimisation (Franceschi et al., 2018), loss function learning (Jenni and Favaro, 2018; Bechtle et al., 2020; Houthooft et al., 2018; Wu et al., 2018a; Gonzalez and Miikkulainen, 2020a), optimiser learning (Wichrowska et al., 2017), and neural architecture search (Real et al., 2019; Tan and Le, 2019). According to different training protocols, meta-learning can be further categorised into two main families, online and offline meta-learning and those have different application scenarios. In online meta-learning methods, the meta-models are treated as auxiliary tasks to help the base model learning process, e.g. online update of the hyperparameter in the optimiser to jump out of the saddle points in the loss landscape, modifying the loss function to adjust the training feedback signal. In these problems, the inner loop only contains one task. While to eliminate the bias introduced by hand-engineering learning components, offline meta-learning methods explore the transferability and reusability of the gained meta-representation. Usually, this protocol uses a multi-task inner loop to prevent meta-overfitting, overfitting to the training tasks, for the generalisation ability of the captured meta-representation.

In this work, we study meta-learning to optimise. Aligning with the multi-task offline meta-learning objective, three meta-learning algorithms are proposed to meta-learn reusable loss functions and optimisers. Loss function learning and optimiser learning are two active research areas under this umbrella which tackle the ”learning to learn what to learn” and ”learning to learn how to learn” problems respectively. Due to the

universal approximation ability, neural networks are commonly applied in the existing loss function learning and optimiser learning research work. However, such a design often suffers from two main problems: i) due to the high model complexity, learning the meta-representation parameterised by neural networks requires a tremendous computation ii) the Low interpretability of the learned neural network makes the analysis of the whole system difficult. It is not clear how those learned meta-models affect the neural network training dynamics. Gonzalez and Miikkulainen (2021) successfully deploy high order multi-variate Taylor polynomial as the target loss function to reduce the meta-learning complexity and make the loss function more transparent and the neural network supervised by the learned loss generalises better than that trained by Cross-Entropy on original training tasks and architecture. We expand this parameterisation to more challenging tasks, such as noisy label classification, and domain generalisation. More importantly, instead of learning task-specific meta-models, we discuss the transferability and reusability of the meta-model and learning dataset and architecture-agnostic loss functions. Similarly and motivated by KFAC (Martens and Grosse, 2015) parameterisation, we alleviate this black-box parameterisation problem in the optimiser learning. By introducing a novel meta-objective to learn the optimiser, the meta-learning algorithm can produce an optimiser with fast convergence speed as well as a convergence guarantee. Besides a large body of application studies based on meta-learning, we also explore meta-learning efficiency by applying efficient meta-learning methods and reformulating the optimisation problems in our cases.

As a whole, we produce two loss function algorithms with different meta-optimisation methods and a theoretically guaranteed optimiser learning algorithm which improves a proposed meta-generalisation bound. The corresponding applications range from noisy label classification, domain generalisation and standard classification. In addition, our work also covers meta-learning efficiency with both empirical and theoretical analysis.

1.1 List of Contributions

The main contributions of this thesis are three meta-learning-based algorithms, two of which focus on learning loss functions on noisy label tasks and domain generalisation. Another algorithm particularly works on transferable optimiser learning by optimising a theoretically guaranteed meta-objective enabling the learned optimiser with a fast convergence rate.

- **ARL** is a family of smooth and differentiable robust loss functions for noisy label tasks. Motivated by the data-driven AutoML (Hutter et al., 2019), these losses can be automatically learned for a given noise ratio, and easily plugged into existing neural network training frameworks. Compared with the hand-engineered drop-in replacement of the standard losses such as Cross Entropy (Wang et al., 2019; Zhang and Sabuncu, 2018; Ghosh et al., 2017), ARL provides a better general purpose loss than prior hand-designed robust losses and it allows problem-specific loss learning.
- **ITL** works as an extension of previous work providing higher meta-learning efficiency compared with evolutionary methods (Gonzalez and Miikkulainen, 2020a) and reinforcement-learning (Li et al., 2019a) based algorithm and is expanded the loss function learning to Domain Generalisation. Inspired by DomainBed (Gulrajani and Lopez-Paz, 2021) where most state-of-the-art Domain Generalisation methods failed to reliably defeat ERM, ITL is the first loss function study on DG problem, and the first demonstration that suitably and automatically chosen loss function can further improve the previously almost unbeaten ERM baseline in DG.
- **MetaMD** starts optimiser learning from revisiting the perspective of *Mirror Descent*. Differing from the existing optimiser learning work (Andrychowicz et al., 2016; Wichrowska et al., 2017; Flennerhag et al., 2020; Bello et al., 2017), the MetaMD provides guarantees of convergence and generalisation by learning the Bregman divergence parameterisation under the Mirror Descent framework.

1.2 Problem Scenarios

Meta-learning has broad applications. In this thesis, we design three meta-learning algorithms to handle three different challenging problems, Noisy Classification, domain generalisation and fast convergence with the algorithms, ARL, ITL and MetaMD respectively. Here only the general ideas about those problems are introduced, the formal version will be discussed in the later chapters.

Noisy Label Classification is an important research problem due to the ubiquitous data noise in real-world applications. The outstanding performance of deep Neural networks requires training on a large amount of correctly labelled data samples. However, due to the imperfect automatic labelling system and the inevitable mistakes from manual

labelling, the generalisation ability of the neural network degrades since this type of overparameterised model is able to overfit all the training sets with any noise ratios. Robust loss functions(Zhang and Sabuncu, 2018; Wang et al., 2019) are handed crafted to reduce the impact of the corrupted labels. We apply ARL to learn loss functions with respect to different noise types and noise ratios to train the models robust to the label noise.

Domain Generalisation studies the out-of-distribution generalisation problem where the training and testing sets violate the independent and identically distributed (i.i.d.) assumption. The machine learning models have rapidly decreased performance when there is a domain shift between the training and testing distribution, especially in the Domain Generalisation setting where the learners have no access to the target domain data. This problem occurs widely in many machine learning application scenarios. Differing from the research works spanning from data augmentation, special training protocols and regularisers. ITL handles this problem by learning the domain-shift robust loss functions.

Fast Convergence studies efficient optimisation algorithms to accelerate the expensive neural network training process. Adam (Kingma and Ba, 2015) leverages second-order momentum information to each optimising iteration, leading to a faster convergence speed than commonly used SGD in deep neural network training. However, this line of research either sacrifices the generalisation ability for training speed or does not have convergence guarantees. Second-order gradient-based optimisers, such as Newton's method, converge to optima in many few iterations, but its notorious Hessian matrix computation is not suitable for the deep learning setting. Estimating the diagonal of the Hessian matrix (Yao et al., 2021) shed the light on the efficient Hessian matrix computation however, the random noise may affect the generalisation ability of the trained model. MetaMD optimises its Bregman divergence on a variety of tasks and generalises well to similar unseen tasks in the deployment stage. In addition, Instead of setting generalisation as the meta-objective, we optimise the convergence rate and make it generalised across unseen meta-test tasks, and then the models trained with learned MetaMD have a fast convergence speed.

1.3 Thesis Structure

The thesis consists of three main chapters, each is supported by a published paper. The three chapters are listed as follows:

- Chapter 3 introduces ARL and it is based on the following paper:

  Searching for Robustness: Loss Learning for Noisy Classification Tasks. Boyan Gao, Henry Gouk, and Timothy Hospedales, *International Conference on Computer Vision (ICCV)*, 2021.

- Chapter 4 introduces ITL and it is based on the following paper:

  Loss Function Learning for Domain Generalization by Implicit Gradient. Boyan Gao, Henry Gouk, Yongxin Yang, and Timothy Hospedales, *International Conference on Machine Learning (ICML)*, 2022.

- Chapter 5 introduces MetaMD and it is based on the following papers:

  Meta Mirror Descent: Optimiser Learning for Fast Convergence. Boyan Gao, Henry Gouk, Hae Beom Lee, and Timothy Hospedales, *Conference on Learning Representation (ICLR) Workshop on GroundedML*, 2022 *(best paper finalist)*

  MetaMD: Learning an Optimiser with Convergence Guarantees. Boyan Gao, Henry Gouk, Jan Stuehmer, and Timothy Hospedales, 2022.

1.4 Side Project Publication

A list of side project publications, which are accomplished during the PhD study period but not highly related to this thesis, is given as follows:

Deep clustering with concrete k-means. Boyan Gao, Yongxin Yang, Henry Gouk and Timothy Hospedales, *IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 2020.

Deep clustering for domain adaption. Boyan Gao, Yongxin Yang, Henry Gouk and Timothy Hospedales, *IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 2020.

Chapter 2

Background

This chapter discusses the required background, especially the techniques for meta-optimisers, for the later chapters, and presents an overview of prior methods for solving the bilevel optimisation problems in the meta-learning area. We start this chapter by introducing the basic setup for meta-learning, and how different types of bilevel optimisations are applied to different meta-learning targets. Then three different families, including explicit hypergradient computation, implicit gradient computation, and gradient-free method will be introduced in Section 2.2.1, Section2.2.2 and Section 2.3 respectively.

2.1 Bilevel Optimisation for Meta-learning

Meta-learning algorithms are designed to learn to learn shared meta-knowledge that should be beneficial for the learning process on the related novel tasks, such as accelerating the learning process and getting better generalisation ability compared with those trained from scratch. To be more specific, a group of meta-train tasks $D^{meta-train}$ are gathered for the meta-model training in the meta-train stage, then in the meta-test (deployment stage), the learned models are evaluated on the unseen novel tasks, $D^{meta-test}$. It is assumed that a task distribution governing the meta-train tasks and the meta-test tasks, which is represented as $D_i^{meta-train}, D_j^{meta-test} \overset{\text{i.i.d.}}{\sim} P(\mathcal{T})$, an analogy to the training and testing setting in conventional machine learning.

Contemporary meta-learning is often formalised as a bi-level optimisation problem where we aim to learn a learning algorithm that improves upon conventional learning.

The two optimisation problems are nested, where solving one level depends on the current situation of the other. For convenience, the two levels are termed as inner loop and outer loop. Plugging into the deep learning setting, the inner loop problem optimises its objective, $\mathcal{L}^{inner}$, w.r.t. the base model (neural network) parameters, $\theta \in \mathcal{R}^m$, and the outer loop optimise meta-models, $\omega \in \mathcal{R}^n$. Constructing the meta-learning target such as neural architecture, loss function, optimiser and data augmentation policy into the outer loop, meta-learning algorithms can work as a general machine learning tool to learn those components. The general framework is given as follows:

$$\omega^* = \arg\min_{\omega} \sum_{i=1}^{N} \mathcal{L}^{outer}(\theta_i^*(\omega), \omega, D_i^{meta-val}) \tag{2.1}$$

$$\text{s.t.}\, \theta_i^*(\omega) = \arg\min_{\theta} \mathcal{L}^{inner}(\theta, \omega, D_i^{meta-train}) \tag{2.2}$$

where Equation 2.1 and Equation 2.2 represents the outer loop and inner loop respectively. Note that the inner loop optimisation depends on the current outer loop condition, ω, and the outer loop relies on the best response from the inner loop, $\theta^*(\omega)$. This bilevel optimisation problem is solved in the meta-train stage and aims to capture interesting meta-knowledge which can be directly utilised in the meta-test stage without further modification and adaptation of the meta-model according to the new tasks. In order to gain such reusable meta-models, the whole inner loop training where the based model is trained from scratch until converge conditioned on meta-models is required to avoid the short horizon bias introduce by truncating inner loop training (Wu et al., 2018b). We will discuss the computational challenges in this situation. For simplicity without losing generality, we assume that in the meta-train stage there is only one meta-train task:

$$\omega^* = \arg\min_{\omega} \mathcal{L}^{outer}(\theta^*(\omega), \omega) \tag{2.3}$$

$$\text{s.t.}\, \theta^*(\omega) = \arg\min_{\theta} \mathcal{L}^{inner}(\theta, \omega), \tag{2.4}$$

based on which we give a general summary in Alg 1. This algorithm is known as the off-line meta-learning framework, in the sense that a full inner loop is required for each outer loop updating and once the outer loop reaches its convergence point, the learned meta-models can be “plug-to-play“ in the novel tasks. However, the framework has a notorious reputation for its computational expense in the meta-train stage and the required full inner loops make the hypergardient computation intractable when the inner loop contains a large neural network with long training horizons. In addition, this offline setting optimising degrades large meta-model training efficiency.

Algorithm 1 Online Meta-learning framework.

Input: ω, η, α
Output: ω^*, θ^*
Init ω
Init θ {Get random network weights}
while not converged or reached max steps **do**
    $\theta = \theta - \alpha \nabla_\theta \mathcal{L}^{inner}(\theta, \omega)$ {Train the base model}
    $h = \text{Hypergradient}(\mathcal{L}^{outer}, \theta(\omega))$
    $\omega = \omega - \eta h$ {Update the meta model}
end while

Algorithm 2 Offline Meta-learning framework.

Input: ω, η
Output: ω^*
Init ω
while not converged or reached max steps **do**
    Init θ {Get random network weights}
    $\theta^* = \arg\min_\theta \mathcal{L}^{inner}(\theta, \omega)$ {Train the base model}
    $h = \text{Hypergradient}(\mathcal{L}^{outer}, \theta^*(\omega))$
    $\omega = \omega - \eta h$ {Update the meta model}
end while

Another line of research (Wang et al., 2020; Li et al., 2019d; Lee et al., 2021; Li et al., 2019a), named online meta-learning, assumes the local response from the base model can sufficiently update the meta-model and the outer loop is updated much more frequently. Instead of focusing on learning reusable meta-knowledge, this type of method improves the performance on a simple task by making the meta-model adapt to the base model training process online. However, since both the inner loop and outer loop optimisation problems are not solved exactly, neither the outer loop nor the inner loop can give a sufficient response to the other, and the drifting problem may happen in the optimisation process. Moreover, the discovered meta-knowledge is subject to a small training period, which vanishes the transferability and reusability of the meta-knowledge. When a new task comes, the whole algorithm is executed from scratch. A summary for a single meta-train task setting is given in Algorithm 1.

2.2 Gradient-Based Bilevel Optimisation

We have framed the bilevel optimisation for meta-learning in the last section with the introduction to two main meta-learning branches, online and offline meta-learning, and their own design objectives. In this section, we brief the main challenge of solving that bilevel optimisation with gradient-based optimisation algorithms.

In Equation 2.1 and Equation 2.2, it is described that the outer loop updating depends on the best response from the inner loop, which is iteratively solved by gradient-based optimisers in the deep learning setting. In addition, due to the nested optimisation that inner loop learning is also conditioned on the outer loop status, thus the best response of the base model is treated as a function of the meta-model. Then the immediate question is when the outer loop is also a gradient-based optimiser how to compute the gradient, usually termed as hypergradient, $\frac{\partial \mathcal{L}^{outer}}{\partial \omega}$, with respect to the meta parameters. From the chain rule, the gradient is decomposed into:

$$\frac{\partial \mathcal{L}^{outer}(\theta^*(\omega),\omega)}{\partial \omega} = \underbrace{\frac{\partial \mathcal{L}^{outer}(\theta^*(\omega),\omega)}{\partial \omega}}_{\text{direct gradient}} + \underbrace{\frac{\partial \mathcal{L}^{outer}(\theta^*(\omega),\omega)}{\partial \theta^*(\omega)} \frac{\partial \theta^*(\omega)}{\partial \omega}}_{\text{indirect gradient}} \tag{2.5}$$

where there are two main terms, direct gradient and indirect gradient. The direct gradient can be easily computed by the existing auto-differentiation library, such as Pytorch (Paszke et al., 2019) and TensorFlow (Abadi et al., 2015). The indirect gradient computation depends on the optimisation algorithm since to compute $\frac{\partial \theta^*(\omega)}{\partial \omega}$ needs to differentiate through the inner loop optimisation $\theta^*(\omega) = \arg\min_\theta \mathcal{L}^{inner}(\theta, \omega, D^{meta-train})$. We denote the iterative update rule in the inner loop as:

$$\theta_t = \pi(\theta_{t-1}, \omega), \tag{2.6}$$

where π can be any gradient-based optimiser. Without losing generality, we illustrate the computational challenges by instantiating π as Gradient Descent which yields:

$$\theta_t = \pi(\theta_{t-1}, \omega) = \theta_{t-1} - \alpha \frac{\partial \mathcal{L}^{inner}(\theta, \omega)}{\partial \theta}\bigg|_{\theta=\theta_{t-1}}, \tag{2.7}$$

and we further assume that one gradient step is enough to achieve the inner loop optimum from a random initialisation:

$$\omega = \arg\min_\omega \mathcal{L}^{outer}(\theta^*(\omega), \omega) \tag{2.8}$$

$$\text{s.t. } \theta^*(\omega) = \arg\min_\theta \mathcal{L}^{inner}(\theta, \omega) = \theta_0 - \alpha \frac{\partial \mathcal{L}^{inner}(\theta, \omega)}{\partial \theta}\bigg|_{\theta=\theta_0} \tag{2.9}$$

then we have the gradient computation for ω:

$$\frac{\partial \mathcal{L}^{outer}(\theta^*(\omega),\omega)}{\partial \omega} = \underbrace{\frac{\partial \mathcal{L}^{outer}(\theta^*(\omega),\omega)}{\partial \omega}}_{\text{direct gradient}} + \underbrace{\frac{\partial \mathcal{L}^{outer}(\theta^*(\omega),\omega)}{\partial \theta^*(\omega)} \frac{\partial \theta^*(\omega)}{\partial \omega}}_{\text{indirect gradient}} \tag{2.10}$$

$$= \frac{\partial \mathcal{L}^{outer}(\theta^*(\omega),\omega)}{\partial \omega} \tag{2.11}$$

$$+ \frac{\partial \mathcal{L}^{outer}(\theta^*(\omega),\omega)}{\partial \theta^*(\omega)} \frac{\partial \theta^*(\omega)}{\partial \omega}\Bigg|_{\theta^*(\omega)=\theta_0-\alpha\frac{\partial \mathcal{L}^{inner}(\theta_0,\omega)}{\partial \theta_0}} \tag{2.12}$$

$$= \frac{\partial \mathcal{L}^{outer}(\theta^*(\omega),\omega)}{\partial \omega} \tag{2.13}$$

$$- \alpha \frac{\partial \mathcal{L}^{outer}(\theta^*(\omega),\omega)}{\partial \theta^*(\omega)} \frac{\partial^2 \mathcal{L}^{inner}(\theta,\omega)}{\partial \theta \partial \omega}\Bigg|_{\theta^*(\omega)=\theta_0-\alpha\frac{\partial \mathcal{L}^{inner}(\theta_0,\omega)}{\partial \theta_0},\theta=\theta_0}, \tag{2.14}$$

where it can be observed that differentiating through the inner loop generates the second-order derivative terms whose dimension increases linearly with respect to both the inner loop and outer loop model's dimension. Note that the computational complexity of hypergradient is also affected by the number of unrolling inner loop iterations. For simplicity, in Equation 2.9, we only take one inner loop step and assume that is enough to reach optima, which is durable in the few-shot learning setting. In more complicated cases, it usually takes a few steps and even whole training iterations in the online and offline meta-learning respectively. In some methods, such as First Order MAML, the indirect gradient term is ignored to reduce the computation cost which leads to a decrease in performance. Nichol et al. (2018) use average moving trajectory to learn their meta initialisation without computing the high order derivative. Shin et al. (2021) extends this idea by introducing adjustments to the trajectory to better approximate update information only for learning the neural network initialisation. Instead of dropping the second order term blindly, we will discuss the meta-learning technique for indirect gradient computation in the next section.

2.2.1 Explicit Gradient Method

We have constructed the general framework for the hypergradient computation consisting of the direct gradient and the indirect gradient parts. The direct gradients are straightforward to compute through the modern automatic differentiation (AD) library. However, the identified indirect gradient computational challenge remains unaddressed. In this section, we introduce two main explicit gradient methods: Forward Mode Differ-

entiation (FMD) (Franceschi et al., 2017) and Reverse Mode Differentiation (RMD). Both methods rely on the inner loop optimisation iteration, while FMD starts its hypergradient computation from the first weight θ_0 to the last θ_*, and RMD does it inversely. The different computation orders equip those two algorithms with different properties and cause different computational complexity in terms of both time and memory cost.

We start the introduction to this family of methods from Reverse Mode Differentiation which computes the hypergradient analogously to the backpropagate through time (Werbos, 1990). The algorithm holds the training information including weights and activation in each inner loop iteration. When the inner loop reaches the last weight θ_T then propagates towards the initial one θ_0 while holding the whole inner loop optimisation trajectory. This method is scalable to the dimension of the meta-model, but the memory cost increases linearly with the inner loop updating iteration steps. In addition due to the low and high curvature in the loss landscape and the highly nonlinear model, the hypergradient may vanish and explode.

We derive the executable hypergradient computation starting from replacing the optimisation problem in Equation 2.4 with unrolling the minimisation problem into iterative updates and assume $\theta^* = \theta^T$ when T is large enough:

$$\omega^* = \underset{\omega,\theta_1,...,\theta_T}{\arg\min}\ \mathcal{L}^{outer}(\theta_T(\omega),\omega) \tag{2.15}$$

$$\text{s.t. } \theta_t = \pi(\theta_{t-1},\omega), \qquad t \in \{1,...,T\}. \tag{2.16}$$

To solve this constrained optimisation problem, we build its Lagrangian:

$$L(\theta,\omega,\gamma) = \mathcal{L}^{outer}(\theta^*(\omega),\omega) + \sum_{t=1}^{T} \gamma_t(\pi(\theta_{t-1},\omega) - \theta_t). \tag{2.17}$$

Then taking the partial derivatives of the Lagrangian yields:

$$\frac{\partial L}{\partial \gamma_t} = \pi(\theta_{t-1},\omega) - \theta_t, \qquad t \in \{1,...,T\} \tag{2.18}$$

$$\frac{\partial L}{\partial \theta_t} = \gamma_{t+1}\frac{\partial \pi(\theta_{t-1},\omega)}{\partial \theta_{t-1}} - \theta_t, \qquad t \in \{1,...,T-1\} \tag{2.19}$$

$$\frac{\partial L}{\partial \theta_T} = \frac{\partial \mathcal{L}^{outer}}{\partial \theta_T} - \gamma_T, \tag{2.20}$$

$$\frac{\partial L}{\partial \omega} = \sum_{t=1}^{T} \gamma_t \frac{\partial \pi(\theta_{t-1},\omega)}{\partial \omega}. \tag{2.21}$$

The notations, $A_t = \frac{\partial \pi(\theta_{t-1},\omega)}{\partial \theta_{t-1}}$ and $B_t = \frac{\partial \pi(\theta_{t-1},\omega)}{\partial \omega}$ are usually introduced for simplifying the expression. By setting the derivatives on the left-hand sides of Equation 2.19 and

Equation 2.20 to zeros in order to reach the optimal conditions, and combine them to Equation 2.21, we obtain:

$$\frac{\partial L}{\partial \omega} = \frac{\partial \mathcal{L}^{outer}}{\partial \theta_T} \sum_{t=1}^{T} \left(\prod_{t'=t+1}^{T} A_{t'} \right) B_t \tag{2.22}$$

Then the pseudo-code for evaluating $\frac{\partial \theta_T}{\partial \omega}$ iteratively is given in Algorithm 3. RMD is widely applied in the meta-learning area. Some research focuses on improving its computational efficiency. For example, Lee et al. (2021) approximates the hyerpgradient by the method used in the knowledge distillation task to avoid saving the whole training trajectory. Alternatively, Fu et al. (2016) proposes to interpolate the intermedia terms between the first and last stage of the training trajectory to compute RMD.

In comparison, FMD avoids saving the whole inner loop computational graph. By decomposing the gradient computation in a single step, we have:

$$\frac{\partial \theta_t(\omega)}{\partial \omega} = \frac{\partial \pi(\theta_{t-1}, \omega)}{\partial \theta_{t-1}} \frac{\partial \theta_{t-1}}{\partial \omega} + \frac{\partial \pi(\theta_{t-1}, \omega)}{\partial \omega}, \tag{2.23}$$

where the indirect gradient component is updated while the base model is updated. A general idea for FMD is summarised in Algorithm 4 from which a significant drawback is observed that the Jacobian matrix $B_t \in \mathcal{R}^{m \times n}$ is explicitly computed in each iteration and matrix size increases along with the meta model's size which makes the memory cost and computation cost increase linearly with the number of meta parameters. This critical problem limits FMD only capable to the low dimensional meta-representation.

By unrolling $\frac{\partial \theta_T}{\partial \omega}$ and instantiating each step by Equation 2.23, the FMD computation is represented as:

$$\frac{\partial \theta_T}{\partial \omega} = \sum_{t=1}^{T} \left(\prod_{t'=t+1}^{T} A_{t'} \right) B_t \tag{2.24}$$

which coincides with the result for RMD derived in Equation 2.22.

FMD has broad applications in recent meta-learning works and has been modified to adapt to various applications. Micaelli and Storkey (2021) utilise FMD to meta-learn their learning rate and momentum parameter on different training stages for a single task which achieves similar performance with the human-turned models. In Chapter 5, FMD is applied to compute the hypergradient with respect to 2×2 diagonal matrices to optimise the corresponding Bregman Divergences, where the learned MetaMDs are able to research the goal minimums of a family of quadratic optimisation problems with much fewer iterations compared with hand-designed optimisers.

Algorithm 3 Reverse Mode Differentiation

Input: ω, θ_0
Output: gradient w.r.t to ω
for t = 1 to T do **do**
    $\theta_t = \pi(\theta_{t-1}, \omega)$
end for
$h_T = \frac{\partial \mathcal{L}^{outer}}{\partial \theta_T}$
$H = 0$
for t = T-1 down to 1 do **do**
    $H = H + h_{t+1} B_{t+1}$
    $h_t = h_{t+1} A_{t+1}$
end for
return H

Algorithm 4 Forward Mode Differentiation

Input: ω, θ_0
Output: gradient w.r.t to ω
$Z_0 = 0$
for t = 1 to T do **do**
    $\theta_t = \pi(\theta_{t-1}, \omega)$
    $Z_t = A_t Z_{t-1} + B_t$
end for
return $\frac{\partial \mathcal{L}^{outer}}{\partial \theta_T} Z_T$

2.2.2 Implicit Gradient Method

Conducting a full optimisation, where the base model is trained from scratch until converging in each outer loop iteration, can prevent meta-models suffer from short-term bias (Wu et al., 2018b). Both FMD and RMD have their own limitation when deployed to meta-learn the complicated reusable meta-models with long training horizons. FMD cannot generalise to the high dimensional meta-models due to the explicit computation of the Jacobian matrix of the base model with respect to the meta-model. RMD avoids computing such a Jacobian matrix, but the expensive memory cost for saving the entire training trajectory makes the RMD limited to short-horizon training. Implicit gradient, which approximates the hypergradient when the inner loop model reaches its fix point, provides a sufficient solution to alleviate both problems.

In general, optimising the inner loop problem until converge reaches a point (θ', ω') satisfying:

$$\frac{\partial \mathcal{L}^{inner}}{\partial \theta}\bigg|_{\theta', \omega'} = 0 \tag{2.25}$$

then by applying the Implicit Function Theorem (IFT) we have:

$$\frac{\partial \mathcal{L}^{inner}}{\partial \theta}\bigg|_{\theta^*(\omega), \omega} = 0. \tag{2.26}$$

where $\theta^*(\omega)$ is a point near θ' and is treated as a function of ω, which is required for the hypergradient computation. By taking the gradient with respect to ω on both sides of Equation 2.26, it yields:

$$\frac{\partial \theta^*(\omega)}{\partial \omega} = -\left[\frac{\partial^2 \mathcal{L}^{inner}}{\partial \theta \partial \theta}\right]^{-1} \frac{\partial^2 \mathcal{L}^{inner}}{\partial \theta \partial \omega}\bigg|_{\theta=\theta^*(\omega)}, \tag{2.27}$$

where we have a closed form solution for $\frac{\partial \theta^*(\omega)}{\partial \omega}$. Instead of computing hypergradient iteratively, all we need is to solve this dense representation. However, the computational complexity of inverting a Hessian matrix is intractable in the modern neural network scenario. Neumann series approximation provides one way to approximate the inverse Hessian matrix computation by expressing the inverse Hessian matrix computation as:

$$-\left[\frac{\partial^2 \mathcal{L}^{inner}}{\partial \theta \partial \theta}\right]^{-1} = \lim_{i \to \infty} \sum_{j=0}^{i} \left[I - \frac{\partial^2 \mathcal{L}^{inner}}{\partial \theta \partial \theta}\right]^j, \tag{2.28}$$

where Lorraine et al. (2020) propose an algorithm to efficiently solve the right-hand side term without computing the Hessian matrix. In practice, i is a design of a hyperparameter which controls the approximation accuracy and the computational cost and it does need to be infinite.

2.2.2.1 IFT Application

In this section, we introduce a special case of the IFT application, iMAML (Rajeswaran et al., 2019). iMAML meta learns the network initialization ω and introduces a regularizer to constrain the distance between the initialization and the convergence point in the inner loop:

$$\omega^* = \arg\min_{\omega} \mathcal{L}^{outer}(\theta^*(\omega), \omega) \tag{2.29}$$

$$\text{s.t.}\, \theta^*(\omega) = \arg\min_{\theta} \mathcal{L}^{inner}(\theta, \omega) + \frac{\lambda}{2}||\theta - \omega||^2. \tag{2.30}$$

Such modification to the inner loop instantiates the IFT-based closed-form gradient into:

$$\frac{\partial \theta^*(\omega)}{\partial \omega} = \left(I + \frac{1}{\lambda} \frac{\partial^2 \mathcal{L}^{inner}}{\partial \theta \partial \theta} \right)^{-1} \Big|_{\theta=\theta^*(\omega)}, \tag{2.31}$$

based on which, Rajeswaran et al. (2019) propose to approximate their indirect hypergradient by a free vector w:

$$||w - (I + \frac{1}{\lambda} \frac{\partial^2 \mathcal{L}^{inner}}{\partial \theta \partial \theta})^{-1} \frac{\partial \mathcal{L}^{outer}}{\partial \theta}|| \leq \varepsilon \tag{2.32}$$

where ε is a small positive number. In order to approximate the hypergradient by w, a quadratic optimisation problem is constructed:

$$\arg\min_{w} w^T (I + \frac{1}{\lambda} \frac{\partial^2 \mathcal{L}^{inner}}{\partial \theta \partial \theta}) w - w^T \frac{\partial \mathcal{L}^{outer}}{\partial \theta}. \tag{2.33}$$

In their work, Conjugate Gradient (CG) method is applied to solve this optimisation problem and the Hessian-vector product is implicitly computed to avoid the intractable Hessain matrix calculation.

IFT-based implicit gradient computation is much more efficient compared with FMD and RMD in terms of both memory and time cost. However, this method is not applicable to the online meta-learning algorithm due to the convergence point assumption in IFT. And also this implicit gradient can only be applied in a certain range of meta-learning cases. For example, it is not able to learn learning rates and the optimisers since those two types of components are not involved in closed-form gradient computation in Equation 2.27. In Chapter 4, we will cover the method of learning loss function for DG, and a special algorithm which leverages the Mirror Descent property to apply implicit gradient for the optimiser learning will be discussed in Chapter 5.

2.3 Gradient Free Methods

Gradient-based meta-learning, such as IFT-based methods and differentiating through the training horizon, offers efficient outer loop update information, however, the quality of the hypergradient is affected by a variety of factors. For example, the outer loop loss curvature may make the gradient vanish and explode, and the approximation of the gradient through the long horizon inner loop may introduce inaccurate estimation making the meta-training inefficient.

Besides gradient-based methods, non-gradient-based searching algorithms are often utilised in outer loop optimisation, such as grid search, random search (Bergstra and Bengio, 2012), Bayesian optimisation (Snoek et al., 2012), and genetic algorithm. Those methods are applied when there is a small parameter set and even when the inner loop is not differentiable. In order to conduct a sufficient grid search, a dense grid searching space is required but going through all the parameter combinations is very expensive. Random search theoretically has a higher chance to reach better sets of parameters than grid search. However, without using historical search information and guided by the efficient research policy, Random search still suffers a poor optimisation efficiency. Bayesian optimisation proposes each search point based on its current uncertainty. This method usually relies on the uncertainty estimator which contains several hyperparameter designs, such as the kernel choice in Gaussian Process, affecting the estimation performance.

Genetic algorithm is also commonly applied in the meta-learning setting, especially, when the meta-models are parameterised in an efficient and representative way. The outer loop iteration of the genetic algorithm does not require gradient computation, instead, it produces a population to gather model update signals to modify the current parameters which are not influenced by loss curvature. With such property, the algorithm can easily jumpy out of the local minimum and potentially solve the optimisation problem with the global optimum. However, the population-producing and the updated information generating process are very expensive. As the genetic algorithm is a diverse algorithm family, in this section we only introduce the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier, 1996), which is applied in Chapter 3 for loss function searching.

We give a rough introduction to CMA-ES in the bilevel optimisation setting. An intuitive explanation of the CMA-ES is searching for a Gaussian distribution to optimise the outer loop target objective. The Gaussian distribution is denoted as $P(\omega|\Omega)$, where $\Omega = \{m, C\}$. m and C represent the mean, and covariance matrix of the Gaussian distribution respectively and ω represents the meta-parameters. In each iteration, the mean of the distribution in the parameter space is modified by a weighted sum of the samples where the weights are evaluated based on the samples' performance. To be more specific, in each (outer loop) iteration, a population, containing λ individuals, is a sample from $\omega_1, ..., \omega_\lambda \sim P(\omega|\Omega)$ by:

$$\omega_k^j \sim m^j + \eta\mathcal{N}(0, C^j), \tag{2.34}$$

Algorithm 5 CMA-ES

1: **Input:** Ω^0
2: **Output:** Ω^*
3: **while** not converged or reached max steps **do**
4: Sample λ independent individuals $\omega_1^j, ..., \omega_\lambda^j \sim P(\omega|\Omega^j)$
5: Evaluate the samples on $\mathcal{L}^{outer}$
6: update Ω by $Eq.$ 2.35 and $Eq.$ 2.37
7: **end while**

where η is the learning rate. The mean is updated by the weighted sum of the top μ population according to their performance which in our case, the base model performances determined by each sample:

$$m^{j+1} = \sum_{i=1}^{\mu} \upsilon_i \omega_{i:\lambda}^{j+1} \tag{2.35}$$

$$\text{s.t.} \sum_{i=1}^{\mu} \upsilon_i = 1, \qquad \upsilon_1 \geq \upsilon_2 \geq ...\upsilon_\mu > 0, \tag{2.36}$$

where $i:\lambda$ denotes the i-th individual in the sampled population after ranking. The Covariance matrix updating is performed as:

$$C^{j+1} = (1 - c_\mu)C^j + c_\mu \frac{1}{\eta^2} C_\mu^{j+1} \tag{2.37}$$

$$C_\mu^{j+1} = \sum_{i=1}^{\mu} \upsilon_i (\omega_{i:\lambda}^{j+1} - m^g)(\omega_{i:\lambda}^{j+1} - m^g)^T. \tag{2.38}$$

An overall summary of CMA-ES is given in Algorithm 5. There are many other existing methods which are able to solve the outer loop optimisation problem, such as Reinforcement Learning. Since there is not a direct relationship to our work. Those methods are not discussed here.

2.4 Out of Distribution Meta-generalisation

The generalisation ability of the intelligence models is framed as the final goal in machine learning research. However, due to the diversity of machine learning problems, the definition of generalization varies from area to area. In standard supervised learning, the learners are trained and tested with the assumption that the training and testing data are independent and identically distributed (i.i.d.) data instances, whose generalisation

ability is described as the model's performance on the test set evaluated by the test error. On the theoretical side, generalisation analysis Valiant (1984); Blum et al. (2017); Shalizi and Kontorovich (2013); Kontorovich (2014) studies the dependence of bounds of test error on the training error and model complexity in various machine learning problems. Instructed by these theoretical works, one can estimate the generalisation performance potentially achieved by making tradeoffs between the amount of training data and the model complicity.

Domain generalisation is the area dealing with the out-of-distribution (o.o.d.) problems, where the learners have access to multiple distinct source domains to train their learning models and the learned models are challenged with cross-domain robustness on the unseen target domains in the test stage. In analogy to the stander supervised learning, the source and the target domains can be imaged as the correspondences to the training and testing sets from the learning and deploying protocol point of view, but the i.i.d. assumption between source and target domains no longer holds. The DG generalisation is evaluated as the models' performance on the unseen target domains and from the DG theoretical analysis (Hu et al., 2020; Muandet et al., 2013; Blanchard et al., 2021; Rosenfeld et al., 2022) scenario, the expected generalisation of target domains is estimated based on the number of source domains, the similarity among source domains and domain shift between source and target domains with the assumptions spanning from model types to the relation between the seen source domains and the unseen target domain.

In the meta-learning setting, generalisation is reflected by the meta-test performance of meta-representation. Its analysis provides bounds for the meta-test error by assuming that the meta-training and meta-test tasks follow i.i.d. learning. Tripuraneni et al. (2021) analyse a guarantee of multi-tasks linear regression model's transferability. The data depended meta-generalisation bounds are proposed by Chen et al. (2021b) for both conventional meta-learning models (Amit and Meir, 2018) and MAML (Finn et al., 2017). Balcan et al. (2019) study the meta-generalisation bound through the regret bound of online convex optimisation. Our works are under the meta-learning umbrella and mainly focus on the o.o.d. task distribution setting. For example, the meta-representation is learned and deployed on the different datasets and neural architectures which are not identical between meta-train and meta-test. As a consequence, the learned meta-representation suffers from the domain shift between task distributions and degrades the meta-generalisation performance. This type of problem exists ubiquitously in real-world

machine learning applications, however, the meta-representation is still expected to be agnostic to the datasets and architectures in the meta-test (deployment) stage. To achieve this, ideally, there should be generalisation analysis providing guidance to construct efficient meta-training protocols and estimate meta-generalisation, but unfortunately, they are not well-developed. Meanwhile, the o.o.d. meta-learning setting has been studied only from an empirical perspective. For example, Lee et al. (2022) proposed to learn the neural network initialisations which can be deployed to o.o.d. datasets. The learned meta-representation, Metaperturb (Ryu et al., 2020), is generalised across datasets and architecture. In our works, the loss function, learned by ARL and ITL for noisy label and domain, are generalised to the unseen dataset and architecture in the meta-train stage and the well-trained MetaMD optimiser can be applied to the novel dataset without any adaptations.

2.5 Summary

In this section, we have discussed all the required techniques for the meta-optimisers in our work. In order to pave the road to the later chapters, we align them with the corresponding meta-objective and meta-knowledge for the applications in each main chapter and give a pre-summary from the meta-learning algorithm design perspective in Table 2.1.

Table 2.1: Pre-summary of main chapters: a view from meta-learning algorithm design

Chapter/Method	Application	Meta Objective ($\mathcal{L}^{outer}$)	Meta-Knowledge (ω)	Meta-Optimiser
Chapter 3: ARL	Label Noise Robust learning	Accuracy after learning on noisy train data	Taylor Loss	CMA-ES
Chapter 4: ITL	Domain-shift Robust Models	Accuracy on held-out validation domains	Taylor Loss	IFT+Hypergradient
Chapter 5: ARL	Fast Convergence	Convergence Rate	Bregman Divergence	IFT+Hypergradient

Chapter 3

Searching for Robustness Loss Learning for Noisy Classification Tasks

This chapter corresponds to the paper:
Searching for Robustness: Loss Learning for Noisy Classification Tasks.
Boyan Gao, Henry Gouk, and Timothy Hospedales, *International Conference on Computer Vision (ICCV)*, 2021.

A motivation for this thesis is to work toward an understanding of multi-task offline meta-learning in transferable loss function learning and optimiser learning. This chapter starts by learning a loss function robust against label noise in the training data. Label noise is inevitable in real-world deep learning applications. The over-parameterised neural networks are able to memorise all training samples and introducing small noise in the training data can improve the generalisation performance of the model. However, when the noise ratio of training data is significant, the learned knowledge from the wrong label data will harm the models' performance in the test stage. We explore the design of the meta-training protocol and carefully select the parameterisation of the potential loss function in order to achieve meta-knowledge generalisation. To further enable the transferability of the learned loss functions, instead of containing a single task in the inner loop optimisation, a multi-task setting is integrated to prevent meta-overfitting. At the time of this study, work on loss function learning was mainly limited to learning auxiliary losses that were added to a conventional hand-designed inner loop loss, e.g. $\mathcal{L}^{inner} = \mathcal{L}^{CE} + \mathcal{L}_{\omega}$ (Li et al., 2019d). Such auxiliary losses were optimised to provide regularisers that enabled the model to better satisfy the other loop loss (Wang

et al., 2020; Li et al., 2019d,a). The original loss function (e.g., Cross Entropy) was not something that was usually learned. As such they could arguably be better thought of as regularising learning, rather than loss function learning. In this work, we learn the main inner loop loss itself.

More concretely, we present a "learning to learn" approach for discovering white-box classification loss functions that are robust to label noise in the training data. We parameterise a flexible family of loss functions using Taylor polynomials and apply evolutionary strategies to search for noise-robust losses in this space. To learn reusable loss functions that can apply to new tasks, our fitness function scores their performance in aggregate across various training datasets and architectures. The resulting white-box loss provides a simple and fast "plug-and-play" module that enables effective label-noise-robust learning in diverse downstream tasks, without requiring a particular training procedure or network architecture. The efficacy of our loss is demonstrated on a variety of datasets with both synthetic and real label noise, where we compare favourably to prior work.

3.1 Introduction

The success of modern deep learning is predicated on large amounts of accurately labelled training data. However, training with large quantities of gold-standard labelled data is often not achievable. This is because professional annotation is often too costly to achieve at scale and so machine learning practitioners resort to less reliable crowd-sourcing, web-crawled incidental annotations (Chen and Gupta, 2015), or imperfect machine annotation (Kuznetsova et al., 2020); while in other situations the data is hard to classify reliably even by human experts, and thus label-noise is inevitable. These considerations have led to a large body of work focusing on developing noise-robust learning approaches (Ren et al., 2018; Han et al., 2018). Diverse solutions have been studied including those that modify the training algorithm through teacher-student (Jiang et al., 2018; Han et al., 2018) learning, or identify and down-weight noisy instances (Ren et al., 2018). Much simpler, and therefore more widely applicable, attempt to define noise-robust loss functions that provide drop-in replacements for standard losses such as cross-entropy (Wang et al., 2019; Zhang and Sabuncu, 2018; Ghosh et al., 2017). These studies hand engineer robust losses, motivated by different considerations including risk minimisation (Ghosh et al., 2017) and information theory (Xu et al., 2019). In this

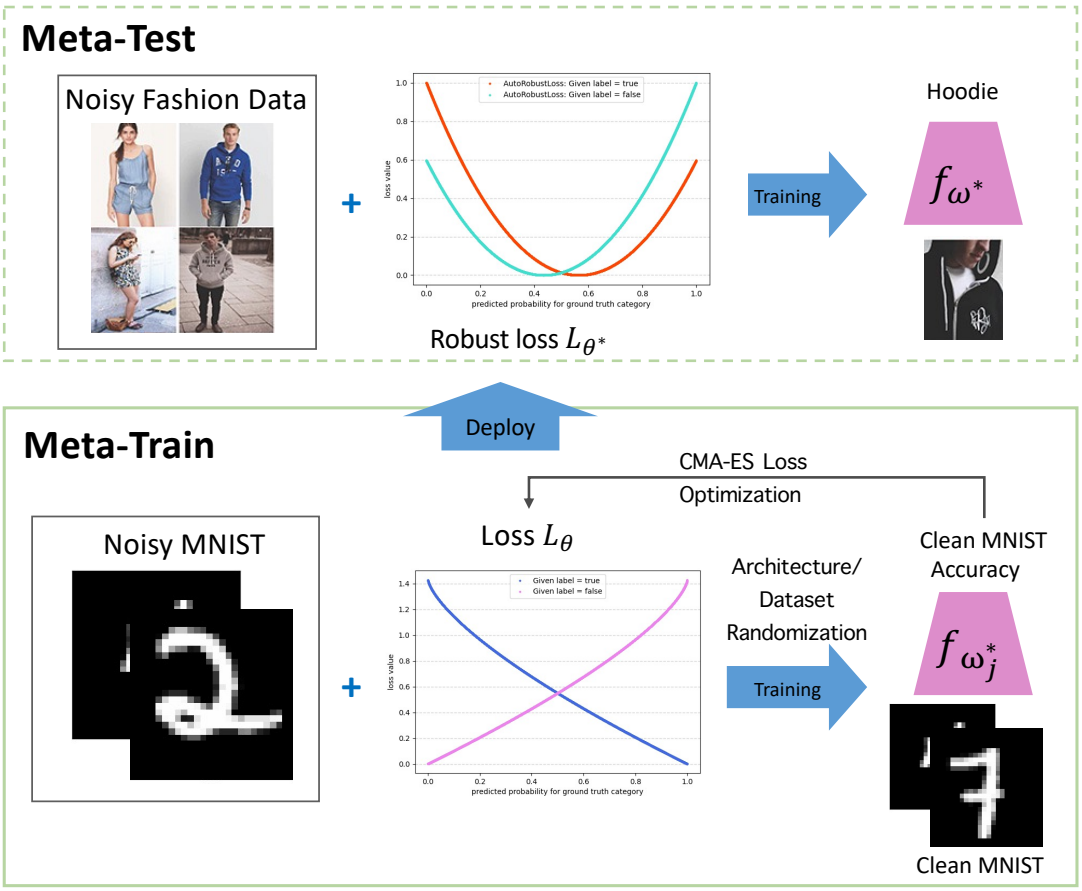

Figure 3.1: Schematic of our robust loss search framework. (1) We train a robust loss function so as to optimise the validation performance of a CNN trained with synthetic label noise using this loss. (2) Thanks to dataset and architecture randomisation, our is reusable and can be deployed to new tasks, including those without clean validation set to drive robust learning.

paper, we explore an alternative data-driven AutoML (Hutter et al., 2019) approach to loss design, and search for a simple white-box function that provides a general-purpose noise-robust drop-in loss. While AutoML approaches have been widely (Real et al., 2019; Elsken et al., 2019) and successfully (Tan and Le, 2019) applied to general purpose neural architecture search (NAS), their application to the discovery of reusable losses is much less widely studied.

We perform an evolutionary search on a space of loss functions parameterised as Taylor polynomials. Every function in this space is smooth and differentiable, and thus provides a valid loss that can be easily plugged into existing deep learning frameworks. Meanwhile, this search space provides a good trade-off between the flexibility to

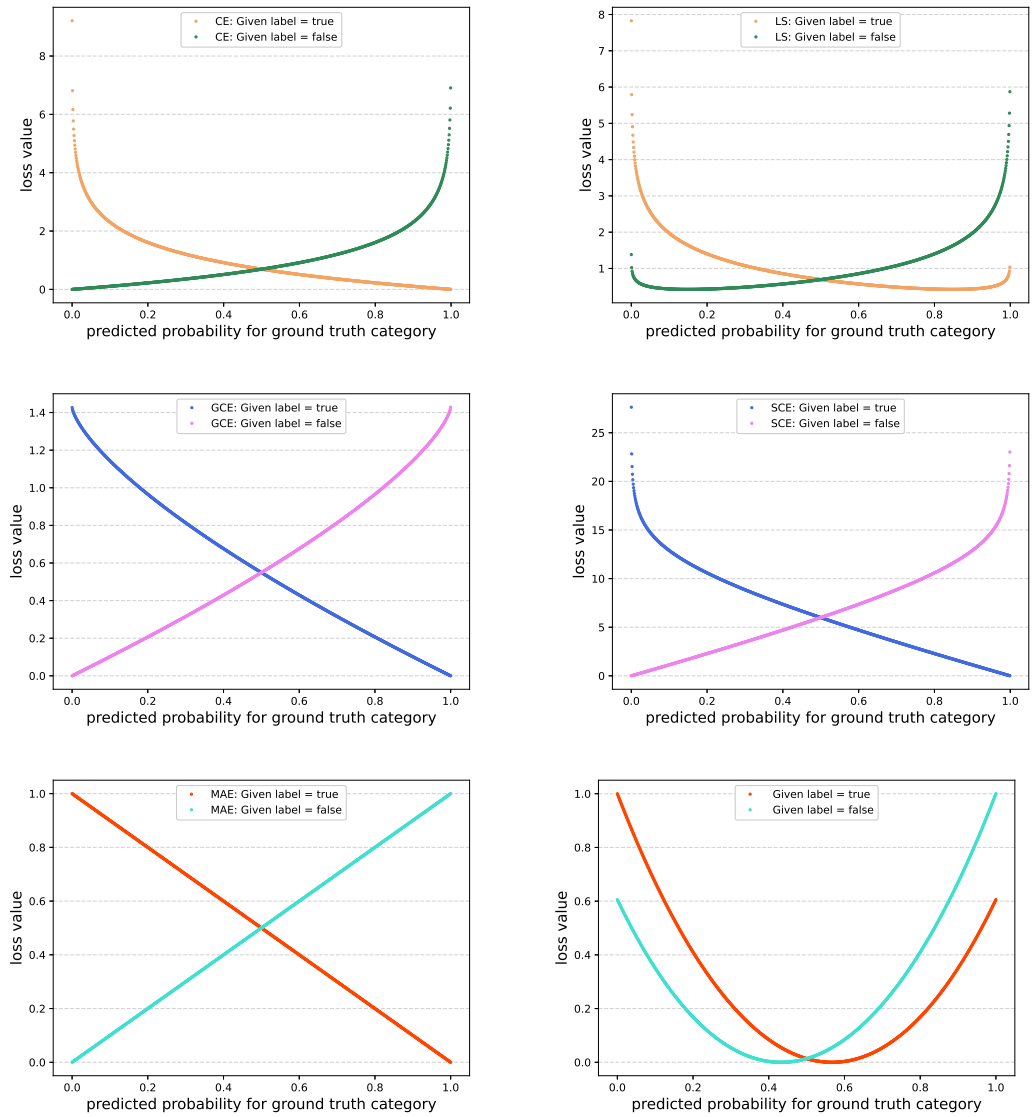

Figure 3.2: Existing hand-designed robust losses and our meta-learned robust loss. Top left: Conventional Cross-Entropy (CE); Top right: label-smoothing (Pereyra et al., 2017). Middle left: Generalised Cross Entropy (GCE) (Zhang and Sabuncu, 2018); Middle right: Symmetric Cross Entropy (SCE) (Wang et al., 2019). Bottom left: Mean Absolute Error (MAE) (Ghosh et al., 2017); Bottom right: Our learned ARL.

represent non-trivial losses and a low-dimensional white-box parameterisation that is efficient to search and reusable across tasks without overfitting. To score a given loss during our search, we use it to train neural networks on noisy data, and then evaluate the clean validation performance of the trained model. To learn a general purpose loss, rather than one that is specific to a given architecture or dataset, we explore domain randomisation (Tobin et al., 2017) in the space of architectures and datasets.

Scoring losses according to their validation performance in diverse conditions leads to reusable functions that can be applied to new datasets and architectures, as illustrated in Figure 3.1.

We apply our learned ARL to train various MLP and CNN architectures on several benchmarks including MNIST, FashionMNIST, USPS, CIFAR-10, and CIFAR-100 with different types of simulated label noise. We also test our loss on a large real-world noisy label dataset, Clothing1M. The results verify the re-usability of ARL and its efficacy compared to state-of-the-art in a variety of settings. This means that, analogously to CNNs discovered by NAS (Real et al., 2019; Tan and Le, 2019), readers are free to use our loss on new noisy problems with no further complicated or expensive AutoML required. This is an important distinction and major advantage of our approach compared to previous work that uses AutoML or meta-learning techniques to perform noise-robust learning (Ren et al., 2018; Shu et al., 2019). These methods often require (i) expensive meta-learning on a per-problem basis, and (ii) a clean (i.e., noiseless) validation dataset to use as a meta-supervision signal, which may not be available in real applications. In contrast, our ultimate contribution is a general-purpose loss (Figure 3.2, bottom right) that provides a simple and fast drop-in replacement for conventional losses (such as cross-entropy) in a standard learning pipeline; and furthermore, no clean validation set is required to use it.

3.2 Related Work

Learning with Label Noise Learning with label noise is now a large research area due to its practical importance. Song (Song et al., 2020) present a detailed survey explaining the variety of existing approaches including designing noise robust neural network architectures (Chen and Gupta, 2015), regularisers such as label-smoothing (Szegedy et al., 2016; Pereyra et al., 2017), sample selection methods that attempt to filter out noisy samples – often by co-teaching or student teacher learning with multiple neural networks (Jiang et al., 2018; Han et al., 2018; Wei et al., 2020; Li et al., 2019c), various meta-learning approaches that often aim to down-weight noisy samples using meta-gradients from a validation set (Ren et al., 2018; Shu et al., 2019; Yao et al., 2020), and robust loss design. Among these families of approaches, we are motivated to focus on robust loss design due to simplicity and general applicability – we wish to provide a loss that can be widely used together with standard architectures

and standard learning algorithms.

Major existing robust losses include: mean absolute error (MAE), shown to be theoretically robust in (Ghosh et al., 2017), but hard to train in (Zhang and Sabuncu, 2018); generalised cross-entropy (GCE) which attempts to be robust yet easy to train (Zhang and Sabuncu, 2018); bi-temper (Amid et al., 2019a), two-temperature (Amid et al., 2019b), and Huber (Hastie et al., 2009) motivated by heavy-tailed outlier robustness; symmetric cross-entropy (Wang et al., 2019) motivated by reducing overfitting; and active-passive loss (APL) (Ma et al., 2020) which aims to balance over-and-underfitting for robust losses. These losses are all hand-designed based on various good motivations, but (as we will see in our evaluation) none provide reliably high performance empirically. Instead, we take a data-driven AutoML approach and search for a loss function that is empirically robust across various benchmark and neural architectures. This draws upon meta-learning techniques but, differently from existing meta-robustness work, focuses on discovering a general white box loss that can be reused in any downstream problem (Figure 3.1), unlike others (Shu et al., 2019; Ren et al., 2018; Li et al., 2019c; Yao et al., 2020) that require expensive per-problem meta-learning. Incidentally, we note that our final loss covers all six desiderata for noise-robust learning outlined in (Song et al., 2020).

Meta-learning, AutoML and Loss Learning Meta-learning, aka learning to learn, and AutoML have been applied for a wide variety of purposes as summarised in (Hospedales et al., 2021; Hutter et al., 2019). Of particular relevance is meta-learning of loss functions, which has been studied for various purposes including providing differentiable surrogates of non-differentiable objectives (Huang et al., 2019), optimising efficiency and asymptotic performance of learning (Jenni and Favaro, 2018; Bechtle et al., 2020; Houthooft et al., 2018; Wu et al., 2018a; Gonzalez and Miikkulainen, 2020a, 2021), and improving robustness to train/test domain-shift (Balaji et al., 2018; Li et al., 2019d). We are interested in learning *white-box* losses – i.e., those that can be expressed in a short human-readable parametric equation – for efficiency and improved task-transferability compared to neural network alternatives (Bechtle et al., 2020; Houthooft et al., 2018; Balaji et al., 2018; Li et al., 2019d), which tend to be less interpretable and need to be learned task-specifically. Meta-learning of white-box model components has been demonstrated for optimisers (Wichrowska et al., 2017), activation functions (Ramachandran et al., 2018), neural architectures (Tan and Le, 2019) and losses for accelerating conventional supervised learning (Gonzalez and Miikkulainen,

2020a, 2021). We are the first to demonstrate the value of automatic loss function discovery for general purpose label-noise robust learning.

3.3 Method

We aim to learn a loss function for multi-class classification that is robust to noisy labels in the training set.

Overview Our workflow has two phases (Figure 3.1). **Meta-train:** Given a set of auxiliary dataset(s) and network architecture(s), we meta learn a label-noise robust loss $\mathcal{L}_\theta$. The auxiliary datasets are assumed to either be clean (in which case we simulate label noise during meta-training), or noisy but come with clean validation sets. The loss $\mathcal{L}_\theta$ should produce models with high performance on the clean validation set(s) after learning on noisy training sets. **Meta-test:** Given the learned robust loss $\mathcal{L}_\theta$ from the previous step, denoted AutoRobustLoss (ARL), we can deploy it to learn any target noisy-label learning problem without requiring a validation set. The target dataset and neural architecture need not overlap with the source dataset/architecture from the meta-train step. The loss provides a drop-in replacement for standard cross-entropy in a conventional learning pipeline.

3.3.1 Meta-Training Procedure

We formalise loss function learning as a bilevel optimisation with an upper/outer loop problem defined as optimising the parameters of an adaptive loss function $\mathcal{L}_\theta$, and a lower/inner loop problem of training neural networks f_ω using the loss function $\mathcal{L}_\theta$. The upper-level optimisation problem uses as a supervision signal for the clean validation performance of models trained with the prospective loss function, averaged across a variety of domains. The lower level optimisation problem consists of learning a collection of neural networks f_ω on noisy-label datasets using the prospective loss function $\mathcal{L}_\theta$. The prospective loss functions are represented by their parameters, θ, which correspond to the coefficients of an n-th order polynomial. These polynomials can be viewed as a Taylor expansion of the ideal loss function. The bilevel optimisation problem is given by

$$\max_{\theta} \mathbb{E}_{D,f}[\mathcal{M}(f_{\omega_D^*}, D^{val})] \tag{3.1}$$
$$s.t. \quad \omega_D^* = \arg\min_{\omega} \mathcal{L}_\theta(f_\omega, D^{train}),$$

Algorithm 6 Robust Loss Function Search

Input: $\mathcal{D}, F, \mu^{(0)}, \Sigma^{(0)}$
Output: $p(\theta; \mu^*, \Sigma^*)$
$t = 0$
while not converged or reached max steps **do**
  $\Theta = \{\theta_1, \theta_2, ..., \theta_n\} \sim p(\theta; \mu^{(t)}, \Sigma^{(t)})$ {Sample losses for exploration}
  $G = F \times \mathcal{D} \times \Theta$ {Assign datasets and architectures to losses}
  $\mathbf{s} = \text{zeros} \in \mathbb{R}^n$
  for all $(f^{(k)}, D_j, \theta_i) \in G$ **do**
    $(D_j^{train}, D_j^{val}) = \text{split}(D_j)$ {Train/Val splits}
    $\omega^* = \arg\min_\omega \mathcal{L}_{\theta_i}(f_\omega^{(k)}, D_j^{train})$ {Train the net}
    $\mathbf{s}_i = \mathbf{s}_i + \frac{1}{|F||D|} \mathcal{M}(f_{\omega^*}^{(k)}, D_j^{val})$ {Evaluate on validation data}
  end for
  $(\mu^{(t+1)}, \Sigma^{(t+1)}) = \text{CMA-ES}(\mu^{(t)}, \Sigma^{(t)}, \Theta, \mathbf{s})$ {Update μ and Σ according to CMA-ES}
  $t = t + 1$
end while

where $\mathcal{M}(\cdot,\cdot)$ is a fitness function measuring network performance, D is a random variable representing a domain, with noisy training D^{train} and clean validation D^{val} splits, and f is a neural network parameterised by ω. The performance of $f_{\omega_D^*}$, as measured by $\mathcal{M}$, reflects the quality of robust supervision provided by the candidate loss $\mathcal{L}_\theta$ on dataset D. We use the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) (Hansen and Ostermeier, 1996) to solve the upper layer problem, and standard stochastic gradient-based optimisation approaches to solve the lower level problems. Algorithm 6 summarises our algorithm for solving the optimisation in Equation 3.1.

CMA-ES for Loss Function Learning We use CMA-ES to solve the upper optimisation problem and any variant of stochastic gradient descent for the lower problem. CMA-ES finds a Gaussian distribution defined over the search space of θ that places most of its mass on high-quality solutions to the optimisation problem. A benefit of using CMA-ES is that it does not require the performance measurement $\mathcal{M}$ to be differentiable, which means the learned loss function can be evaluated using informative metrics, such as accuracy. Each generation consists of a set, Θ, of loss functions obtained

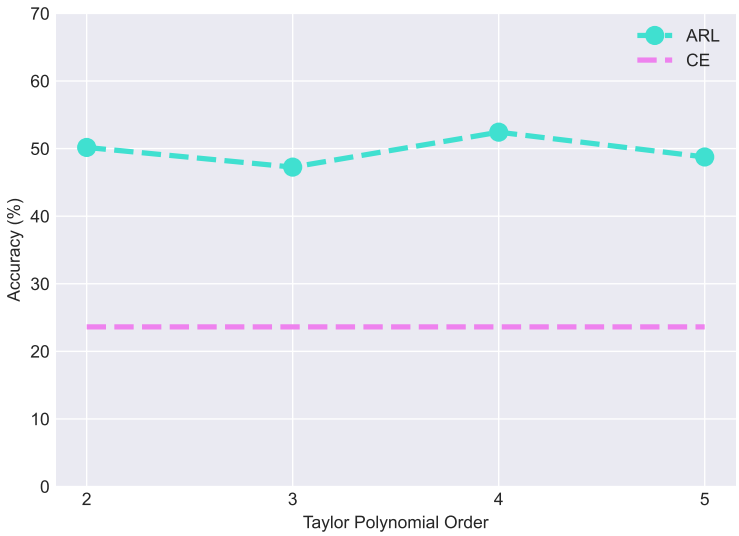

Figure 3.3: A preliminary experiment on hyperparameter selection. The performance of a linear model trained by the ARL loss function with different orders vs training with cross-entropy (CE).

by sampling multiple individuals from the parameter distribution, $p(\theta;\mu,\Sigma) = \mathcal{N}(\mu,\Sigma)$. Each of the individuals, $\theta_i \in \Theta$, is evaluated according to

$$\mathbb{E}_{D,f}[\mathcal{M}(f_{\omega_D^*}, D^{val})] \approx \frac{1}{N}\sum_{j=1}^{N} \mathcal{M}(f_{\omega_j}^{(j)}, D_j^{val}) \tag{3.2}$$
$$\text{s.t.} \quad \omega_j = \arg\min_{\omega} \mathcal{L}_{\theta_i}(f_{\omega}^{(j)}, D_j^{train}),$$

where $f_{\omega}^{(j)}$ and D_j are different network architectures and datasets respectively, as discussed later. We apply a scale normalisation, detailed in Appendix A.3, to the final loss function.

Taylor Polynomial Representation The space of potential loss functions in which CMA-ES searches is a crucial design parameter. For search efficiency, we should consider a space parameterised by a small number of values. This must be balanced with the ability to represent a wide enough variety of functions such that a good solution can be found. By selecting a low-dimensional space with a well-understood nonlinear form, it should be possible to re-use the learned loss on diverse problems. The function space that we choose is the Taylor series approximations of all β-times differentiable

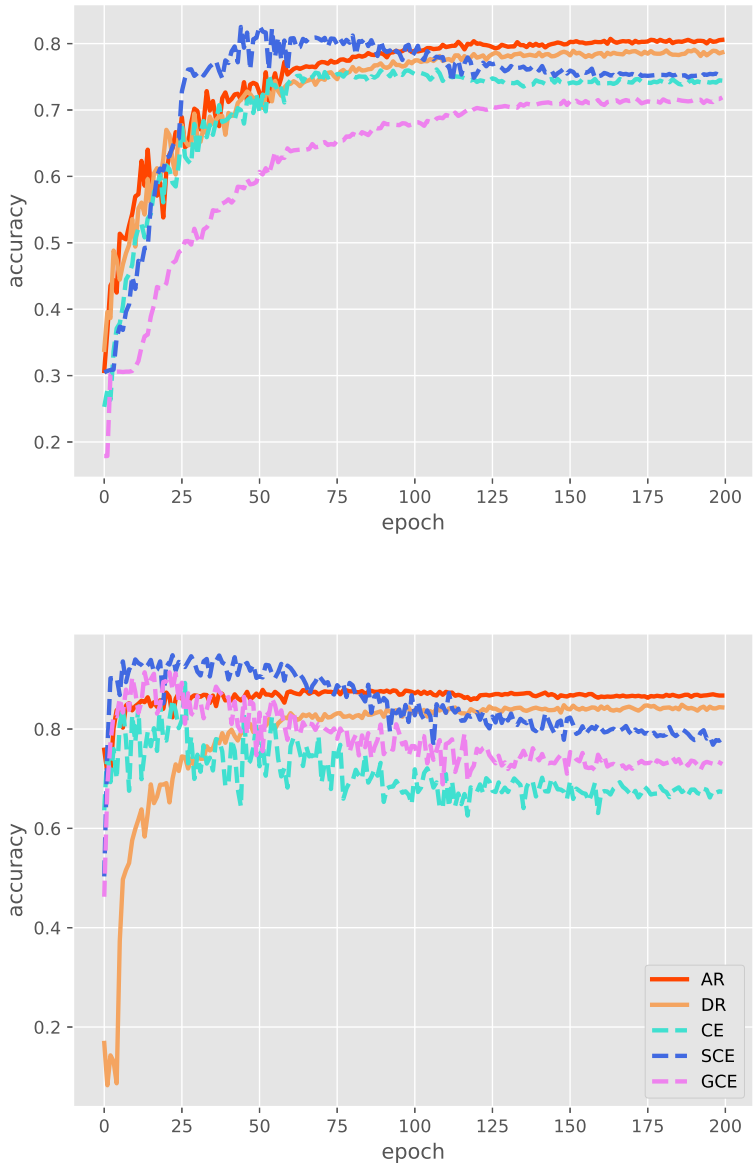

Figure 3.4: Example learning curves of test accuracy vs iterations when using different robust losses. Top: USPS/VGG-11/80% symmetric noise. Bottom: USPS/ResNet-18/40% asymmetric noise.

functions (Gonzalez and Miikkulainen, 2021), $g : \mathbb{R}^m \rightarrow \mathbb{R}$,

$$g(\mathbf{x}) = \sum_{n=0}^{\beta} \frac{1}{n!} \nabla^n g(\mathbf{x}_0)^T (\mathbf{x} - \mathbf{x}_0)^n. \tag{3.3}$$

where each $\nabla^n g(\mathbf{x}_0)$ is the n-th order gradient of g evaluated at a fixed point, $\mathbf{x}_0$. We make the simplifying assumption that the loss function should be class-wise separable. That is, each potential class is considered in isolation, and we learn a loss function that

measures the divergence between a noisy binary label and the probability predicted by the network. To compute the loss on vectors we sum over the C possible classes,

$$\mathcal{L}_\theta(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{C}\sum_{i=1}^{C} \ell_\theta(\hat{\mathbf{y}}_i, \mathbf{y}_i), \tag{3.4}$$

where $\hat{\mathbf{y}}$ and $\mathbf{y}$ are the vectors of predicted probabilities and (possibly noisy) ground-truth labels, respectively. The result of performing this simplification is that the loss function can be used in a variety of settings with different numbers of classes. We found that $\beta = 4$ is a good trade-off between modelling capacity and meta-training efficiency. In a Taylor expansion the polynomial coefficients are given by the fixed point around which the Taylor expansion is being evaluated and the gradients of the function at this fixed point. Hence, for learning a bi-variate function we say that (θ_0, θ_1) gives the location of the fixed point, and $(\theta_2, ..., \theta_{11})$ encode the values of the gradients of the optimal loss function when it is evaluated at (θ_0, θ_1). The resulting loss has the form

$$\begin{aligned}
\ell_\theta(\hat{\mathbf{y}}_i, \mathbf{y}_i) = {} & \theta_2(\hat{\mathbf{y}}_i - \theta_0) + \frac{1}{2}\theta_3(\hat{\mathbf{y}}_i - \theta_0)^2 \\
& + \frac{1}{6}\theta_4(\hat{\mathbf{y}}_i - \theta_0)^3 + \frac{1}{24}\theta_5(\hat{\mathbf{y}}_i - \theta_0)^4 \\
& + \theta_6(\hat{\mathbf{y}}_i - \theta_0)(\mathbf{y}_i - \theta_1) \\
& + \frac{1}{2}\theta_7(\hat{\mathbf{y}}_i - \theta_0)(\mathbf{y}_i - \theta_1)^2 + \frac{1}{2}\theta_8(\hat{\mathbf{y}}_i - \theta_0)^2(\mathbf{y}_i - \theta_1) \\
& + \frac{1}{6}\theta_9(\hat{\mathbf{y}}_i - \theta_0)^3(\mathbf{y}_i - \theta_1) + \frac{1}{6}\theta_{10}(\hat{\mathbf{y}}_i - \theta_0)(\mathbf{y}_i - \theta_1)^3 \\
& + \frac{1}{4}\theta_{11}(\hat{\mathbf{y}}_i - \theta_0)^2(\mathbf{y}_i - \theta_1)^2.
\end{aligned} \tag{3.5}$$

Note that we have omitted terms where $\hat{\mathbf{y}}$ does not appear, as these do not impact the solution of the optimisation problem. In total there are only 12 parameters to fit, which is considerably smaller than the number of parameters found in a typical neural network parameterised loss function (Li et al., 2019d; Bechtle et al., 2020; Kirsch et al., 2020).

Generalisation Across Architectures To enable achieving good generalisation to novel architectures in deployment (meta-testing), we apply the domain randomisation (Tobin et al., 2017) strategy to evaluate the expected performance across a range of architectures during meta-training. Specifically, we use a set of architectures, F, containing a variety of common neural network designs. The total population for evolutionary optimisation is then given by the Cartesian product $F \times \Theta$. The fitness function can then be computed as shown in Equation 3.2, where a mean is taken over all different architectures trained with the same loss.

Table 3.1: Accuracy (%) of robust losses, 80% symmetric noise condition. Our loss trained under architecture randomisation (AR) and dataset randomisation (DR) conditions has the best average rank. Grey cols: datasets seen during DR training. White cols: totally novel datasets.

Architecture type dataset	VGG11 Cifar10	VGG11 Cifar100	VGG11 FashionMNIST	VGG11 USPS	ResNet18 Cifar10	ResNet18 Cifar100	ResNet18 FashionMNIST	ResNet18 USPS	Avg.Rank
CE	18.38 ± 0.21	4.25 ± 0.28	20.55 ± 0.93	51.42 ± 0.94	18.44 ± 0.34	8.86 ± 0.10	21.92 ± 0.74	57.05 ± 0.42	7.75
GCE (Zhang and Sabuncu, 2018)	16.56 ± 0.54	1.04 ± 0.47	25.10 ± 0.68	63.45 ± 0.86	31.69 ± 0.36	11.98 ± 0.18	42.62 ± 0.89	$\mathbf{79.52 \pm 0.63}$	6.25
SCE (Wang et al., 2019)	28.61 ± 0.64	2.31 ± 0.80	36.64 ± 0.59	63.68 ± 0.56	$\mathbf{45.34 \pm 0.40}$	8.16 ± 0.07	59.93 ± 0.75	58.35 ± 0.76	5.25
FW (Patrini et al., 2017)	16.97 ± 0.44	1.41 ± 0.07	22.57 ± 0.76	53.66 ± 0.40	10.15 ± 0.68	1.16 ± 0.04	13.18 ± 0.35	42.80 ± 0.77	10.38
Bootstrap (Reed et al., 2015)	17.58 ± 0.82	4.18 ± 0.72	20.40 ± 0.31	64.58 ± 0.21	12.10 ± 0.32	8.67 ± 0.61	22.36 ± 1.76	72.17 ± 1.24	7.25
MAE (Ghosh et al., 2017)	14.20 ± 0.42	1.01 ± 0.11	63.40 ± 0.16	30.94 ± 0.35	22.95 ± 1.25	0.82 ± 0.17	68.20 ± 1.87	37.17 ± 0.93	9.00
Label-smooth (Pereyra et al., 2017)	17.74 ± 0.46	4.47 ± 0.12	21.19 ± 0.39	54.26 ± 0.19	17.67 ± 0.35	7.66 ± 1.52	20.99 ± 0.83	59.94 ± 0.54	7.75
Huber (Hastie et al., 2009)	10.28 ± 0.68	1.30 ± 0.57	19.66 ± 0.67	23.92 ± 1.34	13.56 ± 0.75	1.14 ± 1.11	17.59 ± 0.91	24.61 ± 0.31	12.00
NCE+MAE (Ma et al., 2020)	40.47 ± 0.93	2.06 ± 0.44	48.40 ± 1.01	70.75 ± 0.71	33.57 ± 1.17	5.72 ± 0.92	48.65 ± 0.96	71.25 ± 1.24	5.00
NFL+MAE (Ma et al., 2020)	41.91 ± 0.98	2.54 ± 0.63	45.06 ± 1.06	69.36 ± 0.84	37.66 ± 0.64	6.03 ± 0.91	54.43 ± 1.15	72.25 ± 1.60	4.00
Bi-Temper (Amid et al., 2019a)	10.44 ± 0.96	3.23 ± 0.11	15.00 ± 0.46	17.67 ± 0.56	40.41 ± 1.33	9.35 ± 0.52	30.06 ± 0.72	26.91 ± 0.64	8.50
Baikal (Gonzalez and Miikkulainen, 2020b)	27.16 ± 0.32	4.73 ± 0.64	68.81 ± 0.71	65.47 ± 0.39	23.79 ± 1.08	8.87 ± 0.76	63.49 ± 0.81	48.43 ± 0.82	4.38
ARL-AR	$\mathbf{41.36 \pm 0.47}$	$\mathbf{5.63 \pm 0.24}$	$\mathbf{70.16 \pm 0.87}$	$\mathbf{78.71 \pm 0.90}$	29.50 ± 0.30	$\mathbf{14.94 \pm 0.26}$	71.96 ± 0.89	68.80 ± 0.92	**1.63**
ARL-DR	31.12 ± 0.23	5.04 ± 0.14	67.29 ± 1.01	77.34 ± 1.34	35.23 ± 0.23	13.36 ± 0.63	$\mathbf{71.97 \pm 0.87}$	70.17 ± 0.64	1.88

Generalisation Across Datasets The learned loss should also generalise to novel datasets in deployment (meta-testing). To this end, we investigate exposing it to several datasets during training, so as to ensure it is maximally agnostic to the specific training dataset. Sampled loss functions are used to train several models with the same architecture and initial weights, but on different datasets. Similarly to architecture generalisation, we use a set of datasets, $\mathcal{D}$, and take the Cartesian product, $\mathcal{D} \times \Theta$, to generate a population to be evaluated. The performance of the loss functions is evaluated by the mean performance of all the networks on their corresponding datasets. In principle, one can perform dataset and architecture randomisation simultaneously. However, due to the implied three-way Cartesian product, we found this computationally infeasible.

3.4 Experiments

In this section, we evaluate ARL on various noisy label learning tasks. In particular, we aim to answer three questions: (Q1) Does our AutoRobustLoss (ARL generalise across different datasets and architectures? (Q2) How well does ARL generalise across different noise levels? (Q3) Can ARL scale to larger real-world noisy-label tasks?

Datasets We experiment on seven datasets: MNIST (LeCun and Cortes, 2010), CIFAR-10, CIFAR-100 (Krizhevsky, 2009), KMNIST (Clanuwat et al., 2018), USPS (Hull, 1994), FashionMNIST (Xiao et al., 2017) and Clothing1M (Xiao et al., 2015). Cloth-

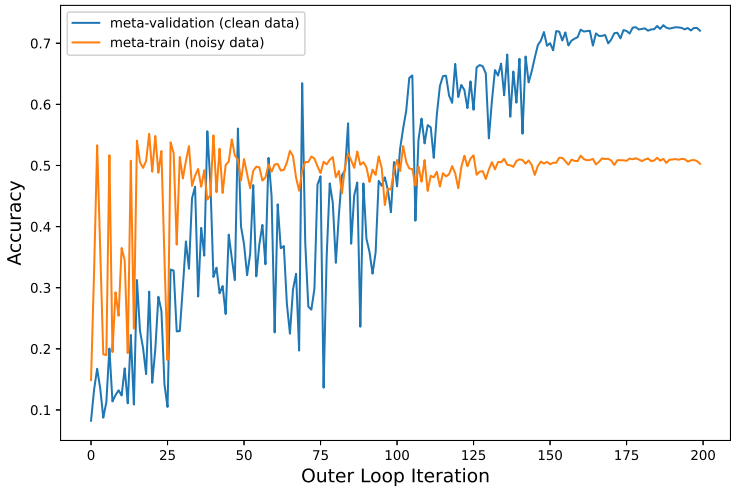

Figure 3.5: The convergence process of loss function meta-learning in terms of (noisy data) meta-training accuracy and (clean data) meta-validation accuracy.

ing1M is a dataset containing 1 million clothing images in 14 classes (T-shirt, Shirt, Knitwear, Chiffon, Sweater, Hoodie, Windbreaker, Jacket, Down Coat, Suit, Shawl, Dress, Vest, Underwear). The images are collected from shopping websites and the labels are generated from the text surrounding images, thus providing a realistic noisy label setting. MNIST, and optionally KMNIST and CIFAR-10, are used for learning the loss function (meta-training), and the others are completely held out for experimental evaluation (meta-testing).

Noise types For loss learning, we simulate both symmetric and asymmetric (pair-flip) noise types. Symmetric noisy labels are generated by uniformly flipping from a positive label to a negative one, while asymmetric noisy labels are produced to simulate the more realistic scenario where particular pairs of categories are more easily confused than others by annotators. For example, among digits label noise could manifest in such a way that a 7 is more likely to be mislabelled as a 1 than as a 6; or a 3 is mislabelled as an 8 rather than a 4.

Architectures We train and evaluate ARL with a range of neural networks including shallow (2-layer MLP, 3-layer MLP, and 4-layer CNN) and deep (VGG-11 (Simonyan and Zisserman, 2015) and ResNet-18 (He et al., 2016)). We also use the medium-size architecture in (Wei et al., 2020), which we term JoCoR-Net (see supplemental for details). For a fair comparison, we train 2-layer MLP, 3-layer MLP, and 4-layer CNN

Table 3.2: Accuracy (%) of robust losses. 40% asymmetric noise condition. Our loss trained under architecture (AR) and dataset (DR) randomisation conditions has the best average rank. Grey cols: datasets seen during DR training. White cols: totally novel datasets.

Architecture type dataset	VGG11 Cifar10	VGG11 Cifar100	VGG11 FashionMNIST	VGG11 USPS	ResNet18 Cifar10	ResNet18 Cifar100	ResNet18 FashionMNIST	ResNet18 USPS	Avg.Rank
CE	56.43±0.12	30.20±0.18	50.34±1.23	77.74±0.74	58.69±0.43	44.14±0.15	58.68±0.63	73.84±0.85	7.00
GCE (Zhang and Sabuncu, 2018)	56.42±0.54	22.39±0.35	53.57±0.47	78.72±0.72	57.90±0.31	40.76±0.24	58.51±0.70	80.77±0.35	7.13
SCE (Wang et al., 2019)	78.23±0.55	25.33±0.73	64.47±0.97	85.50±0.43	63.22±0.22	40.90±0.37	59.63±0.96	81.57±0.17	4.25
FW (Patrini et al., 2017)	54.42±0.79	5.21±0.39	45.18±0.84	76.41±0.81	48.40±0.08	3.83±0.23	49.46±0.73	46.04±0.18	11.13
Bootstrap (Reed et al., 2015)	57.69±0.11	**31.07±1.09**	53.23±1.53	77.81±0.61	57.69±0.76	**45.78±0.15**	54.60±0.85	75.67±0.56	6.50
MAE (Ghosh et al., 2017)	49.06±0.22	0.96±0.10	49.02±0.27	62.38±0.89	55.67±3.05	1.02±0.14	56.31±1.21	70.05±0.35	11.50
Label-smooth (Pereyra et al., 2017)	57.76±0.37	20.64±0.18	51.12±1.03	77.49±0.11	59.69±0.36	39.92±0.49	57.53±0.73	78.97±0.46	7.75
Huber (Hastie et al., 2009)	38.28±0.80	5.18±0.72	75.57±0.93	73.44±2.70	56.11±0.41	4.14±0.37	77.50±1.64	79.37±1.55	8.25
NCE+MAE (Ma et al., 2020)	66.22±0.64	2.06±0.36	69.83±0.73	87.05±1.32	60.51±0.96	45.00±0.87	63.00±1.97	81.81±1.31	4.75
NFL+MAE (Ma et al., 2020)	65.55±1.76	2.59±0.17	69.51±1.59	89.24±1.92	62.51±0.83	44.84±1.76	58.55±1.04	82.96±2.82	4.88
Bi-Temper (Amid et al., 2019a)	10.12±0.17	34.22±1.23	18.02±0.87	17.89±0.82	17.74±0.73	45.36±0.43	19.44±1.32	27.85±0.94	9.88
Baikal (Gonzalez and Miikkulainen, 2020b)	68.92 ± 0.36	25.38 ± 0.85	**88.30 ± 0.74**	87.64 ± 0.53	60.49 ± 0.91	37.24 ± 0.28	77.06 ± 1.18	84.71 ± 0.42	3.5
ARL-AR	74.30±0.20	22.50±0.33	87.23±1.22	**90.67±1.21**	**86.70±0.12**	44.47±0.48	**89.24±0.25**	**91.17±0.25**	**1.50**
ARL-DR	**79.09±0.51**	18.30±0.27	81.18±0.80	89.78±0.46	68.88±0.41	31.47±0.65	88.22±0.97	89.59±1.05	3.00

with SGD optimiser, learning rate 0.01 and momentum 0.9. For JoCor-Net, we apply Adam (Kingma and Ba, 2015) with a learning rate 0.001. For ResNet-18 and VGG-11, we follow the training protocol in (Zhang et al., 2019).

Taylor Polynomial Order Selection We perform a preliminary experiment to select the order of ARL. We train a linear classifier in the inner loop of the dataset randomisation algorithm (on MNIST, KMNIST, and CIFAR-10), and evaluate performance for polynomial orders 2, 3, 4 and 5. From the results in Figure 3.3, we can see that the impact of the specific polynomial order is small compared to the impact of loss learning overall. Nevertheless, we pick order 4 for the subsequent experiments, as this was the hyperparameter that achieved the best performance. In addition, different Taylor polynomial orders make different curvature flexibility of the perspective loss function. Compared with TaylorGLO (Gonzalez and Miikkulainen, 2021) whose Taylor polynomial order is 3, our learned loss function in Figure 3.2 is more rounded, which prevents the models overfit to the noisy data.

Competitors We compare our ARL with the standard cross-entropy (CE) baseline, as well as several strong alternative losses hand-designed for label-noise robustness: **MAE:** Mean Absolute Error was theoretically shown to be robust in (Ghosh et al., 2017). **GCE:** (Zhang and Sabuncu, 2018) analysed MAE as hard to train, and proposed generalised cross-entropy to provide the best of CE and MAE; **FW:** (Patrini et al., 2017) iteratively estimates the label noise transfer matrix, and trains the model corrected by the

label noise estimate; **SCE:** (Wang et al., 2019) argued that symmetrising cross-entropy by adding reverse cross-entropy (RCE) improves label-noise robustness; **Bootstrap:** A classic method of replacing the noisy labels in training by the convex combination of the prediction and the given labels (Reed et al., 2015). **LSR:** Label-smoothing is an effective general purpose regulariser (Pereyra et al., 2017; Szegedy et al., 2016; Müller et al., 2019) whose properties in promoting noise robustness have been studied (Wang et al., 2019). **Huber** (Hastie et al., 2009) and **Bi-Temper** (Amid et al., 2019a): Classic and recent approaches based on robust heavy-tailed loss functions. **Active Passive**: The best out of a selection of normalised losses proposed in (Ma et al., 2020). **Baikal**: A loss function discovered by a genetic algorithm on the mathematic symbolic space (Gonzalez and Miikkulainen, 2020b).

Early Stopping and Hyper-parameter Tuning While conventional supervised learning can use early stopping, the lack of a clean validation set during meta-testing makes this impossible. Therefore our main experiments follow the majority of work (Jiang et al., 2018; Wei et al., 2020) in this area by reporting performance at convergence. Similarly, the lack of a clean validation set prevents automated hyperparameter tuning, so we reuse a single set of hyperparameters chosen on the meta-training sets.

3.4.1 Training a general-purpose robust loss function

Meta-Training Setup We consider two domain randomisation strategies for training a general purpose loss function, namely architecture (AR) and dataset (DR) randomisation. In AR, we build a pool of training architectures including 2-layer MLP, 3-layer MLP, and 4-layer CNN and solely use MNIST as the training set. In DR, we solely use the 4-layer CNN as the architecture builds a dataset pool of MNIST, KMNIST, and CIFAR-10. We train models for 80% symmetric, and 40% asymmetric noise conditions. More details are given in the Appendix.

Meta-Testing (Deployment) Setup Given our ARL learned in meta-training, we evaluate it by deploying on a fresh suite of evaluation datasets and architectures including those unseen during training. We report results in terms of accuracy at convergence, and summarise via the average ranks of each loss across different datasets and architectures (Demšar, 2006).

Benchmark Results The results for symmetric and asymmetric noise are shown in

Table 3.3: Accuracy (%) and average rank of different robust losses using noisy-validation based on early stopping and hyper-parameter tuning.

	Architecture type	VGG11	VGG11	VGG11	VGG11	ResNet18	ResNet18	ResNet18	ResNet18	Avg.Rank
	dataset	Cifar10	Cifar100	FashionMNIST	USPS	Cifar10	Cifar100	FashionMNIST	USPS	
Symmetric 80%	CE	41.94	12.12	76.03	75.19	39.80	18.18	72.18	75.88	2.88
	GCE (Zhang and Sabuncu, 2018)	43.94	5.25	74.88	77.68	40.80	15.16	72.50	75.88	3.00
	SCE (Wang et al., 2019)	**48.48**	7.33	72.17	71.95	**45.81**	16.99	75.36	77.91	2.38
	NCE+MAE (Ma et al., 2020)	38.96	2.28	75.63	74.39	20.47	10.33	74.33	**82.21**	3.86
	NFL+MAE (Ma et al., 2020)	43.35	2.61	73.08	73.44	42.92	2.69	70.45	76.83	4.00
	ARL-AR	42.52	**13.71**	71.47	**79.37**	35.86	**20.87**	**77.23**	79.77	**1.88**
	ARL-DR	42.76	7.17	**77.58**	77.58	34.77	17.75	73.28	74.93	3.00
Asymmetric 40%	CE	79.47	29.89	84.07	92.33	82.15	40.36	87.91	89.74	2.86
	GCE (Zhang and Sabuncu, 2018)	77.13	24.30	85.16	88.09	78.02	43.73	87.57	91.63	3.86
	SCE (Wang et al., 2019)	78.39	**29.90**	82.95	91.68	78.98	43.06	87.20	93.47	2.86
	NCE+MAE (Ma et al., 2020)	74.87	7.49	86.81	90.87	80.05	40.89	82.23	90.63	4.38
	NFL+MAE (Ma et al., 2020)	76.61	7.78	86.36	89.79	76.39	36.79	88.42	92.48	4.25
	ARL-AR	76.13	25.37	**88.81**	**94.27**	**87.03**	45.41	89.44	**94.57**	**1.25**
	ARL-DR	**80.82**	29.21	87.14	91.78	76.62	**47.19**	**89.79**	93.17	1.50

Table 3.1 and 3.2 respectively. From the results, we can see that our ARL performs favourably compared to hand-designed alternatives across a variety of benchmarks, with a higher average rank than competitors in both experiments. However, there is no clear winner between architecture (AR) and dataset (DR) randomization for meta-learning. We expect that the best performance would be obtained by performing these simultaneously during meta-training, but as this experiment is computationally costly, we leave this to future work. Note that during deployment, all methods have a similar computational cost, except for FW which requires training the network twice for noise estimation.

Analysis of Learning Curves The plots in Figure 3.4 compare the learning curves of test accuracy for USPS/VGG-11 and USPS/ResNet-18 with 80% symmetric and 40% asymmetric noise respectively. We can see that while some alternative losses have early peaks, they all overfit after continued training. As discussed earlier, the asymptotic performance is the relevant and standard (Jiang et al., 2018; Wei et al., 2020) metric in this area due to the lack of a clean validation set to cherry-pick a good iteration; and on this metric, our losses are clear winners.

Real-world Clothing1M results The previous experiment reported the performance of the learned model after training on manually corrupted labels. In this section, we follow the ResNet-18 setting described in (Wei et al., 2020) to apply our learned loss to the real-world Clothing1M noisy-label benchmark. Note that neither Clothing1M,

Table 3.4: Test accuracy (%) of robust learners on Clothing1M with ResNet18. *JoCoR is a multi-network co-distillation training framework. The others are simple plug-in robust losses.

Method	CE	Bootstrap (Reed et al., 2015)	GCE (Zhang and Sabuncu, 2018)	FW (Patrini et al., 2017)	SCE (Wang et al., 2019)	Huber (Hastie et al., 2009)	JoCoR* (Wei et al., 2020)
Accuracy	66.88	67.28	66.63	68.33	67.63	10.83	69.79
Method	NCE+MAE (Ma et al., 2020)	NFL+MAE (Ma et al., 2020)	Bi-Temper (Amid et al., 2019a)	ARL (AR-A40)	ARL (DR-A40)	ARL (AR-S80)	ARL (DR-S80)
Accuracy	66.15	65.97	9.46	69.14	**70.09**	68.85	69.34

Table 3.5: Accuracy (%) of different robust learners. JoCoR net CNN used throughout. ARL is trained for each target problem.

	Noise Type	CE (Reproduced)	CE (JoCoR)	GCE (Zhang and Sabuncu, 2018)	SCE (Wang et al., 2019)	FW (Patrini et al., 2017)	Bootstrap (Reed et al., 2015)	JoCoR (Wei et al., 2020)	ARL
MNIST	Sym-20%	81.21±0.53	79.56±0.44	97.64±0.65	89.50±0.44	96.85±0.67	76.18±0.98	**98.06±0.04**	97.90±0.12
	Sym-50%	59.51±0.70	52.66±0.43	94.14±1.32	67.38±0.53	94.25±0.43	51.53±1.56	96.64±0.12	**96.71±0.21**
	Sym-80%	22.43±1.21	23.43±0.31	40.57±0.72	31.23±0.89	54.01±1.82	23.46±0.46	84.89±4.55	**89.88±0.34**
	Asym-40%	78.73±1.16	79.00±0.28	81.94±1.22	79.87±0.78	90.14±0.67	78.31±2.34	95.24±0.10	**97.38±0.17**
CIFAR-100	Sym-20%	39.19±0.58	35.14±0.44	34.66±0.76	35.09±0.50	38.18±0.76	3.53±0.18	**53.01±0.04**	51.34±0.10
	Sym-50%	19.50±0.43	16.97±0.40	10.29±0.53	18.54±0.29	3.25±0.15	18.36±0.63	**43.49±0.46**	42.18±0.27
	Sym-80%	5.56±0.24	4.41±0.14	2.03±0.36	5.75±0.39	6.12±0.27	2.33±0.13	15.49±0.98	**20.20±0.42**
	Asym-40%	30.16±0.44	27.29±0.25	1.32±0.23	27.07±0.42	4.23±0.51	31.72±0.74	32.70±0.35	**36.01±0.39**
	Avg.Rank	5.25	6.13	5.62	5.00	4.38	6.63	1.63	**1.38**

nor ResNet-18 were seen during the loss discovery meta-learning, above. We train with Adam using learning rate 8×10^{-4}, 5×10^{-4}, 5×10^{-5} for 5 epochs each. We report the mean accuracy of each model after ten trials in Table 3.4. Among the competitors, JoCoR is the state art method in the broader range of noise-robust learners. It uses a complex co-distillation scheme with multiple network branches, while the other listed competitors are simple plug-in robust losses applied to vanilla ResNet training. Nevertheless, ARL obtains the top performance.

3.4.2 Additional Analysis

Noisy Validation Very recently, the established protocol for noisy-label experiments used in the previous section was challenged in (Chen et al., 2021a), who claim that the metric of validation set *accuracy* provides a valid model-selection criterion, even when the validation set itself contains label noise. Therefore we select the top performing losses from the previous experiment and report their performance under a new deployment condition using both early stopping and hyper-parameter tuning according to this proxy metric.

From the results in Table 3.3, we can see that: (i) Early stopping allows CE to reduce overfitting to noise and hence improve in rank compared to the asymptotic results in Tables 3.1-3.2, but it is still not best; (ii) Most accuracies have increased compared

to the previous condition (e.g., FashionMNIST), but our losses have increased less by comparison, suggesting that AR and DR rely less on careful parameter tuning and checkpoint selection compared to alternatives. (iii) Overall both AR and DR learned losses perform strongly, with AR performing best overall in both noise conditions.

Generalisation across noise-levels We trained our losses on high levels of label noise (80%-symmetric, 40%-asymmetric) as detailed previously, conjecturing that training on a difficult task would be sufficient for generalisation to other tasks with diverse noise conditions, as shown on Clothing1M. To evaluate this more systematically, we next apply our losses to problems with a range of noise levels. From the results in Figure 3.6 we can see that our loss does provide strong performance across a range of operating points. Notably, the leftmost point on each plot corresponds to the clean data (0% noise) condition. Here our ARL losses provide comparable performance to the standard (i.e., cross-entropy) approach, thus confirming that they are safe to use in cases where it is unknown whether label noise is present or not.

Qualitative analysis of representations We visualise the feature distributions learned by the losses when applied to CIFAR-10 under 40% symmetric label noise in Figure 3.7. We can see that conventional CE applied on noisy labels leads to a very mixed distribution of instances, while our loss leads to quite cleanly separable clusters despite the label noise.

Dataset-specific loss learning Our main goal in this paper has been to learn a general purpose robust loss. In this section, we examine an alternative use case of applying our framework to train a *dataset-specific* robust loss, in which case better performance could be achieved by customising the loss for the target problem. To achieve this, we now additionally assume a clean subset of data for the target problem is available (unlike the previous experiments, but similar to several alternative methods in this area (Wei et al., 2020)) in order to drive loss learning. For this experiment, we focus on comparison with JoCoR (Wei et al., 2020), since this is the current state-of-the-art model. We use the same medium-sized CNN architecture as JoCoR for a fair comparison, and train our loss to optimise the validation performance. From the results in Table 3.5, we can see that our ARL provides comparable or better performance than state-of-the-art competitor JoCoR. However, this is now at a significantly greater cost since the cost of data-specific loss training is not amortizable over multiple tasks as before.

Qualitative Analysis and Intuition of Learned Loss To gain some intuition about our loss' efficacy, we compare popular standard and robust losses in Figure 3.2. Comparing our ARL loss against alternatives, we conjecture that there are two properties that account for our label-noise robustness in practice: Feedback in response to perceived major prediction errors by the network, and the location of the minima where network predictions maximally satisfy the loss. In the case of a noisy labelled example that the network actually classifies correctly, (e.g., $y_{true} = 1$, $y_{label} = 0$, $y_{pred} \approx 1$), conventional CE aggressively "corrects" the network by reporting the exponentially large loss. This aggressive feedback leads to fast training on clean data, but overfitting in noisy data (Zhang and Sabuncu, 2018). Existing robust alternatives MAE (Ghosh et al., 2017) and GCE (Zhang and Sabuncu, 2018) are explicitly motivated by softening this aggressive "correction" compared to CE. Although not explicitly motivated by this, SCE also softens the feedback as shown in the figure. Meanwhile, in terms of the minima that best satisfies the loss, conventional CE, as well as SCE, GCE and MAE lead to maximally confident predictions (minima at 0 or 1); which, if applied to a noisy label, leads to overfitting. In contrast, label smoothing (Pereyra et al., 2017; Wang et al., 2019) improves robustness by inducing softer minima at $[0+\varepsilon, 1-\varepsilon]$ compared to the others' $[0,1]$. However, LS issues the same aggressive correction of large errors as CE, and thus suffers from this accordingly. Only our ARL has learned to exploit both these strategies of less aggressive "corrections" and softer targets.

Learning curve We illustrate the convergence of our loss learning framework by its learning curve in Figure 3.5. The curve illustrated corresponds to the dataset randomisation condition where MNIST, KMNIST and CIFAR-10 are used. For simplicity, we apply a shallow network that has two fully connected layers with *m*-256-256-*C* units, where *m* is the dimensionality of the input and *C* is the number of classes. The meta-train line represents the average accuracy of the trained networks across all the noisy training datasets with 40% asymmetric noise. The meta-validation curve is the average accuracy of the networks on the clean held-out validation splits of these datasets. The x-axis corresponds to outer-loop iterations of re-training the MLP under the evolving loss function. Both (noisy) training and (clean) validation accuracies are evaluated at the end of each outer-loop iteration after the MLP has been completely trained under the current loss function. From the curve, we observe that the loss function continues to improve the noisy-label learning performance of the MLP throughout meta-training, until it eventually plateaus.

Table 3.6: Accuracy difference (%) between training-to-the-end and early-stopping. Higher numbers indicate models that require careful validation-set-driven early-stopping that may not be feasible in noisy label settings. Lower numbers indicate models that are more robust insofar as not requiring carefully chosen stopping times.

Loss type	CE	GCE	SCE	NCE+MAE	NFL+MAE	ARL (AR)	ARL (DR)
Asym04	18.01±10.95	15.82±10.55	10.85±8.76	10.83±6.29	11.87±8.26	1.84±1.25	6.15±4.76
Sym08	26.30±16.38	17.67±15.60	14.12±10.33	10.87±9.90	7.85±8.85	4.97±3.45	3.90±4.31

Sensitivity to early-stopping As discussed before, using noisy validation for hyper-parameter tuning and early stopping generates different models when compared with the models trained with long epochs, with models selected by early stopping tending to have better performance. The performance gap between a model picked by early stopping and that trained to convergence reflects the robustness of a loss function. A model that requires very careful model selection/early stopping can be considered non-robust in this sense, while a robust model performs similarly for different stopping iterations. To evaluate this, we compute the mean and variance of the early-stopping vs convergence type of gap over all the tasks introduced in Tables 3.1-3.2. From the results in Table 3.6, we can see that our ARL exhibits the smallest gap, confirming that it does not require careful tuning compared to alternatives.

3.5 Conclusion

In this work, we took an AutoML perspective on the problem of noise robust loss function design. Our results reveal a new loss function that combines low-penalty and soft minima features to produce a noise-robust loss function. ARL provides a simple reusable loss that can be plugged into diverse benchmarks and model architectures to learn robust features and classifiers in the presence of label noise, all without requiring a clean validation set or expensive meta-learning or distillation procedures.

3.6 Future Work

Besides the good properties of regularising model training on noisy labels discovered by the proposed ARL, there are still some potential directions that can be explored to strengthen the current algorithm from loss function parameterisation and meta-training

protocol scenarios. In the current work, the loss function is learned assuming that the noise ratio and noise type are known in advance. In addition, the visual comparison in Figure A.1 indicates that the curvatures of the learned loss functions vary with respect to different noise levels. This means that the current parameterisation is fail to capture noise ratio invariant information enabling the learned loss functions to generalise, and it also implies every time when there is a new type of noise that appears we have to execute the whole algorithm to produce new loss functions with respect to the novel settings. To this end, we could explore how to expand the current loss parameterisation to integrate noise ratio and noise type as input arguments for the loss function. More interestingly, learning noise ratio and noise type agnostic loss functions is also another important direction for studying the generalisation of meta-representation, which matches more the real word setting since usually the noise information is unknown.

In terms of meta-training protocols, we introduce dataset randomisation and architecture randomisation to make the learned loss function robust to unknown meta-test settings. However, designing efficient datasets and architecture sampling spaces is an essential problem for meta-generalisation remaining unsolved. To handle this problem, several questions should be answered first. Such as how to quantify the similarity between the datasets and between architectures. How to measure the similarity between the meta-train and meta-test tasks? How many meta-train tasks are sufficient to enable meta-generalisation? We leave those questions to future work and believe after answering those questions, ARL will have improved.

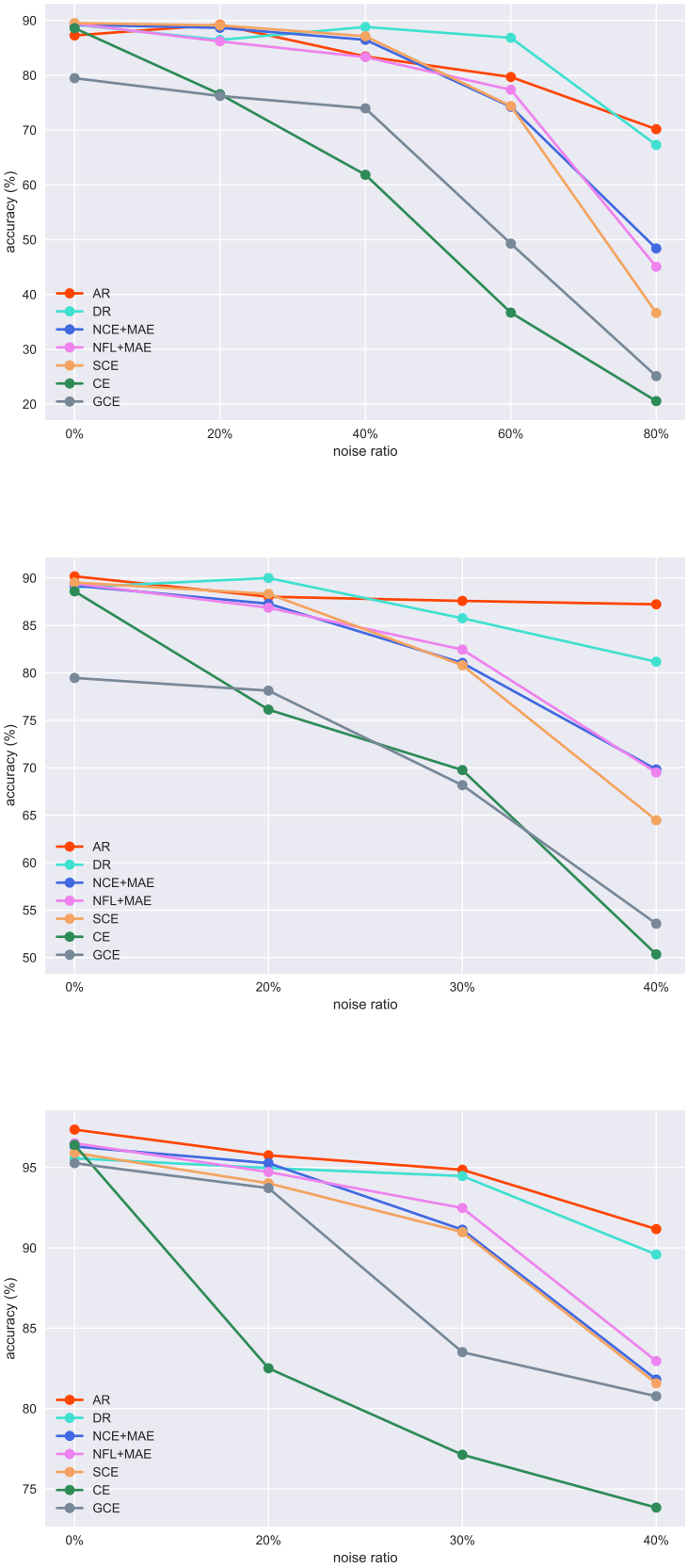

Figure 3.6: Generalisation of learned ARL loss to varying noise levels. Top: VGG11-FashionMNIST (Symmetric noise), Middle: VGG11-FashionMNIST (Asymmetric noise), Bottom: ResNet18-USPS (Asymmetric noise).

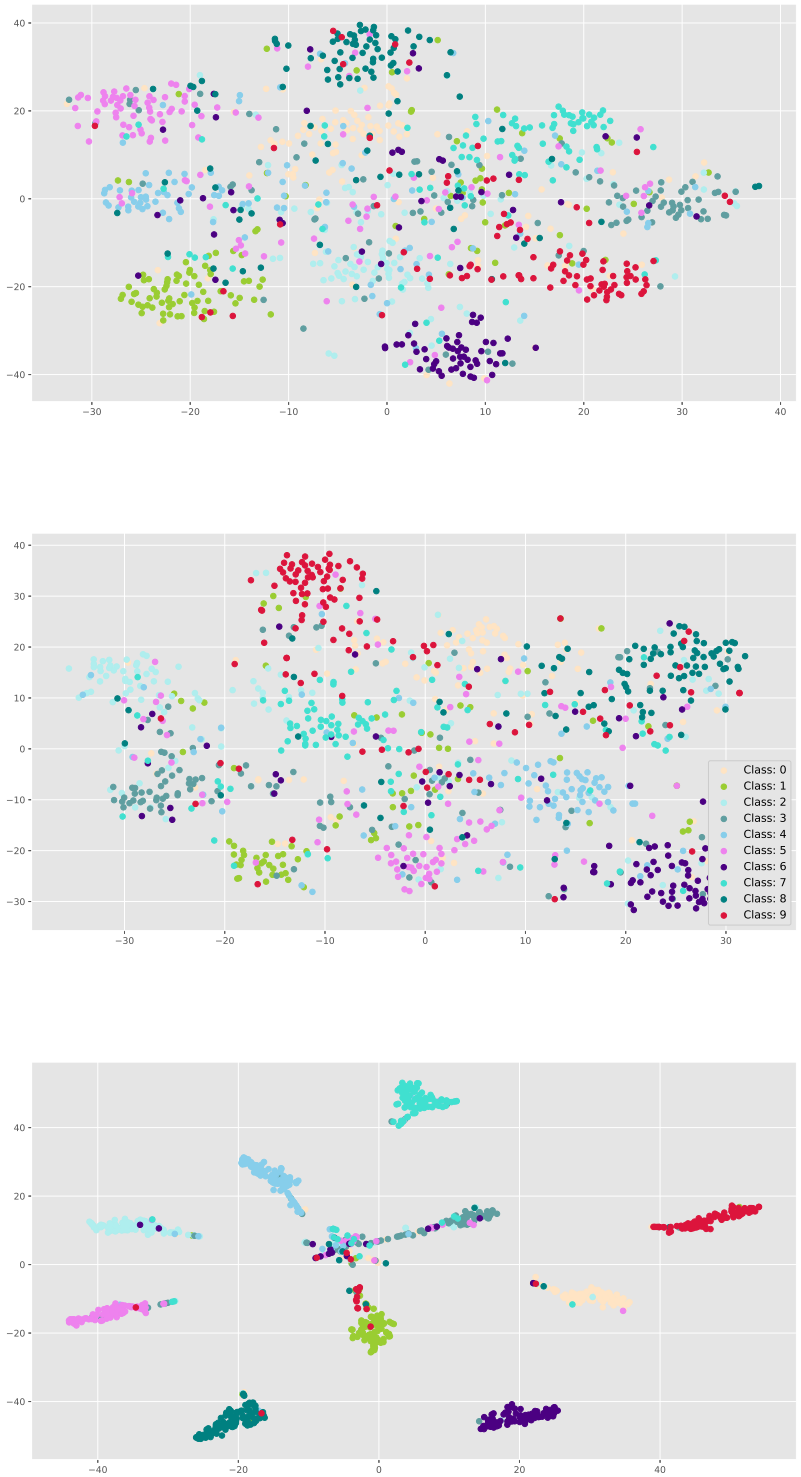

Figure 3.7: t-SNE visualisation of penultimate layer ResNet-18 features after learning on CIFAR-10 with 40% symmetric label noise. Top: CE training. Middle: Bootstrap training. Bottom: Our ARL training.

Chapter 4

Loss Function Learning for Domain Generalization by Implicit Gradient

This chapter corresponds to the paper:
Loss Function Learning for Domain Generalization by Implicit Gradient.
Boyan Gao, Henry Gouk, Yongxin Yang, and Timothy Hospedales, *International Conference on Machine Learning (ICML)*, 2022.

We continue the study of loss function learning and its transfer ability by proposing a more efficient learning algorithm named, Implicit Taylor Loss (ITL), in this chapter. As discussed in the last chapter, the ARL algorithm performs a genetic algorithm, CMA-ES, to search for a set of optimum loss function parameters in the outer loop. So that the models trained under the supervision of the learned losses can still gain knowledge from the polluted data and generalise to the unseen test samples. The performance of genetic optimisation methods highly relies on population diversity and size. In order to conduct such optimisation, a sufficient population is needed, which requires a huge computational cost. In addition, the randomness caused by the population sampling makes the optimisation process converge at a slow speed. An alternative way is to optimise the loss parameters with hypergradient, which provides a more stable and efficient updating signal, while its calculation, backpropagating through the entire base-model training horizon, is not trackable. In practice, many methods approximate hypergradient by only computing the first-order derivative and limiting base model updating iterations. Those (Shin et al., 2021; Nichol et al., 2018; Finn et al., 2017) operations make a trade-off between computational efficiency and performance. And

only observing and learning a piece of the base model training trajectories introduces short-horizon bias into the potential loss and degrades its performance when deployed to train a model from scratch. Bengio (2000) and Pedregosa (2016) apply Implicit Function Theorem to estimate hypergradient for hyperparameter optimisation and Lorraine et al. (2020) develops a more efficent method for high dimentional hypergradient computation with long training horizons. We expand this method in loss function learning. Besides, we study the loss function in another active research field, domain generalisation.

Generalising robustly to distribution shift is a major challenge that is pervasive across most real-world applications of machine learning. A recent study highlighted that many advanced algorithms proposed to tackle such domain generalisation (DG) fail to outperform a properly tuned empirical risk minimisation (ERM) baseline. We take a different approach, and explore the impact of the ERM loss function on out-of-domain generalisation. In particular, we introduce a novel meta-learning approach to loss function search based on implicit gradient. This enables us to discover a general purpose parametric loss function that provides a drop-in replacement for cross-entropy. Our loss can be used in standard training pipelines to efficiently train robust models using any neural architecture on new datasets. The results show that it clearly surpasses cross-entropy, enables simple ERM to outperform some more complicated prior DG methods, and provides excellent performance across a variety of DG benchmarks. Furthermore, unlike most existing DG approaches, our setup applies to the most practical setting of single-source domain generalisation, on which we show significant improvement.

4.1 Introduction

Deep learning is highly successful when the training and testing samples meet the i.i.d. assumption. However, this assumption is violated in many practical applications of machine learning from medical imaging to earth observation imaging (Koh et al., 2021). This has led a large number of studies to investigate approaches to training models with increased robustness to distribution shift at testing-time, a problem setting known as Domain Generalisation (DG). Despite the volume of research in this area (Zhou et al., 2021a), a recent careful benchmarking exercise, DomainBed (Gulrajani and Lopez-Paz, 2021) showed that simple empirical risk minimisation (ERM) on a combination of training domains is a very strong baseline when properly tuned. State-of-the-art alternatives based on sophisticated architectures, regularisers, and data augmentation

schemes failed to reliably beat ERM (Gulrajani and Lopez-Paz, 2021).

Rather than propose an alternative to ERM for DG, we investigate a previously unstudied hyperparameter of ERM, namely the choice of loss function—which has been ubiquitously taken to be standard cross-entropy (CE) in prior DG work. Loss function choice has been shown to impact calibration (Mukhoti et al., 2020), overfitting (Gonzalez and Miikkulainen, 2020a), and label-noise robustness (Wang et al., 2019) in standard supervised learning, so it is intuitive that it would impact robustness to domain-shift. However, it has not yet been studied in this context. Our preliminary experiments showed that equipping ERM with some recent robust loss functions in place of CE does lead to improvements in DG performance where sophisticated alternatives have failed (Gulrajani and Lopez-Paz, 2021). This raises the question: can one design a loss function specialised for DG?

To answer this question, we define a meta-learning algorithm to learn a parametric (white-box) loss function suitable for DG. Our desiderata are: (1) Performing ERM with this loss on a source domain should lead to good performance when tested on out-of-domain target data; and (2) It should provide a 'plug-and-play' drop-in replacement for cross-entropy that, once learned, can be used without further modification or computational expense with any new dataset or model architecture. While there has been growing interest in meta-learning for loss function design (Li et al., 2019a), they mostly fail to meet these criteria. They learn problem-specific—rather than re-usable—losses. If applied to DG, this would imply replacing simple ERM learning with sophisticated meta-learning pipelines to train a loss on a per-problem basis. In contrast, as illustrated schematically in Fig. 4.1, we learn a loss function once on a simple DG task (RotatedMNIST) and demonstrate that it subsequently provides a drop-in replacement for CE that improves an array of more challenging DG recognition tasks.

To train a general purpose robust loss function we need a search space that is flexible enough to include interesting new losses, but simple enough to generalise across tasks without overfitting to the problem used for loss learning. We choose a 12-dimensional space of fourth order Taylor polynomials (Gonzalez and Miikkulainen, 2021). Furthermore, we need a loss that is suitable for all stages of training. This precludes the majority of loss-learning approaches based on online meta-learning which update the loss and base model iteratively (Li et al., 2019a,d), and also suffer from short-horizon bias (Wu et al., 2018a). Evolutionary methods (Gonzalez and Miikkulainen, 2020a) and reinforcement-learning (Li et al., 2019a) could support loss learning in

principle, but are too slow to be feasible. Therefore we develop the first implicit-gradient based approach to loss learning. This allows us to tractably compute meta-gradients of the target recognition performance with respect to the loss used for training in the source domain.

We use a simple DG task (RotatedMNIST) to train our robust loss, termed Implicit Taylor Loss (ITL), to replace CE in ERM. Subsequent experiments show that ERM with ITL surpasses CE across a range of DG benchmarks, and provides very strong performance, despite being much simpler and faster than competitor DG methods. While the majority of existing DG methods require multiple source domains to conduct data augmentation or feature alignment strategies, ITL improves *single-source* domain generalisation, a crucial problem setting which has been minimally studied thus far.

To summarise our contributions: (i) We provide the first study on the significance of supervised loss function choice in DG (ii) We demonstrate the first efficient solution to loss-learning based on meta-gradients computed by the Implicit Function Theorem. (iii) Empirically, we show that our learned ITL loss enhances simple ERM and achieves competitive DG performance across a range of benchmarks, including the challenging single-source DG scenario.

4.2 Related Work

Domain Generalisation: Domain Generalisation aims to learn a model using data from one or more source domains, but with the further requirement that it is robust to testing on novel target domain data—without accessing target data during training. DG is now a well studied (Zhou et al., 2021a) area with diverse approaches including data augmentation (Shankar et al., 2018; Zhou et al., 2021b), robust training algorithms such as domain alignment objectives (Li et al., 2018b), and other regularisers (Li et al., 2019d; Balaji et al., 2018). Most DG studies have assumed the *multi-source* setting, which enables new data-augmentation strategies (Zhou et al., 2021b), and allows generalisation-promoting design features to be tuned by domain-wise cross-validation. In particular, a few studies (Li et al., 2019d; Balaji et al., 2018) have considered meta-learning based DG, where a regulariser applied in a training domain is tuned by meta-gradients from the resulting validation-domain performance. The resulting model is then deployed to the true target domain within the same family. These methods require regulariser meta-learning for each given multi-source DG problem family. In

contrast, we propose to learn a simple loss function once, which then provides a drop-in replacement for CE in any single-, or multi-source DG problem (Fig. 4.1). A recent criticism of the DG literature showed that no method consistently outperformed a well tuned ERM baseline on the carefully designed DomainBed benchmark (Gulrajani and Lopez-Paz, 2021). Rather than competing with ERM, we simply enhance the ERM loss function and this leads to a clear improvement on DomainBed.

Loss Function Learning: Loss function learning aims to discover new losses that improve model optimisation from various perspectives including conventional generalisation, (Gonzalez and Miikkulainen, 2020a; Liu et al., 2021), optimisation efficiency (Li et al., 2019a; Gonzalez and Miikkulainen, 2020a; Wang et al., 2020; Bechtle et al., 2020), and noise robustness (Li et al., 2019a; Gao et al., 2021a). Key dichotomies are in the search space of black box (neural) (Bechtle et al., 2020; Li et al., 2019d) vs white-box (human-readable) (Li et al., 2019a; Gonzalez and Miikkulainen, 2020a; Wang et al., 2020) losses; whether learned losses are problem specific (Li et al., 2019a; Wang et al., 2020) or reusable (Gonzalez and Miikkulainen, 2020a); the meta-optimisation algorithm Hospedales et al. (2021) used—evolution (Liu et al., 2021; Gonzalez and Miikkulainen, 2020a), RL (Li et al., 2019a; Wang et al., 2020; Bechtle et al., 2020), or gradient (Li et al., 2019d); and whether the loss is updated offline (Liu et al., 2021; Gonzalez and Miikkulainen, 2020a) (long inner loop, typically intractable), or online (Li et al., 2019a; Wang et al., 2020) (short inner loop, efficient but suffers from short-horizon bias (Wu et al., 2018a)). No studies have yet investigated loss learning for domain-shift robustness. In order to learn a reusable robust loss we use a white-box loss search space of Taylor polynomials proposed in Gonzalez and Miikkulainen (2021), and offline/long inner loop meta-learning. To make meta-optimisation tractable, we exploit the Implicit Function Theorem, to efficiently generate accurate hypergradients of the validation domain performance with respect to the training domain loss function parameters. Besides being the first demonstration of loss learning for DG, to our knowledge it is also the first demonstration of any implicit gradient-based loss learning.

4.3 Method

The need for Domain Generalisation arises when one is using machine learning to build a model where the available training data is not representative of the data that will be observed by the model once it has been deployed. In particular, it is assumed that

there is an underlying distribution over domains, P, from which we can sample several *source* domain distributions, $\{p_1^{(s)}, ..., p_n^{(s)} \sim P\}$, to make use of during training. We can construct a training set for each of these source domain distributions by sampling K data points, $D_i^{(s)} = \{(\mathbf{x}_i^{(s,j)}, y_i^{(s,j)}) \sim p_i^{(s)}\}_{j=1}^K$, and use the union of all these sets as the full training set, $D^{(s)} = \bigcup_{i=1}^n D_i^{(s)}$. Empirical Risk Minimisation (ERM) then simply finds the model parameters, θ, that minimise the loss measured on this training set,

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{K} \sum_{j=1}^{K} \mathcal{L}(f_\theta(\mathbf{x}_i^{(j)}), y_i^{(j)}), \tag{4.1}$$

where $\mathcal{L}(\cdot,\cdot)$ is a loss function (typically cross entropy) measuring how well the predicted labels match the ground truth labels. One can empirically check the resulting model's robustness to domain shift by sampling one or more *target* domain distributions, $\{p_1^{(t)}, ..., p_m^{(t)} \sim P\}$, from the same distribution over domains that was used to generate the training data. Data can then be sampled for each of these target domains, yielding a test dataset $D^{(t)} = \bigcup_{i=1}^m D_i^{(t)}$. Standard evaluation metrics such as accuracy can then be computed using this data.

4.3.1 Meta-Learning Losses for DG

Our goal is to replace the standard CE loss typically used in ERM with a learned loss function. We are motivated by recent work showing that learned losses can enable models to perform better for a variety of other problem settings, such as training with label noise (Wang et al., 2019) and improving calibration (Mukhoti et al., 2020). We formulate the task of learning the parameters, ω, of a loss function, $\mathcal{L}_\omega$, as a bilevel optimisation problem. The outer objective is to find the ω that maximises the performance of a model evaluated on the target domain data, and the inner problem is to train a model to minimise the value of $\mathcal{L}_\omega$ measured on the source domain data. The loss parameters are optimised using gradient-based methods that take advantage of the implicit function theorem to efficiently compute gradients for the outer optimisation problem. Crucially, once the optimal loss function ω^* has been found, new DG problems can be solved via ERM on the $\mathcal{L}_{\omega*}$ loss.

The bilevel optimisation that we use to formalise the meta-learning process is given by

$$\omega^* = \arg\min_{\omega} \frac{1}{m} \sum_{i=1}^{m} \frac{1}{K} \sum_{j=1}^{K} \mathcal{M}(f_{\theta^*(\omega)}(\mathbf{x}_i^{(t,j)}), y_i^{(t,j)}) \tag{4.2}$$

$$s.t. \; \theta^*(\omega) = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{K} \sum_{j=1}^{K} \mathcal{L}_\omega(f_\theta(\mathbf{x}_i^{(s,j)}), y_i^{(s,j)}) \tag{4.3}$$

where $\mathcal{M}$ is a loss function used to measure the performance of the model on the target domains, typically chosen to be cross entropy.

Optimising ω is challenging due to the need to backpropagate through the long inner loop optimisation of θ Hospedales et al. (2021). Existing approaches for learning loss functions typically resort to slow evolutionary or reinforcement learning updates (Li et al., 2019a; Gonzalez and Miikkulainen, 2020a, 2021; Wang et al., 2020) in the outer loop, or to an online approximation based on alternating steps on ω and θ (Li et al., 2019a; Wang et al., 2020). The latter approach leads to losses ω^* that cannot be transferred to new tasks, as it suffers from a short-horizon bias (Wu et al., 2018a). To solve this problem, we use the Implicit Function Theorem (IFT) to compute $\frac{\partial \mathcal{M}}{\partial \omega}$ without truncating the inner optimisation problem to approximate $\theta^*(\omega)$.

4.3.2 Implicit Gradient

The conceptually simplest way to optimise ω is to store all the intermediate iterates generated by the optimiser when training the network in the inner loop, and to then backpropagate through all of these weight updates (Maclaurin et al., 2015). This becomes prohibitively expensive in both memory and computation. Instead, after finding $\theta^*(\omega)$ we compute the gradient using the Implicit Function Theorem (IFT). The implicit gradient computation takes advantage of the fact that $\frac{\partial \mathcal{L}_\omega}{\partial \theta} = 0$, because we have found locally optimal model parameters for the inner problem. The gradient we want to compute is given by

$$\frac{\partial \mathcal{M}}{\partial \omega} = \frac{\partial \mathcal{M}}{\partial \theta} \frac{\partial \theta}{\partial \omega}\bigg|_{\omega, \theta^*(\omega)}, \tag{4.4}$$

and the IFT can be used to obtain

$$\frac{\partial \theta}{\partial \omega} = -\Big[\underbrace{\frac{\partial^2 \mathcal{L}_\omega}{\partial \theta \, \partial \theta^T}}_{|\theta| \times |\theta|}\Big]^{-1} \times \underbrace{\frac{\partial^2 \mathcal{L}_\omega}{\partial \theta \, \partial \omega^T}}_{|\theta| \times |\omega|}. \tag{4.5}$$

The inverse of the Hessian can be rephrased in terms of a Neumann series,

$$\Big[\frac{\partial^2 \mathcal{L}_\omega}{\partial \theta \, \partial \theta^T}\Big]^{-1} = \lim_{i \to \infty} \sum_{j=0}^{i} [I - \frac{\partial^2 \mathcal{L}_\omega}{\partial \theta \, \partial \theta^T}]^j, \tag{4.6}$$

and approximated by truncating the summation to a finite number of terms. In practice, one can make use of vector-Jacobian products to avoid explicitly constructing the Hessian in the summation. Further details can be found in Lorraine et al. (2020), but we provide pseudo-code for computing the implicit gradient in Algorithm 8.

Algorithm 7 IFT-based loss learning for DG.

Input: P, ω
Output: ω^*
Init ω
while not converged or reached max steps **do**
  sample $p_1, ..., p_n$ from P
  sample $D_1, ..., D_n$ from $p_1, ..., p_n$
  Init $H = 0 \in \mathbb{R}^{n \times |\omega|}$
  for all D_i **do**
    Init θ_i {Get random network weights}
    $D^s = \{D_1, ..., D_n\}/D_i, \quad D^t = D_i$ {Construct source/target splits}
    $\theta_i^* = \arg\min_\theta \mathcal{L}_\omega(\theta_i, D^s)$ {Train the network}
    $h_i = \text{Hypergradient}(\mathcal{L}_\omega, \mathcal{M}, (\omega, \theta_i^*), \alpha)$
    $H[i,:] = h_i$
  end for
  $h = \text{grad-surgery}(H)$
  $\omega = \omega - \eta h$ {Update the loss function}
end while

Algorithm 8 Computing the hypergradient of the meta-objective $\mathcal{M}$, with respect to the loss ω. The $\text{grad}(\cdot,\cdot,\cdot)$ function from PyTorch computes a Jacobian-vector product when called with a non-scalar first argument. Inspired by Lorraine et al. (2020), we use this to efficiently compute the Hessian required for approximating the Neumann series.

Input: $\mathcal{L}_\omega, \mathcal{M}, (\omega, \theta^*), \alpha$
Output: $-p \frac{\partial^2 \mathcal{L}_\omega}{\partial \theta \partial \omega}$
$v = p = \frac{\partial \mathcal{M}}{\partial \theta}|_{(\omega, \theta^*)}$
for all $j = 1, ..., J$ **do**
  $v -= \alpha \cdot \text{grad}(\frac{\partial \mathcal{L}_\omega}{\partial \theta}, \theta, v)$
  $p \mathrel{+}= v$
end for

4.3.3 Robust Gradient Estimation

Algorithm 7 summarizes the gradient estimation procedure. To obtain high-quality gradient estimates in each outer loop iteration, we employ a leave-one-domain-out

strategy. The DG task, P, used for meta-training the loss parameters has m domains associated with it. In each iteration of the outer loop, we train m networks with the prospective loss (i.e., we instantiate m different copies of the inner loop), where each network has a different target domain and the remainder of the domains are used to train the network. We can then compute a gradient for each of the m networks and aggregate them together in order to perform an update to the loss parameters. Rather than using the mean gradient, we found aggregation using gradient surgery (Yu et al., 2020), which reduces the gradient noise caused by different source/target domain splits in the inner loop, to work better in practice.

subsectionTaylor Polynomial Representation The choice of loss function parameterisation is a crucial factor in our framework. One must balance the ability to represent a sufficiently broad range of loss functions, with the susceptibility to overfitting the data used to learn the loss, and hence failing to generalise to novel tasks as desired[1]. The search space we consider is based on the truncated Taylor polynomials proposed by (Gonzalez and Miikkulainen, 2021) and successfully trained with evolution in (Gonzalez and Miikkulainen, 2021) and (Gao et al., 2021a). This family of loss functions treats the point around which the Taylor polynomial is centred, and also the value of the derivatives at this point, as learnable parameters. In this sense, it is a variational learning method—though it should be stressed it is not a variational *Bayesian* method. The family of β order multivariate Taylor polynomials has the form

$$\ell(\mathbf{z}) = \sum_{n=0}^{\beta} \frac{1}{n!} \nabla^n \ell(\mathbf{c})^T (\mathbf{z} - \mathbf{c})^n, \tag{4.7}$$

where $\mathbf{c}$ is a fixed point around which the function is being expanded. Because $\mathbf{c}$ is fixed, the values of the derivative at this point are also fixed. As such, we can replace c and $\nabla^n \ell(\mathbf{c})$ with meta-learnable parameters. This allows us to parameterize the learned loss function in terms of the gradients it should have at a meta-learnable point. We define our learnable loss as

$$\mathcal{L}_\omega(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{C} \sum_{i=1}^{C} \ell_\omega(\hat{\mathbf{y}}_i, \mathbf{y}_i) \tag{4.8}$$

$$\ell_\omega(\hat{\mathbf{y}}_i, \mathbf{y}_i) = \sum_{n=0}^{\beta} \frac{1}{n!} \nabla^n \ell([\omega_0, \omega_1])^T ([\hat{\mathbf{y}}_i, \mathbf{y}_i] - [\omega_0, \omega_1])^n, \tag{4.9}$$

where each $\nabla^n \ell([\omega_0, \omega_1])$ can actually be replaced by introducing more meta-parameters to ω. Please see Appendix B.1.1 for an expanded definition of this loss function.

[1] Eg: A linear combination of existing losses is likely insufficiently expressive, while a neural network is likely too expressive.

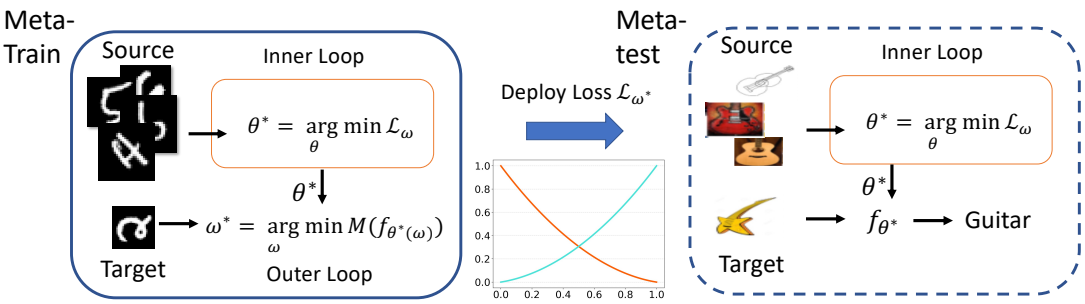

Figure 4.1: Algorithm schematic. Loss $\mathcal{L}_\omega$ is trained to optimize held-out domain performance on R-MNIST and then deployed on novel datasets.

4.3.4 Algorithm Summary

Meta-train: Given a set of training domains, the loss function search space in Section 4.3.3, and efficient update strategy in Section 4.3.2 and Algorithm 7, we are able to train a robust loss function $\mathcal{L}_\omega$. We conduct such loss function learning only once using a small dataset, and then evaluate the resulting loss on a variety of larger datasets that are unseen during meta-training.

Meta-test: Given the learned loss function $\mathcal{L}_{\omega^*}$, we fix it and use it together with the ERM algorithm for novel DG tasks. Each target problem is trained from scratch and has not been seen during loss learning. An overview of the algorithm, and the learning curve of the meta-train phase, are given in Figure 4.1.

4.4 Experiments

4.4.1 Dataset and Implementation Details.

Meta-train stage: We aim to learn a general purpose loss function that can be used in diverse DG problems. We choose RotatedMNIST (Ghifary et al., 2015) as a small dataset suitable for loss learning. It contains six different domains that are all derived from MNIST (LeCun and Cortes, 2010) but with different rotations: 0%, 15%, 30%, 45%, 60%, and 75%. The leave-one-domain-out strategy for robust implicit gradient estimation, therefore, results in six inner loop instantiations for each outer loop iteration. For efficiency, we use 2-layer MLPs as the base model, which contains 1024-256-10 units from the input layer to the output one with ReLU as the activation function. The learning rates in the inner loop and outer loop are both 0.01, a batch size of 32 is used

Table 4.1: **Resnet18** Cross-domain recognition accuracy (%) on PACS.

Method / Target set	Art	Cartoon	Photo	Sketch	Avg.
Epi-FCR (Li et al., 2019b)	82.1	77.0	93.9	73.0	81.5
JiGen (Carlucci et al., 2019)	79.4	75.3	96.0	71.6	80.5
MASF (Dou et al., 2019)	80.3	77.2	95.0	71.7	81.0
CrossGrad (Shankar et al., 2018)	79.8	76.8	96.0	70.2	80.7
Entropy (Zhao et al., 2020)	80.7	76.4	96.7	71.8	81.4
L2A-OT (Zhou et al., 2020)	83.3	78.2	96.2	73.6	82.8
RSC (reported in Huang et al. (2020))	83.43	80.31	95.99	80.85	85.15
Mixstyle (rs) (Zhou et al., 2021b)	82.3 $\pm$ 0.2	79.0 $\pm$ 0.3	96.3 $\pm$ 0.3	73.8 $\pm$ 0.9	82.8
Mixstyle (dl) (Zhou et al., 2021b)	84.1 $\pm$ 0.4	78.8 $\pm$ 0.4	96.1 $\pm$ 0.3	75.9 $\pm$ 0.9	83.7
RSC (reimplemented) + CE	79.3 $\pm$ 0.7	77.6 $\pm$ 0.5	93.6 $\pm$ 0.4	78.1 $\pm$ 1.4	81.9
RSC (reimplemented) + ITL	81.7 $\pm$ 0.8	76.6 $\pm$ 0.5	95.6 $\pm$ 0.2	77.1 $\pm$ 0.6	82.7
ERM + BCE	71.2 $\pm$ 0.8	70.8 $\pm$ 0.3	93.1 $\pm$ 0.8	57.7 $\pm$ 1.0	73.2
ERM + CE	76.9 $\pm$ 0.6	76.5 $\pm$ 0.7	93.3 $\pm$ 0.1	68.8 $\pm$ 0.6	78.9
ERM + SCE (Wang et al., 2019)	80.4 $\pm$ 0.7	73.6 $\pm$ 0.7	92.3 $\pm$ 0.2	74.8 $\pm$ 0.6	80.3
ERM + iSCE	81.2 $\pm$ 0.6	74.4 $\pm$ 0.5	93.6 $\pm$ 0.1	79.2 $\pm$ 0.7	82.1
ERM + ITL (ITL-Net)	83.9 $\pm$ 0.4	78.9 $\pm$ 0.6	94.8 $\pm$ 0.2	80.1 $\pm$ 0.6	**84.4**

for the inner loop, and the Neumann series used for approximating the inverse Hessian is truncated at 15 iterations. The result is a set of 12 parameters that define the learned fourth-order polynomial loss function. A learned loss is plotted in Fig. 4.5, and its parameters are given in Appendix B.1.1. The meta-train compute cost is reported in Appendix B.1.5.

Meta-test (Deployment) Stage We now approach using simple ERM, but replacing cross-entropy with our learned loss. Our complete model is denoted by ITL-Net. We evaluate the learned loss function on the four common DG benchmarks: VLCS (Fang et al., 2013), PACS (Li et al., 2018a), OfficeHome (Venkateswara et al., 2017), and Terra Incognita (Beery et al., 2018). Two sets of experiments are conducted: (i) We evaluate the conventional PACS benchmark, as it is the most widely used in the DG literature, and enables comparison against the most recent state-of-the-art competitors. (ii) We evaluate all four benchmarks using the recent DomainBed platform, which is designed to enforce fair and consistent hyperparameter tuning across different methods.

4.4.2 Results

PACS: Setup A pre-trained ResNet18 backbone is used throughout, together with the source and target domain split described in (Li et al., 2018a). We train ResNet-18 with ITL on the training split and perform model selection using the validation set. We use the same data split to tune the hyperparameters for the baseline ERM with Cross-Entropy (ERM +CE) and Binary Cross-Entropy (BCE). We compare ITL with several state-of-the-art alternatives on this benchmark, including RSC (Huang et al., 2020), data augmentation-based L2A-OT (Zhou et al., 2021b), Mixstyle (Zhou et al., 2020) including random shuffle (rs) and domain label (dl), regulariser-based Entropy (Zhao et al., 2020), adversarial gradient-based CrossGrad (Shankar et al., 2018), meta learning-based MASF (Dou et al., 2019) and Epi-FCR (Li et al., 2019b) and self-supervision-based JiGen (Carlucci et al., 2019).

PACS: Results We first conduct experiments using the classic PACS protocol to facilitate comparison against many recent competitors that were not evaluated on DomainBed. Table 4.1 compares our ITL-Net performance vs state-of-the-art methods. From the results we can see that: (i) Simply swapping out the loss in ERM from CE to ITL-Net, leads to a significant 5.5% improvement. (ii) Overall our ITL-Net provides strong performance on this benchmark, surpassing recent and sophisticated competitors such as Mixstyle.

DomainBed: Setup We next evaluate ITL using the DomainBed platform, which enforces careful and fair evaluation by ensuring that all competitors use the same hyper-parameter tuning strategy (random search, driven by source domain validation performance), and the same number of hyperparameter search iterations. We follow the standard DomainBed protocol and use a ResNet-50, with experiments conducted on VLCS, PACS, OfficeHome, and Terra Incognita.

DomainBed: Setup We next evaluate ITL using the DomainBed platform, which enforces careful and fair evaluation by ensuring that all competitors use the same hyper-parameter tuning strategy (random search, driven by source domain validation performance), and the same number of hyperparameter search iterations. We follow the standard DomainBed protocol and use a ResNet-50, with experiments conducted on VLCS, PACS, OfficeHome, and Terra Incognita.

DomainBed: Results The results in Table 4.2 compare ITL-Net with ERM and some of the competitive published alternatives: SagNet (Nam et al., 2021), CORAL (Sun and Saenko, 2016), CDANN (Li et al., 2018c) and RSC (Huang et al., 2020) in the original DomainBed paper (Gulrajani and Lopez-Paz, 2021). A detailed comparison is given in Appendix B.1.3. The conclusion of the DomainBed study was that existing methods did not reliably beat ERM under this hyperparameter tuning protocol, when using cross-entropy loss. In contrast, we can see that ITL-Net (ERM with ITL loss) provides a clear improvement on ERM and matches or improves on the strongest competitor in each case, especially on TerraIncognita and OfficeHome. To formally compare ITL-Net with other methods, we perform significance testing using a Friedman test with a post-hoc Nemenyi test on the average ranks. Following the recommendation of Demšar (2006), the plot in Figure 4.2 visualises the statistically significant differences between the performance of all methods. From this, we can see that our approach statistically significantly outperforms all of our baselines except for SagNet.

Single Source DG: Setup Most existing DG methods rely on the availability of multiple source domains in some form: For example to synthesise new domains for data augmentation (Zhou et al., 2021b), or perform feature alignment among training domains (Sun and Saenko, 2016). A unique feature of ITL-Net is that, since it is only a small modification to ERM, it can be used to learn on a single source domain. Although this setting is not well explored in the literature, it is obviously highly practical as multiple source domains are often not available in practice. To explore this challenging setting, we modify the DomainBed benchmark to train on a single source at a time and average over each source→target combination, rather than training on the conjunction of all sources.

Single Source DG: Results From the results in Table 4.4, we can see that performance drops across the board compared to multi-source training (Table 4.2), as expected. However, the state-of-the-art alternatives CORAL and SagNet are no longer as competitive compared to ERM as they were in the multi-source case (Table 4.2)—this is expected as they are designed to exploit cues from multiple source domains. In contrast, our ITL-Net maintains a clear lead over the conventional ERM with cross-entropy baseline in this setting. This is a significant achievement as existing work has not produced algorithms that reliably improve robustness under the single-source setting.

Table 4.2: **DomainBed** Cross-domain recognition accuracy (%) with **ResNet50** on **ColoredMNIST VLCS**, **PACS**, **TerraIncognita**, **OfficeHome** and **DomainNet**.

	Models					
Dataset	ERM	SagNet	CORAL	CDANN	RSC	ITL-Net
ColoredMNIST	51.5	51.7	51.5	51.7	51.7	**52.0**
VLCS	77.5	77.8	78.8	77.5	77.1	**78.9**
PACS	85.5	86.3	86.2	82.6	85.2	**86.4**
TerraIncoginita	46.1	48.6	47.6	45.8	46.6	**51.0**
OfficeHome	66.5	68.1	68.7	65.8	65.5	**69.3**
DomainNet	40.9	40.3	41.5	38.3	38.9	**41.6**
Avg. Rank	4.17	2.67	3.00	5.17	5.00	1.00

Table 4.3: Cross-domain recognition accuracy (%) on **DomainBed-PACS-Resnet50**. Comparison with alternative manually-designed robust losses.

Loss	ERM+CE	ERM+FOCAL	ERM+SCE
Avg Perf	83.9 ± 0.5	84.6 ± 0.8	84.2 ± 0.5
Loss	ERM+GCE	EMR+LS	ERM+ITL
Avg Perf	83.0 ± 0.2	84.9 ± 0.6	$\mathbf{86.4} \pm 0.5$

4.4.3 Further Analysis on Training

Meta-Training Convergence Figure 4.3 shows the convergence of ITL during meta-training on R-MNIST. The x-axis shows outer loop iterations/loss function updates. The lines show (i) the inner loop accuracy (mean over all source domains) after model training with the current loss function, and (ii) and the outer loop accuracy (held out domain accuracy). Clearly the convergence process is quite smooth.

Implicit Gradient vs Evolutionary Optimisation An evolutionary optimiser, such as Covariance Matrix Adaptation Evolution Strategy (CMA-ES (Hansen and Ostermeier, 1996)), is an alternative to gradient-based optimisation. We compare the proposed implicit gradient-based algorithm with CMA-ES in the meta-train stage. CMA-ES needs to train K models (typically $5 \leq K \leq 50$) to estimate gradient by finite difference, and implicit gradient only trains one. Thus implicit gradient is K-times faster than

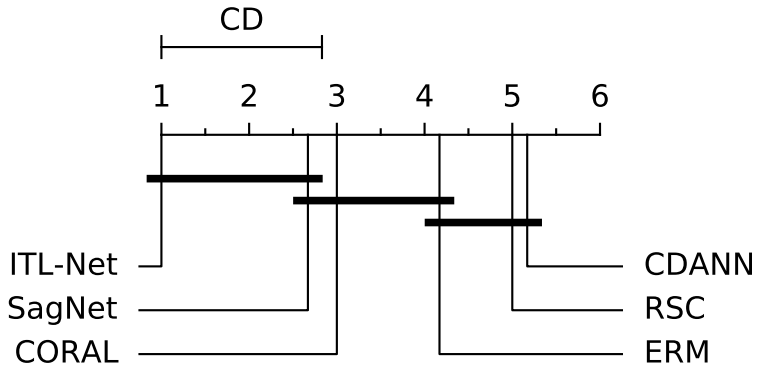

Figure 4.2: A critical difference diagram showing the results of Nemenyi post-hoc test on the average ranks. Methods connected by a thick black bar indicate the lack of a statistically significant difference in performance. ITL-Net significantly outperforms all competitors besides SagNet.

Table 4.4: Cross-domain recognition accuracy (%) on DomainBed with a single source domain. The heading of the table denotes the single source domain, and results average across all target domains.

Source Dataset	VLCS	PACS	OfficeHome	TerraIncognita
ERM	**64.08**	51.85	53.57	32.13
CORAL	64.07	51.84	53.51	32.13
SagNet	61.78	53.00	51.30	33.93
Mixup	59.01	54.92	52.70	30.80
ITL-Net	62.17	**56.54**	**55.04**	**35.09**

CMA-ES in FLOPs. As can be seen in Figure 4.4, CMA-ES makes noisy stochastic estimates of gradients w.r.t. the loss function parameters. The implicit gradient gets the exact gradient of validation loss, which leads to a better solution and more stable convergence. In our empirical comparison, we can see that our proposed method leads to smoother convergence to a higher accuracy solution (while being K times faster).

Repeatability Analysis Our message thus far is that a single loss produced by our pipeline can be re-used as a plug-and-play modification to improve vanilla ERM+CE on a wide variety of held-out downstream DG tasks. That said, one might reasonably wonder about the reliability of the loss function learning procedure itself. To investi-

Table 4.5: Cross-domain recognition accuracy (%) on **OfficeHome**: Impact of meta-train seed ($\pm$ standard deviation), and choice of pre-training dataset.

Target set	Artistic	Clipart	Product	Real World	Avg.
ITL-NET (RotatedMNIST)	64.22 ± 0.84	56.25 ± 0.38	77.52 ± 0.32	78.12 ± 0.32	69.03 ± 0.18
ITL-NET (RotatedKMNIST)	64.28 ± 0.30	55.84 ± 0.29	76.89 ± 1.04	77.96 ± 0.30	68.74 ± 0.35

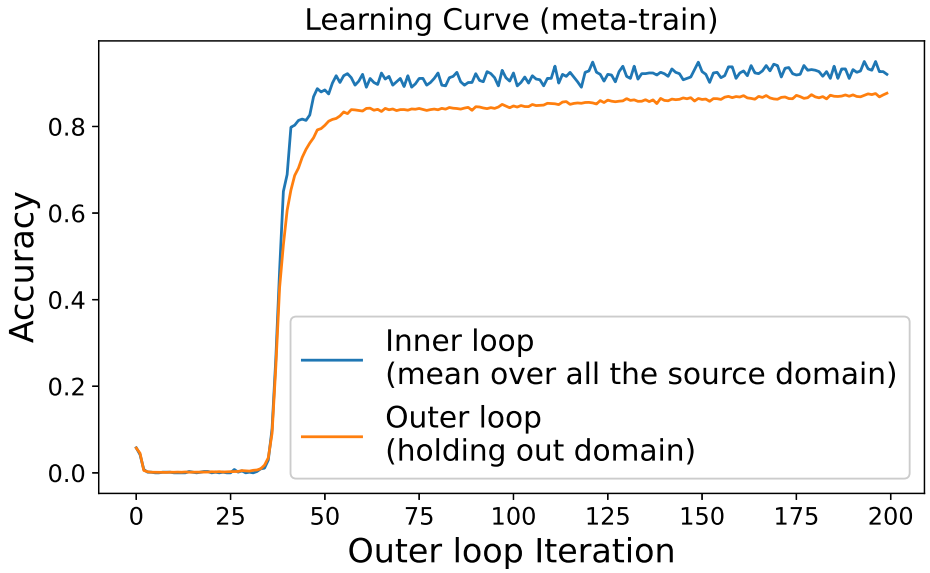

Figure 4.3: The learning curve for ITL meta-training stage.

gate this, we repeat our entire pipeline *including* the meta-train stage five times. We then evaluate the consistency of the resulting five loss functions on the downstream ColoredMNIST task. Furthermore, to evaluate the dependence of our result on the choice of the meta-training dataset, we repeat the above experiment using RotatedKMNIST (Clanuwat et al., 2018) to replace the RotatedMNIST used previously. From the results in Table 4.5, we can see that performance is quite consistent over trials (small standard deviation). It also differs little with the choice of pre-training dataset - with both options performing well compared to competitors in Table 4.2.

Task Specific Loss Learning We aimed to learn a reusable loss function. This is because the cost to meta-learn a loss is substantial and thus less practical to perform on a per-dataset/task basis. That said, our framework supports task-specific loss learning, and this could potentially outperform a general purpose loss. To demonstrate this we deploy our proposed algorithm for task-specific loss learning on a small benchmark – ColoredMNIST. The results in Table 4.6 show a slight improvement on the generic ITL.

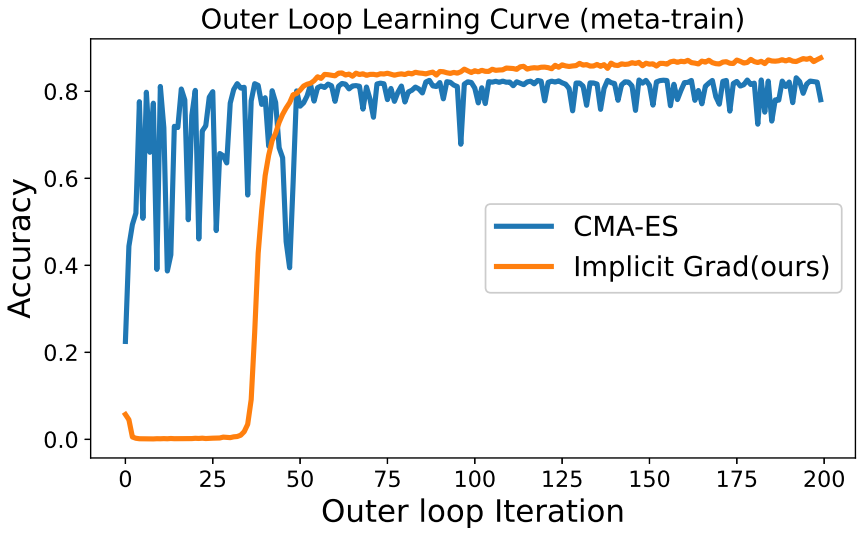

Figure 4.4: Meta-Learning curves for Implicit Gradient vs Evolution.

Table 4.6: **MNIST NET** Cross-domain recognition accuracy (%) on coloredMNIST with Task-specific loss vs generic ITL-Net.

Target set	+90%	+80%	-90%	Avg.
ITL-NET	71.3 ± 0.6	73.4 ± 0.1	11.3 ± 0.7	52.0
Task-specific ITL-NET	72.2 ± 0.7	73.9 ± 0.5	10.4 ± 0.7	52.2

4.4.4 Further Analysis on Learned Loss

Visualising ITL To interpret our learned loss, we visually compare it in Figure 4.5 with several other loss functions: CE, SCE (Wang et al., 2019), Label Smoothing (LS) (Pereyra et al., 2017), and Focal (Mukhoti et al., 2020). Compared to the standard cross-entropy loss, we can see that ITL has softer penalties for severe misclassification, and stronger penalties for moderate misclassification. Among existing losses, ITL is visually most similar to Focal.

Quantitative Comparison to Other Robust Losses We next investigate whether the good cross-domain performance of ITL can be easily replicated by applying existing robust loss functions, or whether our meta-learning pipeline has learned something new in terms of robust model training. SCE (Wang et al., 2019) and GCE (Zhang

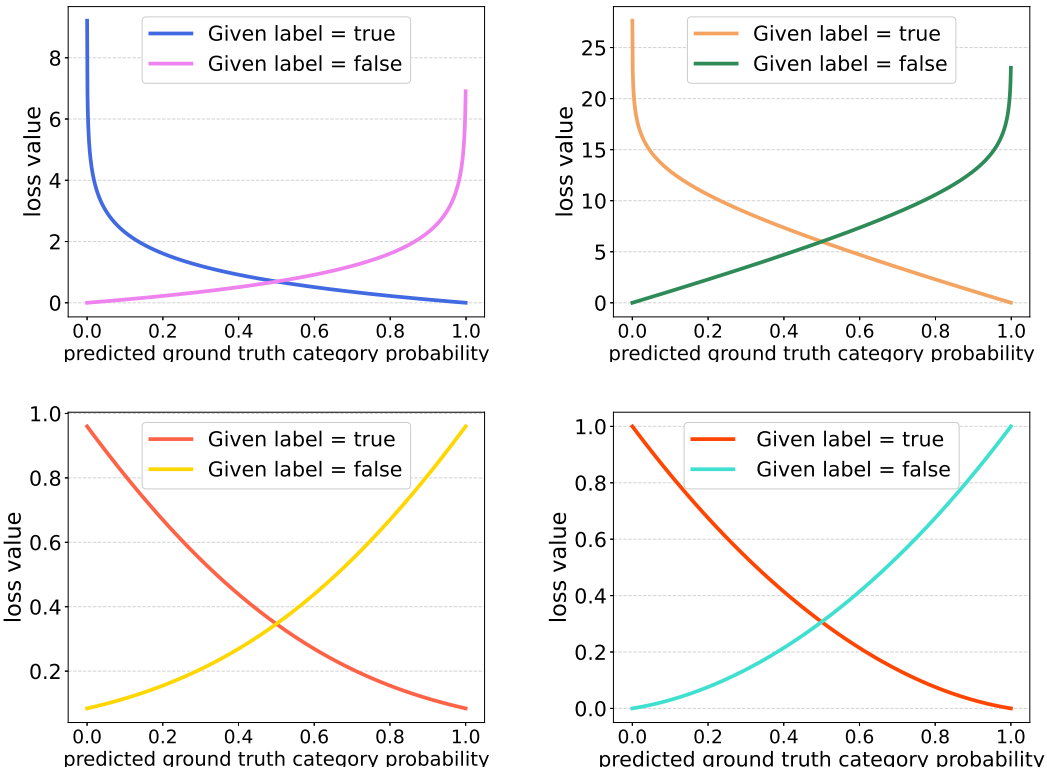

Figure 4.5: Comparison of loss functions (from left to right): CE, SCE, FOCAL and ITL. The range of ITL is normalised between 0 and 1.

and Sabuncu, 2018) were designed with label-noise robustness in mind, while Focal (Mukhoti et al., 2020; Lin et al., 2017) was designed for class imbalance and calibration. Label-smoothing (LS) (Pereyra et al., 2017) is for improving generalisation and reducing overconfidence. From the results in Table 4.3, we can see that while some losses improve on CE, ITL leads to the clearest improvement. Note that all experiments in Table 4.3 were run by us, while competitor performance in Table 4.2 is taken from (Gulrajani and Lopez-Paz, 2021).

Choice of Loss Function Family We focused on Taylor polynomials as the main family of loss functions to explore given the motivation in Gonzalez and Miikkulainen (2021). One might ask whether learning other loss families are possible, and how they compare. To explore this we consider Symmetric Cross Entropy Wang et al. (2019), $SCE(p,q) = \alpha CE(p,q) + \beta CE(q,p)$, which is a linear combination of cross-entropy functions, and was shown to improve learning robustness. We consider learning the linear combination hyper-parameters (α, β) of SCE with our IFT-based meta-learning pipeline, and denote the result iSCE. Comparing ERM+CE with

ERM+SCE and ERM+iSCE in Tab 4.1, we see that SCE improves on CE; learning the linear combination of losses in iSCE improves on a fixed linear combination; but both perform worse than our ERM + ITL. Thus we can conclude that (i) our framework is generic and can be used with different loss paramaterisations, but (ii) ITL provides a better loss family than SCE. In addition, we analyse the effect introduced by different Taylor polynomial orders. The selected order is determined by performance in the meta-train stage. We compare the inner loop and outer loop accuracy of the models trained by different with the order ranging from 2 to 5 for selecting the order in Fig 4.6.

Combination with Other Base DG Methods Our training framework and evaluation focused on pairing ITL with vanilla ERM. However, as a plug-in module, ITL can be applied with any other downstream DG method. To illustrate this we re-implement the state-of-the-art RSC method Huang et al. (2020) and plug in our ITL. While our re-implementation slightly underperforms the previously reported numbers, the direct comparison of RSC+CE vs RSC+ITL in Tab 4.1 shows that ITL can also benefit other more sophisticated methods than ERM.

In-distribution vs Out-of-distribution Our meta-objective for loss learning is OOD performance after train-test domain shift. One hypothesis Li et al. (2022) for why ITL works is that it leads to better-regularised models than CE for the purpose of OOD evaluation. To analyse this we report accuracy and loss for training and testing, both in-distribution and cross-distribution in Table 4.7. We can see that: (i) ITL always leads to slightly worse training loss than CE, which reaches zero; (ii) ITL leads to better testing loss and accuracy than CE in cross-domain conditions; (iii) ITL leads to worse testing loss and testing accuracy than CE for in-domain condition. Together this suggests that ITL regularises more strongly than CE (worse training fit), preventing over-learning the details of the source domain. This in turn is beneficial for cross-domain evaluation, but detrimental to performance within-domain. Worse in-domain performance is a weakness of ITL.

4.4.5 Discussion and Limitations

In terms of the full meta-training procedure, our framework requires multiple source domains for training, though the resulting loss can be deployed on single-source problems. While our empirical results show that ITL generalises well on the benchmarks tested, more evaluation is necessary to fully validate the generality of the learned loss.

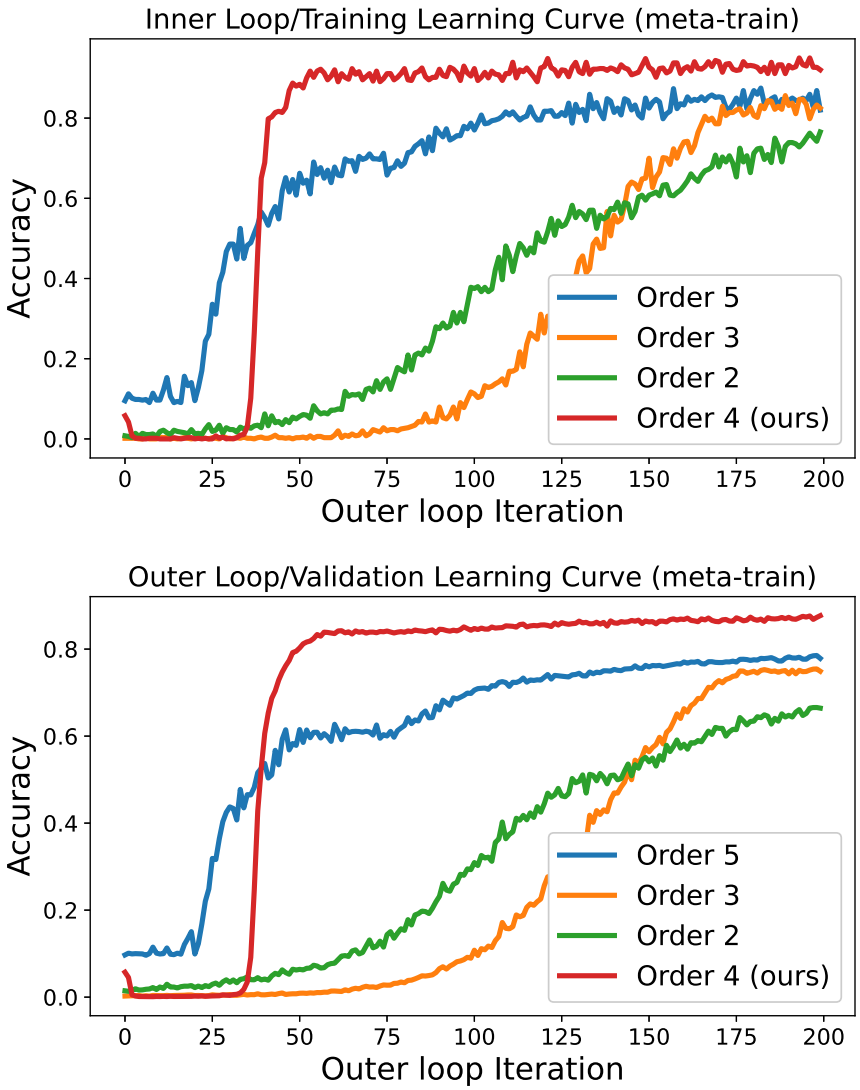

Figure 4.6: Dependence of loss order in meta-train Stage.

For example, if the source-target distribution shifts $D(p_s(x), p_t(x))$ seen in meta-test deployment is much larger or smaller than in meta-training, the loss may not be well-tuned. Similarly, all our empirical results involve benchmarks exhibiting a marginal shift in $p(x)$. Whether performance improvement is retained under conditional shift $p(y|x)$ remains to be seen.

While our results showed ITL consistently improving on ERM+CE, state-of-the-art competitors may perform similarly or better than ITL-Net – when those competitors are substantially better than ERM to start with. We emphasise that we are not trying to propose a state-of-the-art DG method, (i) provide a simple and efficient baseline for use when more complex alternatives are unsuitable, and (ii) raise awareness of the loss

Table 4.7: In-distribution (Cross-distribution) comparison between CE and ITL on PACS with the loss measured by CrossEntropy.

Model	Train accuracy (%)	Train Loss
CrossEntropy	100.00 (100.00)	0.00 (0.00)
ITL	100.00 (100.00)	1.15×10^{-4} (7.3×10^{-3})
	Test accuracy (%)	Test Loss
CrossEntropy	99.34 (81.5)	6.61×10^{-4} (3.64×10^{-2})
ITL	98.96 (84.4)	7.67×10^{-4} (1.95×10^{-2})

function as an important design parameter in DG. Our initial experiment with RSC showed that ITL is potentially complementary to other state-of-the-art DG methods, but a deeper exploration of this is left to future work. To this end, better performance is likely achievable by meta-training ITL together with a more sophisticated base model of interest, rather than meta-training with ERM and deploying with RSC as we demonstrated earlier. ERM-like methods such as SWAD Cha et al. (2021) would complement ITL naturally.

4.5 Conclusion

We provided the first study of the effect of loss functions in ERM-based Domain Generalisation. We empirically observe that choice of loss function during ERM training impacts domain generalisation performance. To discover the best loss for DG, we perform meta-learning to find a re-usable white-box loss function. This is tractably solved using IFT to obtain gradients of the target domain performance with respect to the source domain loss parameters. This also provides the first demonstration of IFT-based loss learning in the literature. The results show that a simple modification to a simple ERM pipeline improves both multi-source and single-source DG, and even surpasses several sophisticated purpose-designed state-of-the-art models.

4.6 Future Work

Similar to the previous chapter, where a loss function algorithm for noisy label task is proposed, we can argue that the potential research interest of ITL lies in the loss function parameterisation and the meta-training protocol design. The parameterisation of the loss function seems less important than the latter since it is not obvious what type of extra information from the unseen DG tasks can be considered as input to the loss function, but the cardinality and network architecture are still important information which can be taken as input for loss functions in general. Interestingly, ITL learns domain generalisation task agnostic loss functions with a group of simple DG tasks, which makes the training process of ITL much cheaper and the task-specific ones only have slight improvements. A natural question based on this observation is how a loss function is learned with great generalisation ability, which enables it to be applied to much more complicated tasks where both architecture and dataset are very different to those on meta-train. Unfortunately, we are missing a concrete theoretical explanation for the good empirical results. The answer to this question will help us to know how to efficiently design a task space to reduce the meta-train cost, and we leave it to future work.

We have discussed two algorithms for learning loss functions and both of them rely on the multi-task setting in the inner loop to achieve meta-generalisation. For now, the multi-task design depends only on empirical experiences. How to decide the number of tasks and how to build the task axis to span the meta-train tasks remains unknown. To produce a more reliable meta-training protocol, some theory-based design principles should be proposed and to tackle this problem. The first step should be to find an efficient way of measuring the similarity between meta-train tasks from the loss function perspective.

Chapter 5

MetaMD: Learning an Optimiser with Convergence Guarantees

This chapter corresponds to the papers:
Meta Mirror Descent: Optimiser Learning for Fast Convergence.
Boyan Gao, Henry Gouk, Hae Beom Lee, and Timothy Hospedales, *Conference on Learning Representation (ICLR) Workshop on GroundedML*, 2022 *(best paper finalist)*
MetaMD: Learning an Optimiser with Convergence Guarantees.
Boyan Gao, Henry Gouk, Jan Stuehmer, and Timothy Hospedales, 2022.

In this chapter, we discuss optimiser learning. Optimisers are an essential component for training machine learning models, and their design influences learning speed and generalisation. Several studies have attempted to learn more effective gradient-descent optimisers via solving a bi-level optimisation problem where generalisation error is minimised with respect to optimiser parameters, such as learning rates for SGD (Micaelli and Storkey, 2021), symbolic gradient-descent rules (Bello et al., 2017) neural network gradient-descent rules (Andrychowicz et al., 2016; Li and Malik, 2017), and gradient-free optimisers (Sandler et al., 2021; Chen et al., 2017). However, most existing optimiser learning methods are intuitively motivated, without clear theoretical support. We take a different perspective starting from mirror descent rather than gradient descent, and meta-learning the corresponding Bregman divergence. Within this paradigm, we formalise a novel meta-learning objective of minimising the regret bound of learning. The resulting framework, termed Meta Mirror Descent (MetaMD), learns

to accelerate optimisation speed. Unlike many meta-learned optimisers, it also supports convergence and generalisation guarantees and uniquely does so without requiring validation data. We evaluate our framework on a variety of tasks and architectures in terms of convergence rate and generalisation error and demonstrate strong performance.

5.1 Introduction

Gradient-based optimization algorithms, such as stochastic gradient descent (SGD), are fundamental building blocks of many machine learning algorithms – notably those focused on training linear models and deep neural networks. These methods are typically developed to solve a broad class of problems, and therefore the method developers make as few assumptions about the target problem as possible. This leads to a variety of general purpose techniques for optimization, but such generality often comes with slower convergence. By taking advantage of more information about the target problem, one is typically able to design more efficient—but less general—optimization algorithms. For example, by taking advantage of second order information, Newton's method is able to converge to optima in many fewer iterations than gradient descent, which uses only first order information. However, the application of Newton's method is limited compared to gradient descent, because it can only be used to solve problems where the second order information exists and can be computed efficiently. Another challenge in a non-convex deep learning context, is that that many of the empirically fastest optimizers such as Adam (Kingma and Ba, 2015) lack convergence guarantees.

While one line of research hand-designs optimisers to exploit known properties of particular problems, a complementary line of research focuses on situations where optimisation problems come in families. This allows using meta-learning techniques to fit an optimiser to the given problem family with the goal of maximising convergence speed or generalisation performance. For example, in the many-shot regime, Andrychowicz et al. (2016) and Wichrowska et al. (2017) learn black-box neural optimisers to accelerate training of neural networks, while Bello et al. (2017) learn symbolic gradient-based optimisers to improve generalisation. MAML (Finn et al., 2017) and Meta-SGD (Li et al., 2017) learned initialisation and learning rate for SGD training of neural networks with good generalisation performance in the few-shot regime. Later generalisations focused on learning problem family-specific curvature information (Park and Oliva, 2019; Flennerhag et al., 2020). Nevertheless, most existing learned optimisers such as

(Andrychowicz et al., 2016; Wichrowska et al., 2017; Flennerhag et al., 2020; Bello et al., 2017) can not provide convergence or generalisation guarantees.

In this work, we revisit the optimizer learning problem from the perspective of *mirror descent*. Mirror descent introduces a Bregman divergence that regularises the distance between current and next iterate, introducing a strongly convex sub-problem that can be optimised exactly. In mirror descent, the choice of Bregman divergence determines optimisation dynamics. In a meta-learning context, the Bregman divergence thus provides a novel representation of an optimisation strategy that can be fit to a given family of optimisation problems, leading to our learned optimiser termed Meta Mirror Descent (MetaMD). Existing learned optimisers do not have a formal notion of convergence rate, and in practice typically optimise a meta-objective reflecting training or validation loss after a fixed number of iterations. In contrast, MetaMD is directly trained to optimise the convergence rate bound for mirror descent. Importantly, this means we can adapt theoretical guarantees from mirror descent to provide convergence guarantees for MetaMD, an important property not provided by most learned optimsers, and many hand-designed optimisers widely used in deep learning.

An important issue in meta-learning a mirror descent algorithm is specifying the family of Bregman divergences to learn. Meta-learning with general Bregman divergences leads to an intractable tri-level optimisation problem. Thus, we seek a family of divergences for which the innermost optimisation has a closed form solution. The chosen paramaterisation should be complex enough to exhibit interesting optimisation dynamics, simple enough to provide a closed form solution, while always providing a valid Bregman divergence. We provide an example parameterisation that meets all these desideratum in the form of a Kronecker factorised preconditioner (Zehfuss, 1858; Martens and Grosse, 2015). Preconditioners have been successfully exploited in meta-learning methods such as Meta-Curvature (Park and Oliva, 2019), and Warp-Grad (Flennerhag et al., 2020), and are richer than the parameter-wise learning rates considered by MetaSGD (Li et al., 2017) etc, which is a special case. Importantly, we also show how to train the divergence efficiently with implicit gradient computation (Lorraine et al., 2020). Uniquely, this means that our framework can effectively train learning-rates (as a special case of preconditioning) with implicit gradient, an idea that was previously suggested to be impossible (Lorraine et al., 2020).

Empirically we demonstrate that we can train MetaMD given a model architecture and a suite of training tasks. We then deploy it to novel testing tasks and show that MetaMD

provides fast convergence and good generalisation compared to several existing hand-designed and learned optimisers.

5.2 Related work

Meta-learning Meta-learning aims to extract some notion of '*how to learn*' given a task or distribution of tasks (Hospedales et al., 2020), such that new learning trials are better or faster. These two stages are often called meta-training, and meta-testing respectively. Key dichotomies include: meta-learning from a single task vs a task distribution; the type of meta-knowledge to be extracted; and long- vs short-horizon meta-learning. For few-shot problems with short optimization horizons, the seminal model-agnostic meta-learning (MAML) (Finn et al., 2017) learns an initial condition from which only a few optimisation steps are required solve a new task. Meta-SGD (Li et al., 2017) and Meta-Curvature (Park and Oliva, 2019) extend MAML by learning a parameter-wise learning rate, and a preconditioning curvature matrix respectively. Another group of methods focus on larger scale problems in terms of dataset size and optimization horizon. For example, neural architecture search (NAS) (Real et al., 2019; Zoph and Le, 2017) discovers effective neural architectures. MetaReg (Balaji et al., 2018) meta-learns regularization parameters to improve domain generalisation. ARL (Gao et al., 2021b) meta-learns a loss function to improve robustness of learning form noisy labels.

Several studies focus specifically on optimiser meta-learning for many-shot problems, which we address here. In this case, the extracted meta-knowledge spans learning rates for SGD (Micaelli and Storkey, 2021), symbolic gradient-descent rules (Bello et al., 2017) neural network gradient-descent rules (Andrychowicz et al., 2016; Li and Malik, 2017), and gradient-free optimisers (Sandler et al., 2021; Chen et al., 2017). Differently to the gradient-descent based methods, we start from the perspective of mirror descent, where mirror descent's Bregman Divergence provides an target for meta-learning. This perspective has several benefits, notably the ability to derive a learned optimizer with convergence and generalisation guarantees. Our suggested practical instantiation is to meta-learn a KFAC structured pre-conditioner, which is effective and efficient. However, the framework is general and can be used to learn particular special cases such as element-wise learning rates (Li et al., 2017; Micaelli and Storkey, 2021). Beyond supporting this richer representation, we provide convergence guarantees, do

not rely on a validation set, and demonstrate cross-dataset generalisation theoretically and empirically, enabling us to amortize meta-learning cost. In contrast, (Micaelli and Storkey, 2021)'s single task meta-learner needs to repeat meta-learning on each specific dataset to optimize per-dataset validation performance.

5.3 Mirror Descent

We formalise the problem of learning an optimiser using the Mirror Descent (MD) framework, which can be thought of as a generalisation of gradient descent. MD optimisers produce a series of progressively better estimates for the optimal parameters of the objective function. This is accomplished by solving a convex optimisation problem at each step, t,

$$\theta_{t+1} = \arg\min_{\theta} \langle \nabla_\theta \mathcal{L}_{tr}(\theta_t), \theta \rangle + \frac{1}{2\eta} B_\phi(\theta || \theta_t) \tag{5.1}$$

where $\mathcal{L}_{tr}$ represents the training loss function, η is the step size and B_ϕ denotes a Bregman divergence. Bregman divergences can be thought of as a way of measuring distance in parameter space, and each choice of Bregman divergence leads to a different optimisation algorithm. One can define Bregman divergences as

$$B_\phi(\theta || \theta') = \phi(\theta) - \phi(\theta') - \langle \nabla\phi(\theta'), \theta - \theta' \rangle, \tag{5.2}$$

where ϕ is a λ-strongly convex function. There are several choices of ϕ that result in existing algorithms specialised for various types of optimisation problems in machine learning. For example, if one chooses ϕ to be $\frac{1}{2}\|\cdot\|_2^2$, then mirror descent becomes gradient descent, while choosing ϕ to be the negative entropy results in the Kullback-Leibler divergence leads to the exponentiated gradient algorithm (Kivinen and Warmuth, 1997). A significant benefit of deriving new algorithms that fit into the mirror descent framework is that one can obtain a bound on the rate of convergence towards a minima θ_* for any valid choice of ϕ. This bound also applies to learned divergences B_ϕ.

Proposition 1. *Let B_ϕ the Bregman divergence w.r.t $\phi : X \to \mathbb{R}$ and assume ϕ to be λ-strongly convex with respect to $||\cdot||$ in Θ. Let $\Theta \subset X$. Set $\theta_1, \theta_* \in \Theta$ such that ϕ is differentiable in θ_1. Then the following holds*

$$\sum_{t=1}^{T} (l(\theta_t) - l(\theta_*)) \leq \frac{B_\phi(\theta_* || \theta_1)}{\eta} + \frac{\eta}{2\lambda} \sum_{t=1}^{T} ||g_t||_*^2,$$

where $\|\cdot\|_*$ is the dual norm of $\|\cdot\|$. g_t represents the t step gradient of the objective function ℓ whose minimiser is denoted as θ_*.

5.4 Meta-Learning a Mirror-Descent Bregman Divergence

5.4.1 Optimiser learning framework

We propose a meta-learning algorithm to learn mirror descent optimisers. We consider the multi-task meta-learning setting (Hospedales et al., 2020; Finn et al., 2017), assuming that a task distribution $p(\mathcal{T})$ is available from which we can draw tasks for meta-training, and that we will evaluate the learned optimizer by meta-testing on novel tasks from the same distribution. For gradient-based meta-learning, the meta-training procedure is conventionally framed as a bilevel optimization problem where the inner problem solves learning tasks given the optimiser, and the outer problem updates the optimiser (Hospedales et al., 2020). The outer problem is to minimise some meta objective denoted $\mathcal{E}(\phi)$ with respect to the optimiser parameters. Since we are learning a mirror descent optimizer defined by a Bregman divergence B_ϕ, this leads to a tri-level optimisation problem with a new layer corresponding to the problem given in Eq. 5.1 required to complete a single mirror descent step,

$$\min_{\phi} \mathcal{E}(\theta_*(\phi)) \tag{5.3}$$

$$\text{s.t.}\, \theta_*(\phi) = \arg\min_{\theta} \mathcal{L}_{tr}(\theta) = (\pi_\phi \circ \pi_\phi \circ \ldots \circ \pi_\phi)(\theta_1) \tag{5.4}$$

$$\text{s.t.}\, \pi_\phi(\theta_t) = \arg\min_{\theta} \langle \nabla_\theta \mathcal{L}_{tr}(\theta_t), \theta \rangle + \frac{1}{2\eta} B_\phi(\theta || \theta_t). \tag{5.5}$$

In the outer loop (Eq. 5.3), the algorithm aims to learn a divergence by optimising the meta-objective $\mathcal{E}$, which evaluates the optimiser performance. To achieve this requires getting the best response from the mid-level problem (Eq. 5.4) where the base model is trained from the initialisation θ_0 to θ_* by a sequence of π_ϕ which is the innermost problem in (Eq. 5.5)—which we denote the *mirror loop*, due to the convention of using iterative solvers for such multi-level optimization problems. The mirror loop performs mirror descent updates using a Bregman divergence based on ϕ. Compared with the standard bilevel problems in meta-learning, introducing this third layer adds significant cost to both meta-train and meta-test stages. However, with a suitable choice of divergence, we can obtain a closed-form solution for the mirror loop, which thus incurs similar cost to a standard bilevel optimisation problem. Our meta-learning framework for mirror descent optimisation is summarised in Alg 9.

Algorithm 9 Meta Mirror Descent learning algorithm.

Input: $p(\mathcal{T})$, ϕ_M {Task distribution, and initial divergence paramaterised by M}
Output: ϕ_M^*
while not converged or reached max steps **do**
  sample $T_1, ..., T_n$ from $p(\mathcal{T})$ {Get meta-train tasks}
  for all T_i **do**
    Init θ_i {Set random weights for base model}
    $\theta_i^* = \arg\min_\theta \mathcal{L}_\omega(\theta_i, T_i)$ {Train the base model}
    $h = h + \text{Hypergradient}(\mathcal{L}_\omega, \mathcal{E}, (\phi_M, \theta_i^*))$ {Obtain $d\mathcal{E}/dM$}
  end for
  $M = M - \frac{\rho}{n} h$ {Update the ϕ_M function}
end while

5.4.2 Divergence Parameterisation

The paramaterisation of the divergence is important for a practical instantiation of our framework. Ideally it should be expressive enough to represent interesting optimisation dynamics, while being simple enough to provide an efficient or closed form solution to the innermost convex mirror descent optimisation. We describe a reasonable compromise as follows.

We define our family of Bregman divergences via the squared norm, $\phi(\theta) = \frac{1}{2}\theta^T M \theta$, where M is a learnable symmetric positive definite matrix. In this case, the mirror descent loop has a closed form solution,

$$\theta_{t+1} = \theta_t - \eta M^{-1} \nabla_\theta \mathcal{L}_{tr}(\theta_t).$$

The derivation of this closed-form mapping is given in Appendix C.1. This paramaterisation can be interpreted as a preconditioner, which has high representation potential(Park and Oliva, 2019; Flennerhag et al., 2020), but it can be inefficient to compute M^{-1} when the dimensionality of θ is high—as is typically the case for neural networks. To provide a better trade-off between capacity and efficiency, we reduce the capacity of ϕ by using block diagonal matrix $M = diag(M_1, M_2, ..., M_Z)$ with each block operating on the gradient of one layer in a neural network. We additionally represent each M_ζ by a Kronecker factorisation (Zehfuss, 1858; Hensel, 1889; Martens and Grosse, 2015),

$$M_\zeta = A_\zeta \otimes B_\zeta,$$

where $A_\zeta \in R^{\kappa\times\kappa}$ and $B_\zeta \in R^{\iota\times\iota}$ are two symmetric positive definite matrices parameterised by the column vectors a_ζ and b_ζ,

$$A_\zeta = a_\zeta a_\zeta^T + I_\kappa$$
$$B_\zeta = b_\zeta b_\zeta^T + I_\iota.$$

To avoid explicitly constructing M_i, we make use of the fact that

$$\begin{aligned} M_\zeta^{-1}\nabla_{\theta_\zeta}\mathcal{L}_{tr}(\theta_t) &= (A_\zeta \otimes B_\zeta)^{-1}\nabla_{\theta_\zeta}\mathcal{L}_{tr}(\theta) \\ &= B_\zeta^{-1}\nabla_{\theta_\zeta}\mathcal{L}_{tr}(\theta)(A_\zeta^{-1})^T \end{aligned}$$

where θ_ζ denotes the weights in layer ζ. A_ζ^{-1} and B_ζ^{-1} can be easily computed by composing the eigenvector matrices and inverting the diagonal eigenvalue matrix generated by Singular Value Decomposition from A_ζ and B_ζ respectively.

5.4.3 Meta-Objective

The next step is to define the meta-objective with respect to the Bregman divergence B_ϕ. An advantage of the mirror-descent framework is that we have a formal notion of convergence from an initialization θ_1 to a solution θ_* (from Theorem 1). By defining the meta objective $\mathcal{E}(\phi)$ as a bound on the convergence rate, meta-learning ϕ leads to a faster optimizer.

In particular, we further bound Theorem 1, by assuming that the loss function is L-Lipschitz continuous, which bounds the gradient, leading to

$$\sum_{t=1}^{T}(l(\theta_t) - l(\theta_*)) \leq \frac{B_\phi(\theta_*||\theta_1)}{\eta} + \frac{\eta}{2\lambda}\sum_{t=1}^{T}||g_t||_*^2 \leq \frac{B_\phi(\theta_*||\theta_1)}{\eta} + \frac{\eta}{2\lambda}TL^2. \tag{5.6}$$

Minimising the right hand side of Eq. 5.6 with respect to ϕ_M improves the convergence rate of the learned optimiser. As our goal is to optimise the expected speed of convergence on future tasks, we design a meta-objective that considers the average convergence rate over n different meta-train tasks. Thus, we define the meta-objective as

$$\mathcal{E}(\phi) = \frac{1}{n}\sum_{i=1}^{n} B_\phi(\theta_T^{(i)}||\theta_1^{(i)}) + \frac{k}{\lambda}, \tag{5.7}$$

where $\theta_1^{(i)}$ and $\theta_T^{(i)}$ are the initial and final weights for task i, and we leave k as a hyperparameter that can be tuned heuristically or by using the relationship in Eq. 5.6.

The strong convexity parameter is given by $\lambda = \min_\zeta \sigma_{min}(M_\zeta)$, where σ_{min} denotes the smallest eigenvalue.

We solve the outer loop optimisation problem (Eq. 5.3) by gradient descent using $\partial \mathcal{E}(\phi_M)/\partial M$. This gradient computation relies on unrolling training trajectories in the inner loop where it is expensive to compute the gradient with standard reverse-mode differentiation. In this work we apply implicit differentiation (Lorraine et al., 2020) to solve this problem, which will be discussed later in Section 5.4.4.

5.4.3.0.1 Generalisation of the Learned Optimiser Having meta-learned our MetaMD optimiser (Bregman divergence) on a set of training tasks, we can ask how well it is expected to perform on novel tasks? The training convergence rate, or outer objective value in Eq. 5.7 will be an optimistically biased estimate of the convergence rate one can expect on future tasks. Given the relatively simple family of Bregman divergences we employ, it is possible to construct a high-confidence bound on how biased this estimate will be, and therefore provide convergence guarantees that can be trusted:

Proposition 2. *If we restrict* $\|M - I\|_F \leq C$ *and* $\|\theta_1 - \theta_*\|_2 \leq r$*, then the following holds with probability at least* $1 - \delta$*,*

$$\mathbb{E}\left[\sum_{t=1}^{T}(l(\theta_t) - l(\theta_*))\right] \leq \frac{1}{n}\sum_{i=1}^{n}\frac{B_\phi(\theta_*^{(i)}||\theta_1^{(i)})}{\eta} + \frac{\eta T L^2}{2\lambda} + \frac{Cr^2}{2\sqrt{n}} + 3Cr^2\sqrt{\frac{ln(2/\delta)}{8n}}. \quad (5.8)$$

This tells us that the expected convergence rate on novel tasks depends on the learning divergence on training tasks, plus complexity terms such as the F-norm of the meta-learned optimiser weights M. Note that restricting the diameter r of the parameter space is usually required to obtain generalisation guarantees (Bartlett et al., 2017; Long and Sedghi, 2020; Gouk et al., 2021), so this is not an unusual or counterproductive requirement. The details for computing $\|M - I\|_F$ are given in Appendix 5.5.

We will make use of a well-known bound on suprema of empirical processes due to (Bartlett and Mendelson, 2002).

Proposition 3. *For all functions* $f \in \mathcal{F}$ *with* $\|f\|_\infty \leq a$*, and i.i.d.* Z_i*, the following holds with probability at least* $1 - \delta$*,*

$$\mathbb{E}[f(Z)] \leq \frac{1}{n}\sum_{i=1}^{n} f(Z_i) + 2\hat{R}_n(\mathcal{F}) + 3a\sqrt{\frac{ln(2/\delta)}{2n}}. \quad (5.9)$$

In this theorem, $\hat{R}_n(\mathcal{F})$ is the empirical Rademacher complexity of the class $\mathcal{F}$, defined as

$$\hat{R}_n(\mathcal{F}) = \mathbb{E}_\sigma\left[\sup_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i f(Z_i)\right], \tag{5.10}$$

where σ_i is a Rademacher random variable, so $P(\sigma_i = -1) = P(\sigma_i = 1) = 0.5$.

Proof. It suffices to bound, with high confidence, the difference between the first term of the meta-objective, and the expected Bregman divergence between initializations and solutions on new tasks sampled from the same task distribution. We will obtain such a bound using Rademacher complexity, and the main result will follow from standard applications of Rademacher complexity-based generalisation bounds (Bartlett and Mendelson, 2002), along with the observation that $B_\phi(\theta_*||\theta_1) \leq \frac{Cr^2}{2}$. That is, we will treat the initial and final points of an optimisation trajectory for each task as a random variable, and we will bound to what extent the mean Bregman divergence between initial and trained parameters on some meta-train tasks can deviate from the expected Bregman divergence on unseen tasks. In particular, we analyse the following class:

$$\mathcal{F} = \{(\theta_*, \theta_1) \mapsto B_\phi(\theta_*||\theta_1) : \phi(\theta) = \frac{1}{2}\theta^T M\theta,\ \|M - I\|_F \leq C\}. \tag{5.11}$$

Denoting $\theta_*^{(i)} - \theta_1^{(i)}$ by $\tilde{\theta}_i$, we can bound the Rademacher complexity of this class from above by

$$\begin{aligned}
\hat{R}_n(\mathcal{F}) &= \mathbb{E}_\sigma\left[\sup_{B_\phi\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i B_\phi(\theta_*^{(i)}||\theta_1^{(i)})\right] && (5.12)\\
&= \frac{1}{2n}\mathbb{E}_\sigma\left[\sup_{M} \sum_{i=1}^{n} \sigma_i \tilde{\theta}_i^T M\tilde{\theta}_i\right] && (5.13)\\
&= \frac{1}{2n}\mathbb{E}_\sigma\left[\sup_{M} \sum_{i=1}^{n} \sigma_i \langle M, \tilde{\theta}_i\tilde{\theta}_i^T\rangle_F\right] && (5.14)\\
&= \frac{1}{2n}\mathbb{E}_\sigma\left[\sup_{M} \langle M, \sum_{i=1}^{n} \sigma_i \tilde{\theta}_i\tilde{\theta}_i^T\rangle_F\right] && (5.15)\\
&= \frac{1}{2n}\mathbb{E}_\sigma\left[\sup_{M} \langle (M-I), \sum_{i=1}^{n} \sigma_i \tilde{\theta}_i\tilde{\theta}_i^T\rangle_F\right] + \frac{1}{2n}\mathbb{E}_\sigma\left[\langle I, \sum_{i=1}^{n} \sigma_i \tilde{\theta}_i\tilde{\theta}_i^T\rangle_F\right], && (5.16)
\end{aligned}$$

where $\langle\cdot,\cdot\rangle_F$ is the Frobenius inner product. Note that the second term in the final equality is equal to zero. As such we can continue by further bounding the first term

using the Cauchy-Schwarz inequality,

$$\hat{R}_n(\mathcal{F}) \leq \frac{C}{2n}\mathbb{E}_\sigma\left[\left\|\sum_{i=1}^{n}\sigma_i\tilde{\theta}_i\tilde{\theta}_i^T\right\|_F\right] \tag{5.17}$$

The remainder of the proof follows a well-known sequence of steps using when bounding the expected norm of a Rademacher sum (see, e.g., (Shalev-Shwartz and Ben-David, 2014)), which we include here for completeness. Jensen's inequality tells us that

$$\mathbb{E}_\sigma\left[\left\|\sum_{i=1}^{n}\sigma_i\tilde{\theta}_i\tilde{\theta}_i^T\right\|_F\right] \leq \sqrt{\mathbb{E}_\sigma\left[\left\|\sum_{i=1}^{n}\sigma_i\tilde{\theta}_i\tilde{\theta}_i^T\right\|_F^2\right]} \tag{5.18}$$

$$= \sqrt{\mathbb{E}_\sigma\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_i\sigma_j\langle\tilde{\theta}_i\tilde{\theta}_i^T,\tilde{\theta}_j\tilde{\theta}_j^T\rangle_F\right]}. \tag{5.19}$$

Noting that $\mathbb{E}[\sigma_i\sigma_j]$ is one if $i=j$ and zero otherwise, we obtain

$$\sqrt{\mathbb{E}_\sigma\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_i\sigma_j\langle\tilde{\theta}_i\tilde{\theta}_i^T,\tilde{\theta}_j\tilde{\theta}_j^T\rangle_F\right]} \leq \sqrt{\sum_{i=1}^{n}\|\tilde{\theta}_i\tilde{\theta}_i^T\|_F^2} \tag{5.20}$$

$$\leq \sqrt{\sum_{i=1}^{n}(\|\tilde{\theta}_i\|_2\|\tilde{\theta}_i\|_2)^2} \tag{5.21}$$

$$\leq \sqrt{nr^4}. \tag{5.22}$$

Substituting this back into our earlier derivation yields

$$\hat{R}_n(\mathcal{F}) \leq \frac{C\sqrt{nr^4}}{2n} \tag{5.23}$$

$$= \frac{Cr^2}{2\sqrt{n}}, \tag{5.24}$$

which concludes the proof. □

5.4.4 Meta-Gradient computation with Implicit Gradient

We optimise the outer objective using gradient descent. Typical meta-learning pipelines will either unroll the sequence of updates performed by an iterative optimiser to solve the inner problem (Finn et al., 2017), or make use of the Implicit Function Theorem (IFT) to compute derivatives w.r.t. parameters that appear in the inner objective (Lorraine et al., 2020). The number of inner loop iterations required to arrive at a solution in our use-case (i.e., medium/many shot learning) makes the memory and computation requirements of unrolling this loop infeasible, so we use our prospective MD optimiser to find a

solution and apply the IFT method. However, the Bregman divergence parameters do not appear in the inner problem objective, leading to a roadblock when applying the IFT strategy to compute hypergradients after the inner problem has been solved using our MD optimiser. To this end, when computing the hypergradients, we use a rephrased inner objective expressed in terms of the Bregman divergence parameters. This is possible by using the property that mirror descent finds the fixed point closest to the initial guess of the network parameters, as measured by the corresponding Bregman divergence (Azizan et al., 2021). The alternative objective function is

$$\arg\min_{\theta} B_{\phi}(\theta, \theta_1) \tag{5.25}$$

$$\text{s.t.}\, \theta \in \arg\min_{\theta'} \mathcal{L}_{tr}(\theta'). \tag{5.26}$$

Framing the constrained optimisation problem as its Lagrangian and replacing the constraints with $(\mathcal{L}_{tr}(\theta_*) - \mathcal{L}_{tr}(\theta))^2 = 0$, we obtain

$$\arg\min_{\theta} \sup_{\mu} B_{\phi}(\theta, \theta_1) + \mu(\mathcal{L}_{tr}(\theta_*) - \mathcal{L}_{tr}(\theta))^2, \tag{5.27}$$

where θ_* is the solution obtained via solving the inner problem with our prospective MD optimiser, and we note that $\theta = \theta_*$ provides us with an immediate solution to this transformed problem. This means the second term always evaluates to zero when evaluated at the solution, and we therefore only need to apply the IFT to the first term, yielding

$$\frac{\partial \mathcal{E}}{\partial \phi} = \frac{\partial \mathcal{E}}{\partial \theta} \left(\frac{\partial^2 B_{\phi}}{\partial \theta \partial \theta} \right)^{-1} \frac{\partial^2 B_{\phi}}{\partial \theta \partial \phi} \bigg|_{\phi, \theta_*(\phi)}. \tag{5.28}$$

5.5 Meta Regulariser Computation

We provide the meta regualriser computation details in this section. The proposed meta regulariser is expressed as

$$\begin{aligned} ||M - I||_F^2 &= ||M||_F^2 + ||-I||_F^2 + 2\langle M, -I \rangle_F && (5.29)\\ &= ||M||_F^2 + ||I||_F^2 - 2\text{tr}(M) && (5.30)\\ &= ||M||_F^2 + ||I||_F^2 - 2\text{tr}(A)\text{tr}(B) && (5.31)\\ &= \sum_{i,j} \sigma_i(A)\sigma_j(B) + ||I||_F^2 - 2\text{tr}(A)\text{tr}(B), && (5.32) \end{aligned}$$

where $\sigma_i(\cdot)$ denotes the i-th singular value of a matrix. To get from Eq. 5.30 to Eq. 5.31 we use that $\text{tr}(A \otimes B) = \text{tr}(A)\text{tr}(B)$, and from Eq. 5.31 to Eq. 5.32 we use that $\|M\|_F^2 = \sum_i \sigma_i^2(A \otimes B) = \sum_{i,j} \sigma_i(A)\sigma_j(B)$.

5.6 Experiments

We evaluate our learned optimiser and compare its performance on a variety of tasks against commonly used standard optimisers including SGD, SGD-M (with momentum), Adam (Kingma and Ba, 2015) RMSProp (Tieleman and Hinton, 2012) and KFAC (Martens and Grosse, 2015), and meta-learning based methods, Meta-SGD (Li et al., 2017), PowerSign (Bello et al., 2017) and AddSign (Bello et al., 2017). We first explore synthetic tasks, followed by shallow neural networks on digit datasets, and then training ResNet on CIFAR-10, before finally evaluating deep neural networks on high resolution images.

Algorithm deployment pipeline: For each set of experiments, we train MetaMD on a set of meta-train datasets, and evaluate it on a disjoint set of meta-test datasets. In the meta-test stage, models are trained by MetaMD (or competitors) using each dataset's standard training set, and evaluated on the corresponding test splits. We emphasize that for meta-testing, each optimiser consumes the same amount of data, and a comparable amount of computing per iteration, except KFAC is much more expensive due to its inverse operation in every iteration. The learned optimisers such as MetaMD, MetaSGD (Li et al., 2017) and Signs (Bello et al., 2017) use additional data and compute for the prior meta-training stage, but this is a one-off cost that can be amortized across different meta-test problems of interest.

5.6.1 Synthetic Problem: Meta-Quadratic Optimisation

Setup We start with a synthetic experiment to illustrate the impact of the proposed meta objective, and simply using exact FMD for hypergradient for learning. Meta Mirror Descent is deployed on a family of 2D quadratic optimisation problems from which we can sample a disjoint set of meta-training and meta-testing optimisation problems. We sample tasks of the form:

$$\min_{\theta} \theta^T Q \theta - b^T \theta$$

where Q and b are random variables. b follows a Gaussian distribution with mean vector $[1,1]^T$ and identity covariance. To generate Q, we sample a two-dimension lower triangular matrix C to construct the symmetric positive defined matrix $Q = C \cdot C^T$. We also illustrate problems with different loss flatness by specifying the mean of $Q_{0,0}$ and $Q_{1,1}$.

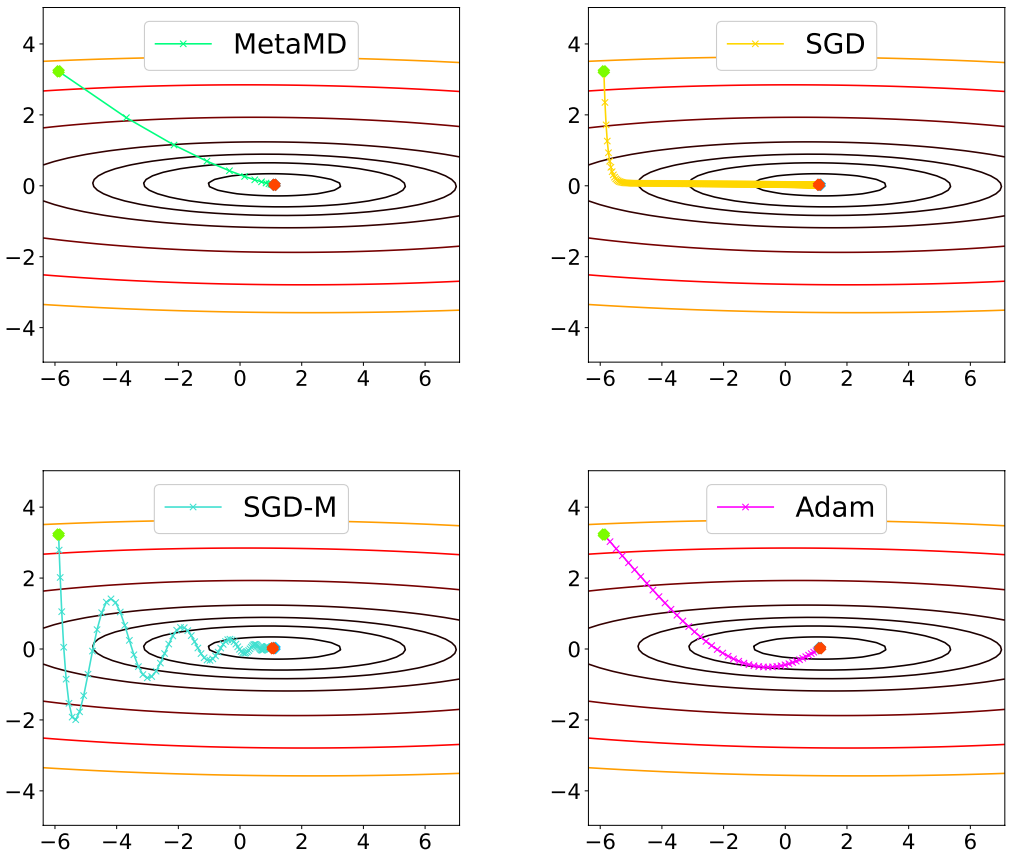

Figure 5.1: Trajectory comparison of different optimisers on a quadratic optimisation problems. Top: MetaMD, SGD. Bottom: SGD-M and Adam. The point green point denotes the starting point and the orange point denotes the minima. Iterations to convergence are 23, 928, 104 and 45 respectively.

Results A comparison of optimisation trajectories on two kinds of meta-test quadratic problems is shown in Figure 5.1 and Figure 5.2. All the optimisers are initialised in the same position and stopped when the norms of the gradient are smaller than the same threshold. All the optimisers reach the minima, but those that have not meta-learned the curvature information in this family of problems require many more iterations to converge compared with MetaMD.

5.6.2 Learning Mirror Descent for Neural Networks

RotatedMNIST and MLPs We first evaluate optimiser learning for neural networks using the RotatedMNIST dataset and a 3-layer MLP architecture. RotatedMNIST

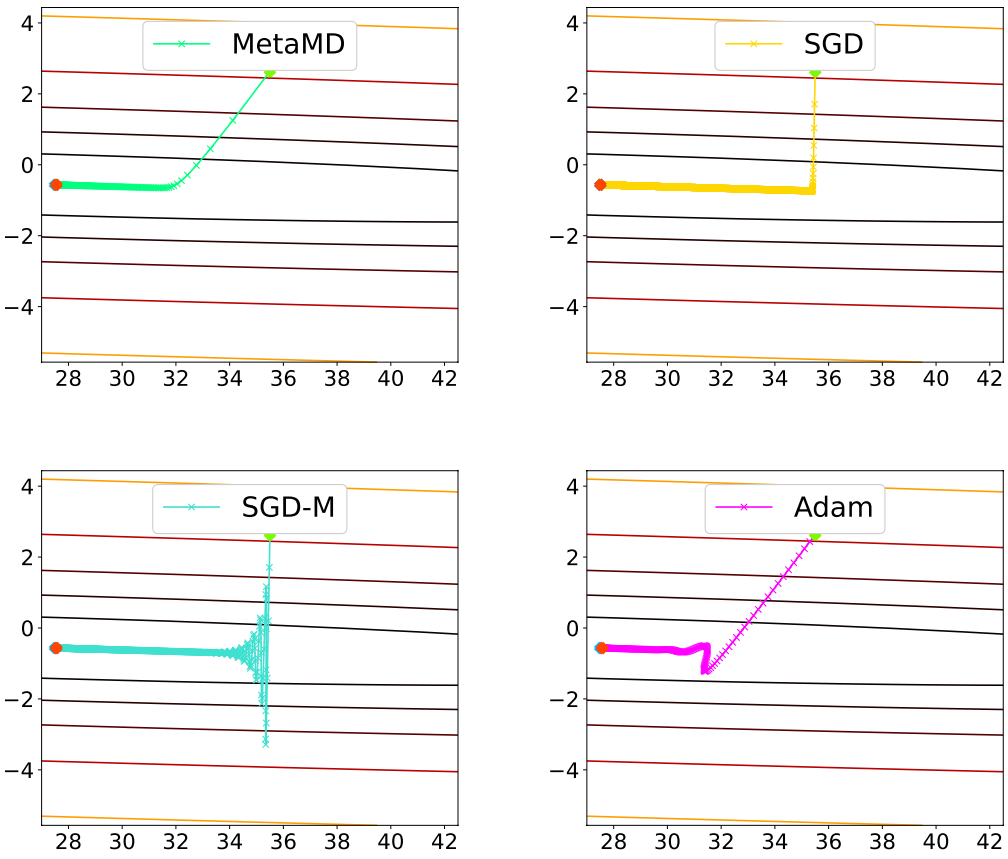

Figure 5.2: Trajectory comparison of different optimisers on a quadratic optimisation problems. Top: MetaMD, SGD. Bottom: SGD-M and Adam. The point green point denotes the starting point and the orange point denotes the minima. Iterations to convergence are 280, 27,328, 1,838 and 324 respectively. MetaMD is the fastest.

defines 6 domains by rotating the original MNIST dataset by 0, 15, 30, 45, 60 and 75 degrees. We use 5 domains for meta-training, and train MetaMD to convergence in the inner loop, and evaluate the performance on the held-out domain. This process is repeated, holding out each domain in turn as meta-test. The convergence curve is shown in Fig. 5.3, and the testing performance in Table 5.1(top). We can see that MetaMD converges rapidly and trains models with strong testing performance. The hyperparameter tuning protocol for this and other experiments in this section is explained in Appendix C.3.

Diverse Digit Datasets and Small CNNs Next we explore applying MetaMD to a more diverse set of datasets and CNN classifiers. We collect a set of of datasets, which we denote as DiverseDigits. They include: MNIST (LeCun and Cortes, 2010),

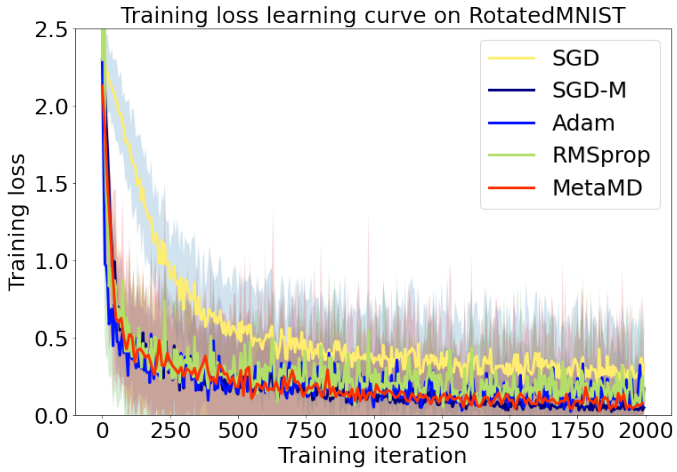

Figure 5.3: Comparison of training loss curves for different optimisers: RotatedMNIST, averaged over all held-out domains.

QMNIST (Yadav and Bottou, 2019), KMNIST (Clanuwat et al., 2018), FashionMNIST (Xiao et al., 2017), USPS (Hull, 1994) and SVHN (Netzer et al., 2011). We train LeNet classifier using MetaMD, resizing all images to 28×28 greyscale. The same leave-one-dataset-out protocol is used: Each dataset is held out in turn for evaluation after MetaMD is trained on on the other datasets. Compared to the previous RotatedMNIST experiment, the distribution of tasks used for meta-training and meta-testing is now more diverse and challenging. Due to the greater cost of training the base model here, we use $T = 500$ iterations for the inner loop, and leave efficient meta-learning under longer-horizons as future work. We compare all methods fairly using a common hyper-parameter (learning rate, weight-decay, etc) tuning protocol for meta-test. Specifically, we perform grid search with respect to meta-test validation accuracy for each competitor and more detail is given in Appendix C.3.

The results averaged over 3 meta-test trials are shown as testing performance at convergence in Table 5.1(bottom) and selected meta-test learning curves in Fig. 5.5, with the remaining learning curves given in Appendix C.4. We can see that MetaMD is clearly faster than SGD and SGD-M in training convergence (Fig. 5.5), while typically producing models with the strongest generalisation error (Table 5.1). It is noteworthy that MetaMD exhibits strong cross-dataset generalisation here, corroborating our Theorem 2 on cross-task optimiser generalisation.

keypointApplication to ResNet18 and CIFAR10 We next explore scaling up to deeper architectures, aiming to train a ResNet18 on CIFAR10 as a held out testing task. To

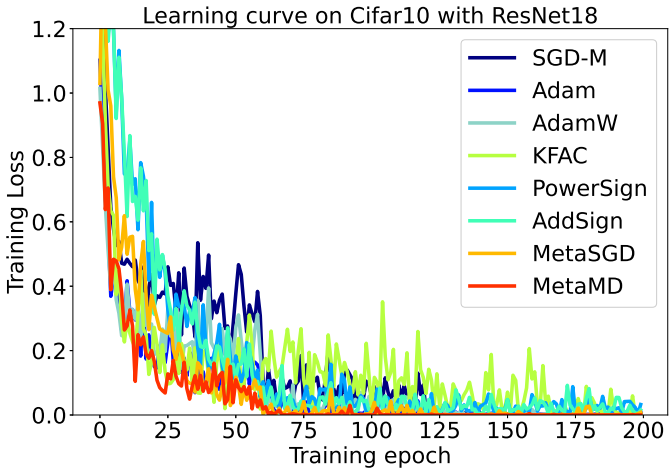

Figure 5.4: Comparison of training loss curves for different optimisers: Right: Training loss curve of ResNet18 trained on CIFAR10

this end we construct a suite of meta-training datasets by combining STL10 (Coates et al., 2011) and DiverseDigits from the previous setting. ResNet18+CIFAR10 is a well-studied problem with lots of known tuning tricks for standard optimisers. For fair comparison, we therefore tune all methods with exactly the same BayesOpt-based hyperparameter tuning protocol, based on CIFAR10 validation performance. For this experiment we also compare some other learned optimisers, including: (i) PowerSign and AddSign that are not directly comparable. They were trained on CIFAR10 (for us it is a held out dataset), but on a different architecture (Bello et al., 2017). (ii) KFAC which is directly comparable insofar as using the same structure of preconditioner to MetaMD, but not comparable in that it computes the preconditioner during meta-test via the Fisher Information Matrix of the target problem – while MetaMD amortizes this cost and meta-learns it from a disjoint set of tasks, without tuning to the target task. (iv) MetaSGD which we re-implement to use exactly the same meta-train data as MetaMD for direct comparability. From the results in Figure 5.4, we can see that MetaMD converges rapidly as expected; and from the results in Table 5.2 we can see that it also provides the best generalisation performance.

High Resolution Image Application Finally, we explore benchmarks with higher resolution images with ResNet18. We construct a suite of datasets including Aircraft (Maji et al., 2013), Butterfly (Chen et al., 2018), Pets (Parkhi et al., 2012), Caltech (Griffin et al., 2007), DTD (Cimpoi et al., 2014), Flowers (Nilsback and Zisserman, 2008), Pubfig (Pinto et al., 2011). See Appendix C.5 for the details about each. Compared with

Table 5.1: Test Accuracy (%) on RotatedMNIST and DiverseDigits with 3-layer MLP and LeNet respectivly. Each column is a test dataset, and MetaMD is trained on the other datasets.

	Test domain	0	15	30	45	60	75
3-Layer MLP	SGD	92.23 ± 0.57	91.91 ± 0.49	92.57 ± 0.32	92.89 ± 0.35	92.73 ± 0.32	92.36 ± 0.87
	SGD-M	94.77 ± 0.58	94.64 ± 0.14	94.66 ± 0.29	94.67 ± 0.47	94.60 ± 0.47	94.47 ± 0.63
	Adam	92.96 ± 0.58	93.29 ± 0.92	93.51 ± 0.84	93.69 ± 0.99	93.67 ± 0.35	92.98 ± 1.17
	RMSprop	92.48 ± 0.49	93.56 ± 0.51	92.77 ± 0.50	93.58 ± 0.32	93.43 ± 0.32	93.14 ± 0.31
	MetaMD	**95.67** ± 0.39	**95.59** ± 0.28	**95.50** ± 0.61	**95.33** ± 0.45	**95.03** ± 0.63	**94.99** ± 0.52
	Test domain	MNIST	QMNIST	KMNIST	FashionMNIST	USPS	SVHN
LeNet	SGD	96.44 ± 0.91	96.23 ± 0.73	87.61 ± 1.87	**88.95** ± 1.43	92.73 ± 1.13	85.44 ± 1.22
	SGD+M	98.47 ± 0.16	97.21 ± 0.15	92.54 ± 0.62	86.44 ± 0.45	95.37 ± 0.24	86.26 ± 0.48
	Adam	98.49 ± 0.17	98.10 ± 0.33	93.20 ± 0.82	87.36 ± 0.55	93.68 ± 0.38	**87.07** ± 0.61
	RMSprop	98.65 ± 0.21	98.30 ± 0.09	93.14 ± 0.87	87.45 ± 0.13	95.43 ± 1.06	87.01 ± 0.18
	MetaMD	**98.72** ± 0.33	**98.74** ± 0.24	**96.56** ± 0.49	87.33 ± 0.36	**95.96** ± 0.14	86.43 ± 0.64

Table 5.2: Test accuracy on CIFAR10 using Resnet18 and various optimisers

Method	KFAC	SGD-M	Adam	AdamW	PowerSign	AddSign	MetaSGD	MetaMD
Accuracy	87.54 ± 0.23	91.37 ± 0.45	91.64 ± 0.56	92.66 ± 0.32	87.77 ± 0.32	88.29 ± 0.67	91.54 ± 0.17	**93.56** ± 0.34

the ($28\times$ and $32\times$) images used in the previous experiment settings, these datasets are much higher resolution images. We resize them to standard 224×224 for ResNet18. We divide this suite of datasets into meta-train and meta-test groups. Meta-train contains Aircraft, Butterfly and Pets. Meta-test has Caltech, DTD, Flowers, and Pubfig. MetaMD and Meta-SGD, a meta-learning based competitor, are trained on the meta-train domains. For a fair comparison, the initialisation learning module in Meta-SGD is turned off and it only meta-learns learning rate. Our algorithm requires the model trained to be converged in each outer loop iteration. To improve meta-training efficiency, we therefore train the base model on 20 subcategories of each domain which are randomly sampled in each outer loop iteration. All the competitors are tested on the meta-test tasks by training ResNet18 from random initialisation for each dataset individually. We compare the learning curve and the test set performance of the learned model on each dataset. The learning curves on Pubfig are shown in Fig. 5.6 and the others are in Appendix C.5. One can notice that all competitors converge in training loss and their training accuracies reach 100%. Due to comparatively limited training examples in these datasets, there is likely over-fitting. Nevertheless, MetaMD exhibits rapid convergence as expected, and also the knowledge transfer from source tasks reduces

Table 5.3: Test accuracy (%) on high resolution datasets. Comparison with meta-learned and manually-designed optimisers.

Datasets	Caltech	DTD	Flowers	Pubfig
SGD-M	26.95 ± 0.35	32.66 ± 0.61	47.18 ± 0.17	74.10 ± 0.11
Adam	27.21 ± 0.53	28.09 ± 0.44	41.03 ± 0.48	73.31 ± 0.18
PowerSign	26.11 ± 0.39	31.91 ± 0.31	44.66 ± 0.53	68.25 ± 0.42
AddSign	27.21 ± 0.37	29.52 ± 0.24	44.79 ± 0.47	67.77 ± 0.29
Meta-SGD	25.41 ± 0.58	27.93 ± 0.34	48.77 ± 0.44	66.51 ± 0.36
MetaMD	**37.50** ± 1.59	**41.01** ± 0.87	**54.84** ± 1.62	**77.77** ± 0.87

overfitting and leads to improved testing performance.

5.7 Loss Landscape Analysis

The generalisation ability is reflected by the flatness of the converged loss landscape (Foret et al., 2021). Motivated by this, we compare the converged loss landscapes achieved by MetaMD with those by SGD on the High Resolution Image setting. From Figure 5.7, it can be observed that the landscapes reached by MetaMD are much more flattened than SGD.

5.8 Conclusion

We explored meta-learning optimisers from the Mirror Descent perspective – in practice by meta-learning a Bregman Divergence to manipulate the gradient during base model learning. Our approach has clear theoretical motivation by optimizing a regret bound on the convergence rate, and has both a convergence guarantee and a cross-dataset generalisation guarantee. Empirically, our results demonstrate rapid convergence compared to SGD and strong generalisation and new guarantees vs other fast heuristic optimisers such as Adam.

Limitations and Social Impact: Similar to other meta-learned optimisers (Bello et al., 2017; Li et al., 2017), our framework requires an up-front compute investment of training multiple neural networks. This can be amortised by faster training of

downstream tasks, as we demonstrate. However, our optimiser is architecture specific, and thus the up-front cost investment is substantial if applied to large neural architectures. This imposes a carbon cost and may restrict its use to well resourced institutions, potential negative social impacts. However, on the other hand our good empirical and theoretical transferability may mean that such institutions could pay the up front cost of training MetaMD for large architectures, and then redistribute the associated optimiser for smaller stakeholders to enjoy faster training – similar to how the foundation model (Bommasani et al., 2021) economy is evolving today, but extended to optimisers.

5.9 Future Work

It has been mentioned in the previous chapters that designing the meta-train protocol, by estimating the number of meta-train tasks and constructing a tasks axis in terms of dataset and architecture is essential for meta-train efficiency and meta-generalisation which is also true for optimiser learning. We would not repeat those problems to avoid redundancy. Instead, a more important question that should be answered is why MetaMD achieves significant generalisation performance without minimising the meta-validation error which is usually set as a meta-objective for improving generalisation on the unseen meta-test tasks. We (Gao et al., 2021a, 2022b,a) empirically observe that MetaMD converges to much more flat loss landscapes than those reached by SGD. This observation matches the recent empirical study (Foret et al., 2021). However, it is not satisfactory since this problem should be answered from a more concrete theoretical perspective. In the future, we aim to study the algorithmic stability of the learned optimisers and this will pave the road to understanding their meta-generalisation.

Furthermore, still MetaMD can be refined by changing some technique designs. For example, due to the current Kronecker product parameterisation, the learned optimiser is specific to the meta-train base model architecture. This is less convenient than an architecture-agnostic optimiser which generalises to the arbitrary dataset and architecture combination in the meta-test stage.

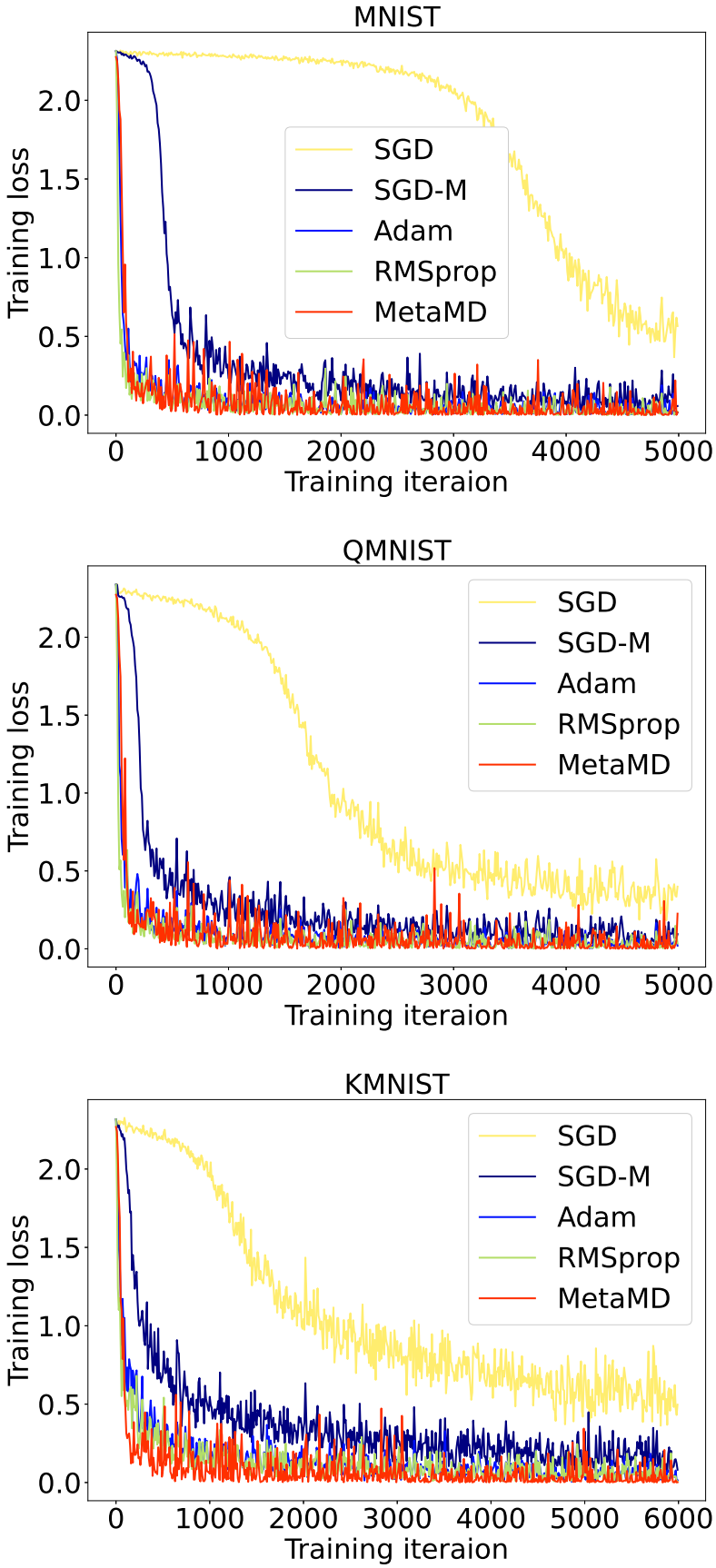

Figure 5.5: Learning curves for DiverseDigits. From left to right: MNIST, QMNIST and KMNIST.

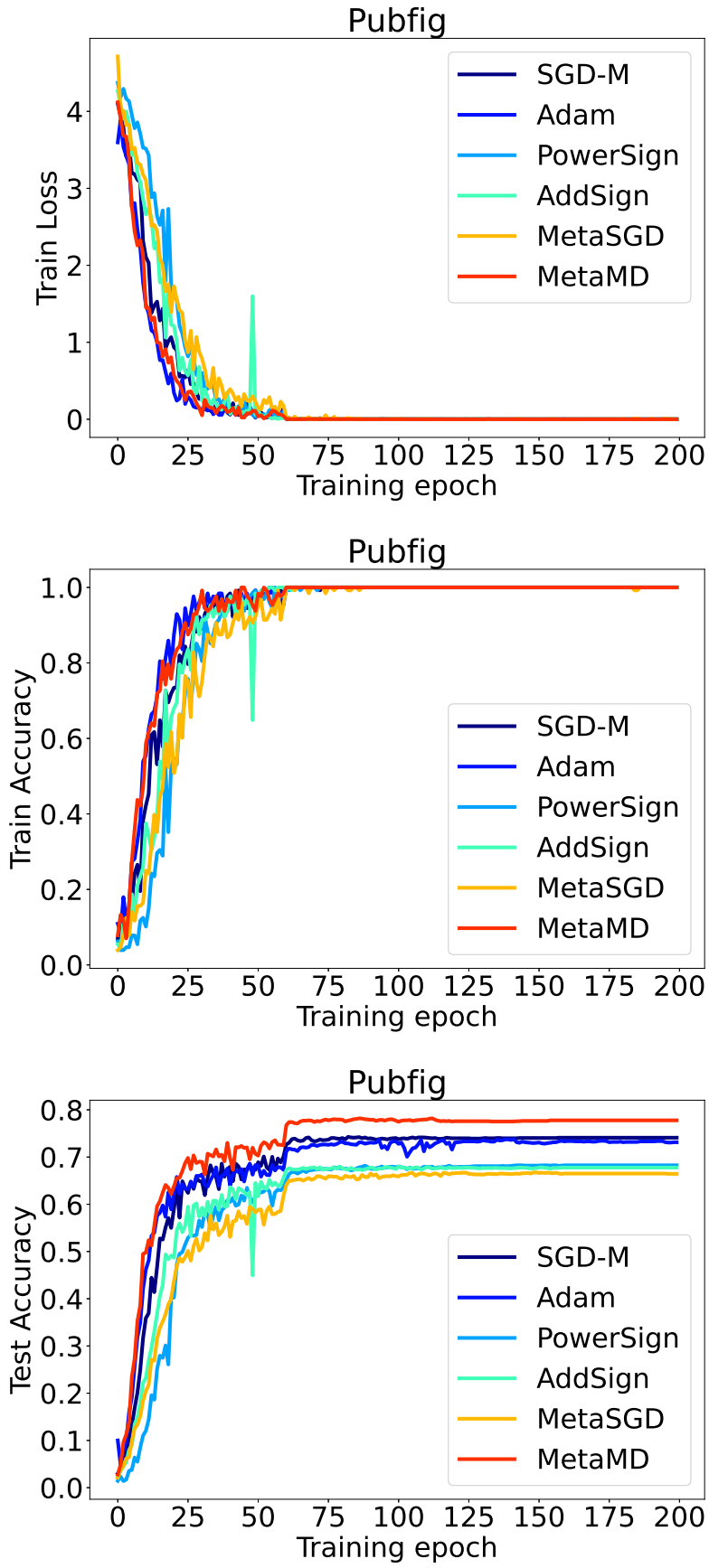

Figure 5.6: Learning curves on Pubfig. The left, middle and right column represent Training loss, Training accuracy and Test accuracy respectively.

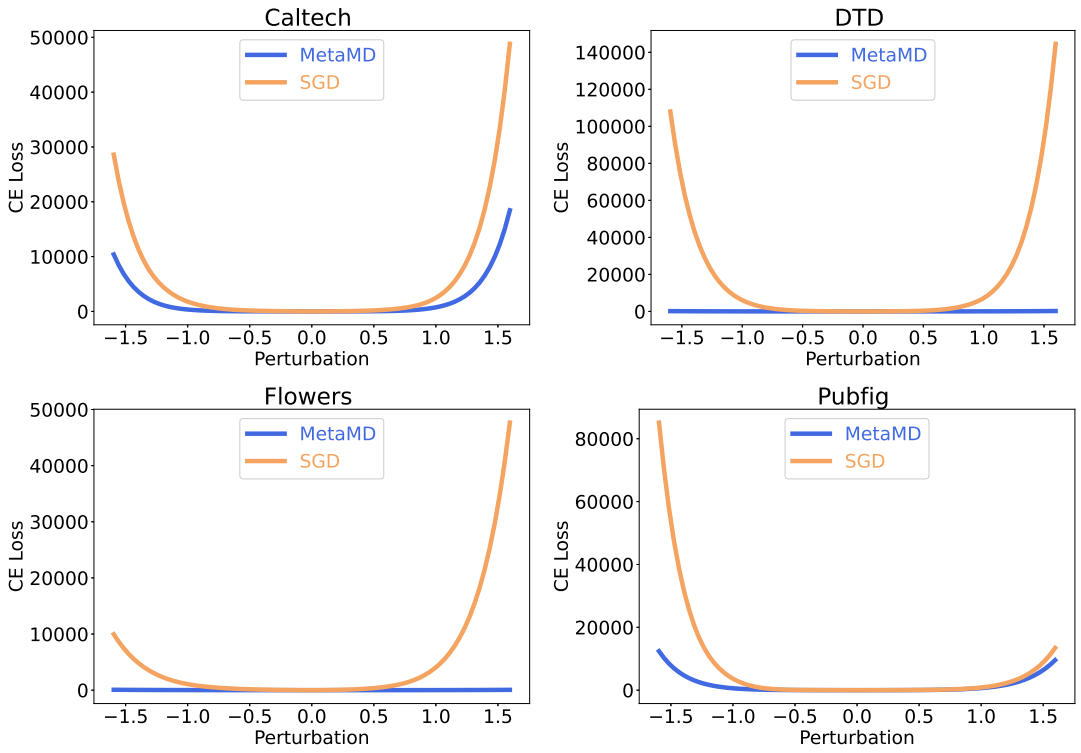

Figure 5.7: 1D Training Loss Landscape on Caltech, DTD, Flowers and Pubfig, where perturbations are performed in the direction of the eigenvector of the loss Hessian matrix corresponding to the largest eigenvalue.

Chapter 6

Conclusion

In this thesis, we study the learning to optimise topics from loss function learning and optimiser learning perspective under the umbrella of meta-learning. Since the neural training process is usually formed as an optimisation problem and the performance of the models highly depends on the supervision provided by the loss function and the update rules from the optimiser, instead of designing such essential components relying on human knowledge, we introduce three algorithms in three different main chapters to automatically search for the desirable loss functions and optimisers through optimising carefully designed bilevel optimisation problem for the target task.

We first study loss function learning in Chapter 3 where an algorithm, searching for robust loss named ARL, is introduced for the noisy label tasks. In Chapter 4, a more efficient loss learning algorithm, ITL which accelerates the learning speed and reduces the learning cost in the meta-train stage, is proposed. By simply deploying the learned loss functions, ARL and ITL, to their corresponding tasks with standard training protocols, the trained model outperforms a variety of state-of-the-art methods in terms of robustness to the noise data and the domain shifts. Comparing the shape of our learned loss function and commonly used Cross Entropy, we observe that the loss function prevents the models to overfit training data, including the noisy data and source domain data, by modifying the model's fixed point and softening the supervision signals. In Chapter 5, MetaMD is introduced to search for the strongly convex function to parameterise Bregman divergence in Mirror Descent to learn update rules. By optimising the convergence rate, the learned optimiser outperforms the stander hand-designed first-order optimiser on a variety of tasks in both generalisation ability and

convergence rate.

Compared with the prior works, our learned components have advanced meta-learning efficiency and transferability with some level of interpretability. Instead of blindly applying neural networks, we carefully chose the loss function and the optimiser parameterisation for learning efficiency. To gain the "plug-to-play" property, all the algorithms perform offline meta-learning fashion, and to avoid short-term bias and to get unbiased update information, the target meta-models are exposed to the whole base model training to gain the best responses. Even though these designs make the meta-train process of learning loss function and optimiser extremely expensive and complex, the learning results are simple to use and can be generalised to large-scale tasks, which is much cheaper than the deployment stage where online meta-learning methods are applied.

Despite the good performance and the properties of the proposed algorithm and learned optimisation components, there are still some hard questions that we cannot answer. We leave them to the future work:

1. The loss functions learned from ARL, and ITL, have analytical form and intuitively interpretability through the plot. However, there is any direct theoretical response for why those losses work?

2. The parameterisation for Bregman divergence is an important design factor in MetaMD. Current Kronecker Product parameterisation is only a static version. Can adaptive parameterisation respond to the local geometry?

3. Meta generalisation like the generalisation in conventional machine learning is an essential topic in a meta-learning setting. Is there any way to incorporate generalisation regulariser for both loss function and optimisers in the meta-train stage?

4. We applied IFT to efficiently compute hypergradient for long horizontal in the inner loop training. Is it possible to develop an inner loop-free algorithm for loss function learning?

5. To prevent the short horizon bias, we conduct an expensive full inner loop in every meta-train iteration. Can we develop evaluation surrogates for the inner loop, which could be archived through another machine learning process, to make the meta-training more efficient with low-cost inner loop?

6. We have compared the genetic algorithm-based and gradient-based searching methods for loss function learning in the domain generalisation in Chapter 4. By comparing the learning curves of two types of algorithms, we can notice the genetic algorithm (CMA-ES) is explorative and the gradient-based one (Implicit-gradient) is fine-grained. To be more specific, in the early stage of meta-training, CMA-ES can easily find relatively good solutions while implicit-gradient converges slower but better. Exploring the combination of those two algorithms will provide a more efficient outer loop.

7. When we study the MetaMD for deep neural networks, for simplicity we did not consider the double descent effect in the inner loop (Nakkiran et al., 2021). In future work for optimiser learning, measuring and optimising this effect will give a better fit for its application to very deep models.

8. In this work, we have explored when a loss function is given how to learn an optimiser and vice versa. One of the very direct extensions of this work can be jointly learning the loss function and optimiser together.

In general, we want to explain why the meta-learning results work from a learning theory perspective beyond the current empirical study. A clear mathematical understanding of the learned loss function is desirable, from which the gradient can be easily analysed for further study. We shed some light on theoretical optimiser learning, where optimiser learning and convergence analysis were usually in parallel. However, it is believed the adaptive model has great potential. We hope that future work will integrate more representative parameterisation for Bregman divergence. Instead of a block diagonal, the convex deep model, taking as input the local geometry, is one of the potential directions to improve the current model and achieve the architecture-agnostic Bregman divergence design.

6.1 Potential Impact

While our loss function studies, ARL and ITL, and optimiser learning algorithm, MetaMD, have not gotten a lot of attention according to the number of citations from Google Scholar, we still believe that our works have a great potential impact on the researchers who are keen to carry their research in the meta-learning field. Practically, we have developed general frameworks for conducting multi-task offline meta-learning using CMA-ES and Implicit gradient-based optimisation algorithms as meta-optimiser,

which can be easily applied to learn loss functions and optimisers in other machine learning application fields. and the current generated loss functions are reusable for noisy label and domain generalisation tasks. Since those loss functions work as the plus-and-play replacements for the existing handcraft loss function, they are compatible with the training of the feature noisy label and domain generalisation algorithm. The application's potential impact can go beyond the noisy label and domain generalisation tasks. One can extend the loss function learning idea to imbalanced data set problems and even learn the loss function for unsupervised learning and self-supervised learning tasks. Since MetaMD is not domain-specific, the optimiser learning algorithm also has a variety of application potentials. More interestingly, once the parameterisation complexity of MetaMD is reduced and the dependence on the architecture is relaxed, MetaMD can be applied to clients with limited computation resources and heterogenous architectures in the Federated learning setting to gain fast convergence speed and good generalisation performance.

We have some concrete empirical studies about multi-task offline meta-learning by learning the transferable loss functions and optimisers. In our experiments, the strong assumption about the i.i.d. from task distribution seems to be violated, based on which future research can focus on the study of quantifying similarity between the meta-train tasks and measuring the domain shift between meta-train and meta-test. Answering those questions helps to know how the performance of the meta-testing model is guaranteed by the learned meta-representation. This sheds light to design efficient meta-training protocols with theoretical principles, which will give people a better idea about meta-learning.

Appendix A

ARL

A.1 Taylor Polynomial

We illustrate the bi-variate 4^{th} order Taylor series,

$$\begin{aligned}\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) =& \theta_2(\hat{\mathbf{y}}_i - \theta_0) + \frac{1}{2}\theta_3(\hat{\mathbf{y}}_i - \theta_0)^2 + \frac{1}{6}\theta_4(\hat{\mathbf{y}}_i - \theta_0)^3 + \frac{1}{24}\theta_5(\hat{\mathbf{y}}_i - \theta_0)^4 \\ &+ \theta_6(\hat{\mathbf{y}}_i - \theta_0)(\mathbf{y}_i - \theta_1) + \frac{1}{2}\theta_7(\hat{\mathbf{y}}_i - \theta_0)(\mathbf{y}_i - \theta_1)^2 + \frac{1}{2}\theta_8(\hat{\mathbf{y}}_i - \theta_0)^2(\mathbf{y}_i - \theta_1) \\ &+ \frac{1}{6}\theta_9(\hat{\mathbf{y}}_i - \theta_0)^3(\mathbf{y}_i - \theta_1) + \frac{1}{6}\theta_{10}(\hat{\mathbf{y}}_i - \theta_0)(\mathbf{y}_i - \theta_1)^3 + \frac{1}{4}\theta_{11}(\hat{\mathbf{y}}_i - \theta_0)^2(\mathbf{y}_i - \theta_1)^2 \\ &+ \theta_{12}(\mathbf{y}_i - \theta_1) + \frac{1}{2}\theta_{13}(\mathbf{y}_i - \theta_1)^2 + \frac{1}{6}\theta_{14}(\mathbf{y}_i - \theta_1)^3 + \frac{1}{24}\theta_{15}(\mathbf{y}_i - \theta_1)^4,\end{aligned} \tag{A.1}$$

to compare with the ARL format presented in Equation 3.5. The terms only containing the ground truth, $\mathbf{y}$, are kept in Equation A.1 and these terms will not effect the training as no gradients are backpropagated into the network through $\mathbf{y}$.

A.2 Architecture of Neural networks

The architectures we used to conduct architecture randomisation and dataset randomisation are given in Table A.1. In architecture randomisation, 2-Layer MLP, 3-Layer MLP and 4-layer CNN are applied. For the dataset randomisation, only 4-Layer CNN are utilised. JoCoR-Net is only used in the dataset-specific loss learning task.

Table A.1: The architectures used in the experiments

2-Layer MLP	3-Layer MLP	4-Layer CNN	JoCoR-Net
28×28 Gray Image	28×28 Gray Image	32×32 RGB Image	32×32 RGB Image
FC $28 \times 28 \longrightarrow 256$, ReLU	FC $28 \times 28 \longrightarrow 256$, ReLU FC256 $\longrightarrow 256$, ReLU	5×5, 32, ReLU 5×5, 64, ReLU FC $1024 \longrightarrow 1024$, ReLU	3×3, 64 BN, ReLU 3×3, 64 BN, ReLU 2×2 Max-pool 3×3, 128 BN, ReLU 3×3, 128 BN, ReLU 2×2 Max-pool 3×3, 196 BN, ReLU 3×3, 196 BN, ReLU 8×8 Avg-pool
FC $256 \longrightarrow 10$	FC $256 \longrightarrow 10$	FC $1024 \longrightarrow 10$	FC $196 \longrightarrow 100$

A.3 Further Implementation Details

We make use of normalisation to ensure the loss values are bounded in a well behaved range for CMA-ES loss function search,

$$\hat{f} = \eta \frac{f - f_{min}}{f_{max} - f_{min}}, \tag{A.2}$$

where f_{min} and f_{max} denotes the minimum and maximum and η is a hyperparameter deciding the dynamic range of the loss function. Both f_{min} and f_{max} are easily approximated by sampling random points satisfying $\{(\hat{\mathbf{y}}, \mathbf{y}) | \hat{\mathbf{y}}_i \geq 0, \sum_i \hat{\mathbf{y}}_i = 1; \mathbf{y}_i \in \{0,1\}, \sum_i \mathbf{y} = 1\}$, which defines the domain of the loss function. Note that this can help improve CMA-ES optimisation stability during meta-training.

A.4 Details of Learned Losses

For reproducibility, we give the complete parameters of the learned ARL from each training condition in Table A.2 and their corresponding plots in Figure A.1. They all have similar shape to the example shown in Fig 3.2.

A.5 Experimental datasets

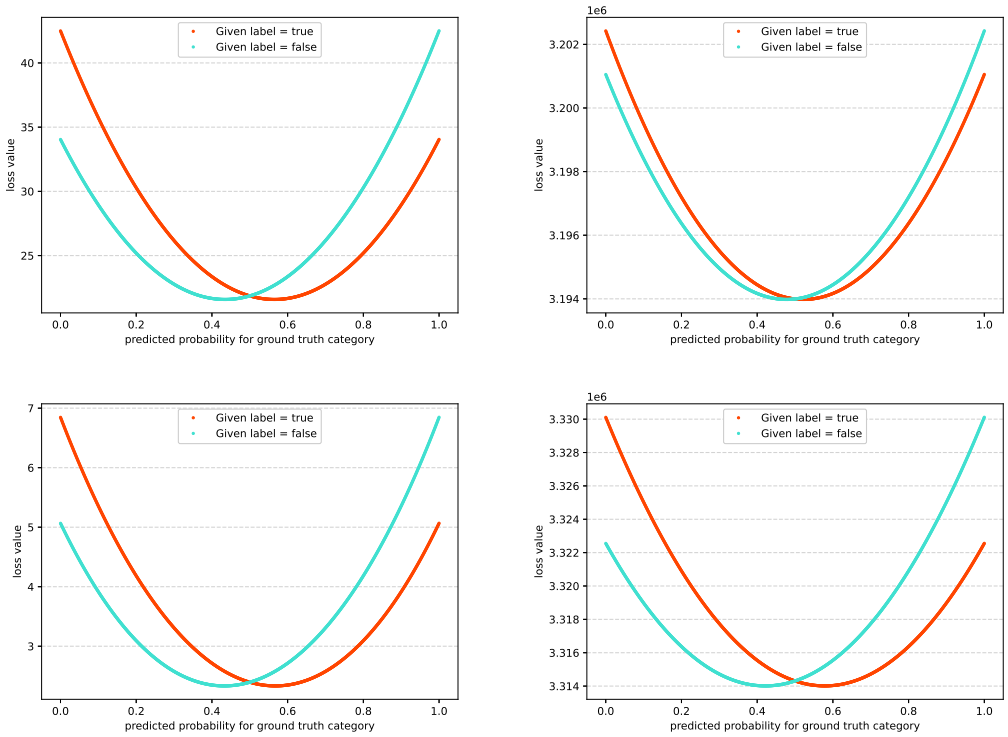

Figure A.1: The plot of the learned loss function. Top: from left to right: ARL(AR-A40), ARL(DR-D40). Bottom: from left to right: ARL(AR-S80), ARL(DR-S80).

Table A.2: The parameters of the learned ARL

Loss type	parameters $\theta_0, \theta_1, ..., \theta_{11}$
ARL(AR-A40)	0.4397, 0.9187, -0.4554, 6.0881, -0.5869, 1.6765, -2.8526, 0.5748, -4.2756, 1.3126, -0.3326, 1.7649
ARL(DR-A40)	1.3754, 25.9954, 8.7507, 3.6501, 9.1560, -2.8144, -5.4006, -7.9342, -8.3143, 6.5751, 3.6237, -2.3279
ARL(AR-S80)	0.7043, -0.4858, -0.0541, -1.8132, 2.2370, 2.4009, 1.4415, 6.4514, -2.7152, -2.6928, -3.2049, -0.5511
ARL(DR-S80)	2.7187, 25.0616, -23.2701, 17.0804, -21.4348, 44.8821, -13.6956, 27.6005, -17.3943, 43.7386, 7.2981, -17.5286

Table A.3: The datasets used in the experiments.

	number of training	number of test	number of class	image size
MNIST	60,000	10,000	10	28×28
KMNIST	60,000	10,000	10	28×28
USPS	7,291	2,007	10	16×16
FashionMNIST	60,000	10,000	10	28×28
CIFAR-10	50,000	10,000	10	32×32
CIFAR-100	50,000	10,000	100	32×32
Clothing1M	1,000,000	10,000	14	224×224

Appendix B

ITL

B.1 Appendix

B.1.1 The Learned Loss Function

B.1.1.0.1 Taylor Loss Family We apply fourth order bi-variate Taylor polynomial to parameterise the learnable loss function. The terms only contain $\mathbf{y}_i$ are removed from the polynomial since these do not generate gradients with respect to the prediction of the network. The final loss function contains 12 learnable parameters ω and can be expressed as

$$\begin{aligned}\mathcal{L}_{\omega}^{(i)}(\hat{\mathbf{y}}_i, \mathbf{y}_i) = \omega_2(\hat{\mathbf{y}}_i - \omega_0) + \omega_3(\hat{\mathbf{y}}_i - \omega_0)^2 + \omega_4(\hat{\mathbf{y}}_i - \omega_0)^3 + \omega_5(\hat{\mathbf{y}}_i - \omega_0)^4 \\ + \omega_6(\hat{\mathbf{y}}_i - \omega_0)(\mathbf{y}_i - \omega_1) + \omega_7(\hat{\mathbf{y}}_i - \omega_0)(\mathbf{y}_i - \omega_1)^2 + \omega_8(\hat{\mathbf{y}}_i - \omega_0)^2(\mathbf{y}_i - \omega_1) \\ + \omega_9(\hat{\mathbf{y}}_i - \omega_0)^3(\mathbf{y}_i - \omega_1) + \omega_{10}(\hat{\mathbf{y}}_i - \omega_0)(\mathbf{y}_i - \omega_1)^3 + \omega_{11}(\hat{\mathbf{y}}_i - \omega_0)^2(\mathbf{y}_i - \omega_1)^2.\end{aligned} \tag{B.1}$$

B.1.1.0.2 ITL Recall that our experimental design trained a single loss function on RotatedMNIST, and then evaluated it from different perspectives across all our main experiments. The specific set of loss function parameters for ITL (as plotted in Fig. 4.5) are given in Table B.1. Then reader can plug-and-play for their own Domain Generalisation problems.

Table B.1: The parameters of the learned ITL

Loss	parameters $\omega_0, \omega_2, ..., \omega_{11}$
ITL	-2.0193, -1.2234, 0.1363, 0.1269, -0.4566, -0.1016, -0.2545, 1.0971, -0.9203, 0.2368, 0.4795, 0.9975

B.1.2 Further Analysis

B.1.2.0.1 Entropy Analysis Since FOCAL (Mukhoti et al., 2020) is the most visually similar to ITL in Figure 4.5, and FOCAL is designed for regularising networks to reduce overconfidence, one might ask whether ITL's good performance can be can be trivially replicated by existing regularisation techniques that simply reduce network confidence. As a measure of confidence, we report the evolution of the target-domain entropy of correctly and incorrectly classified samples during training in Figure B.1(left) and Figure B.1(right) respectively. Clearly, a byproduct of ITL compared to CE and SCE is a (desirable) increase in uncertainty for misclassified instances. FOCAL has similar behaviour to ITL in general, especially for the entropy of prediction distributions for misclassified instances. However, the results in Table 4.3, showed that these alternative losses do not provide strong DG performance compared to ITL. Together this shows that ITL's performance can not simply be replicated by standard over-confidence reduction techniques.

Loss landscape analysis and Perturbation analysis As as possible explanation of why ITL outperforms CE, we study the loss landscape at convergence. Keskar et al. (2017); Chaudhari et al. (2019) observed that flatter loss landscapes lead to good generalisation of the learned model. To this end, we compared a 1D slice through the loss landscape of ITL-Net with that of ERM on both source domain and target domains. Namely, we perturb the converged parameters by moving it around though gradient direction which generated by the eigenvector of the Hessian matrix. From Figure B.2, we can see that for each held out target domain, ITL-Nets have flatter loss landscapes compared with models trained by CE with respect to the source domains. To further analyse the quality of the minimas provided by CE and ITL, we follow the perturbation analysis routine in (Keskar et al., 2017; Zhang et al., 2018) by adding multiplicative noise to the weights of the converged models. In Figure B.3, we can observe that the model trained by ITL is much more robust than that trained by ERM with CrossEntropy against a variety of noise perturbations on learned parameters.

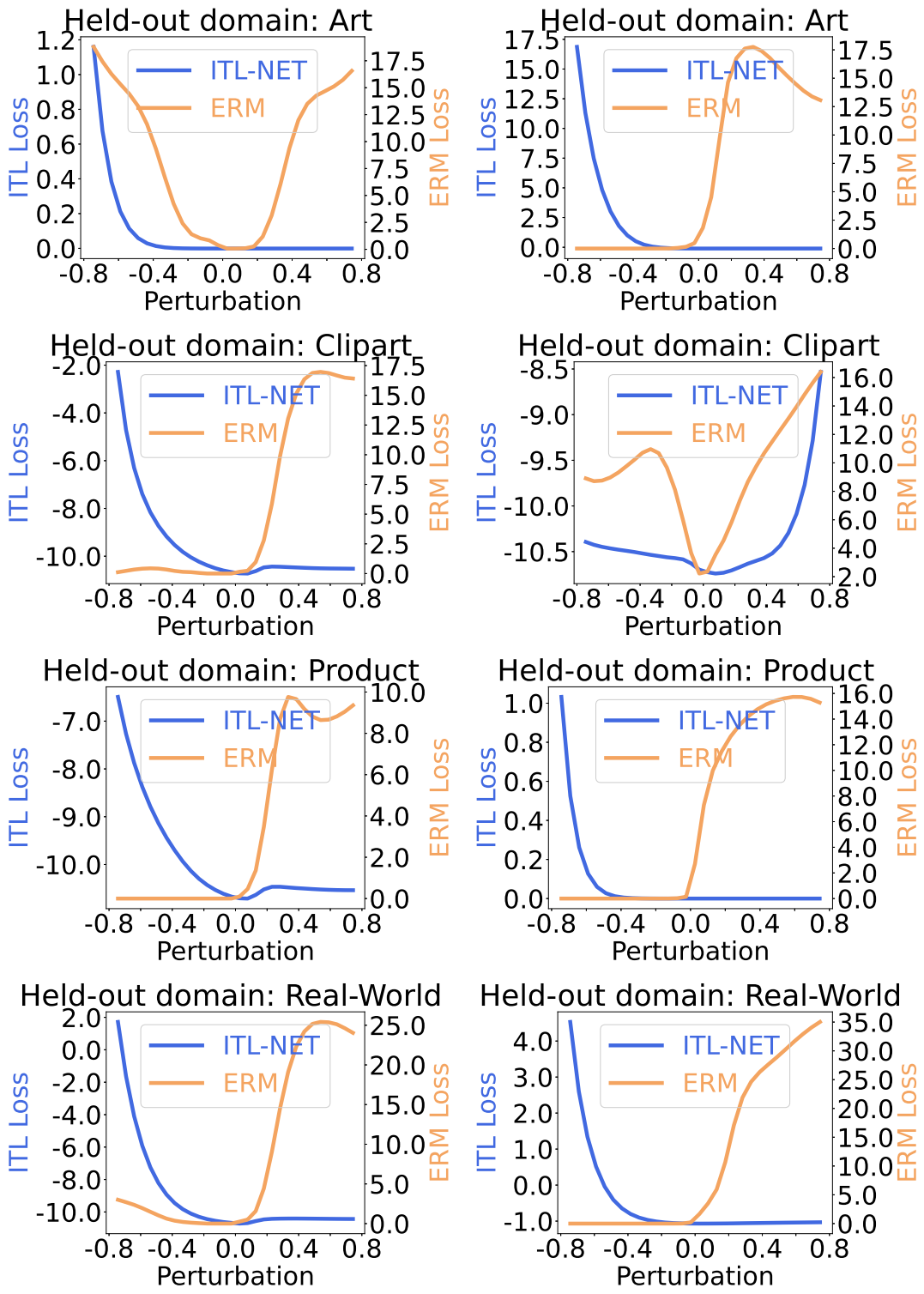

Figure B.2: 1D Loss Landscape: ITL-Net vs ERM on OfficeHome. Left column: Source domain loss landscape. Right column: Target domain loss landscape.

Table B.2: **DomainBed** Cross-domain recognition accuracy (%) with **ResNet50** on **ColoredMNIST** .

	Target set	+90%	+80%	-90%	Avg.
ColoredMNIST	ERM	71.7 ± 0.1	72.9 ± 0.2	10.0 ± 0.1	51.5
	SagNet	71.8 ± 0.2	73.0 ± 0.2	10.3 ± 0.0	51.7
	CORAL	71.6 ± 0.3	73.1 ± 0.1	9.9 ± 0.1	51.5
	CDANN	72.0 ± 0.3	73.0 ± 0.2	10.2 ± 0.1	51.7
	RSC	71.9 ± 0.3	72.9 ± 0.4	10.2 ± 0.2	51.8
	ITL-Net	71.3 ± 0.6	73.4 ± 0.1	11.3 ± 0.7	**52.0**

B.1.3 Detailed results for DomainBed

In Table 4.2 of the main paper, we summarized performance across held out target domains for the standard multi-source DG problem. In Table B.2, Table B.3 and Table B.4, we now give the detailed results of ITL for each target domain.

B.1.4 Detailed Results for Single Source Domain Experiment

In Table 4.4 of the main paper, we reported single-source DG results, summarising performance across choice of source domain and over all target domains. In Table B.5, B.6, B.7, B.8 we now give the detailed results of ITL-Net for each choice of source and target domain.

B.1.5 Meta-train compute cost

Due to the efficiency of implicit gradient, training our ITL required using PyTorch (Paszke et al., 2017) only required 8 hours on a single V100 GPU to complete 200 gradient descent steps on ω. While the goals and base learning problems are not directly comparable, this is dramatically faster than alternatives that require an entire cluster (Li et al., 2019a), and where even very recent fast methods require about 12 GPU-days (Liu et al., 2021).

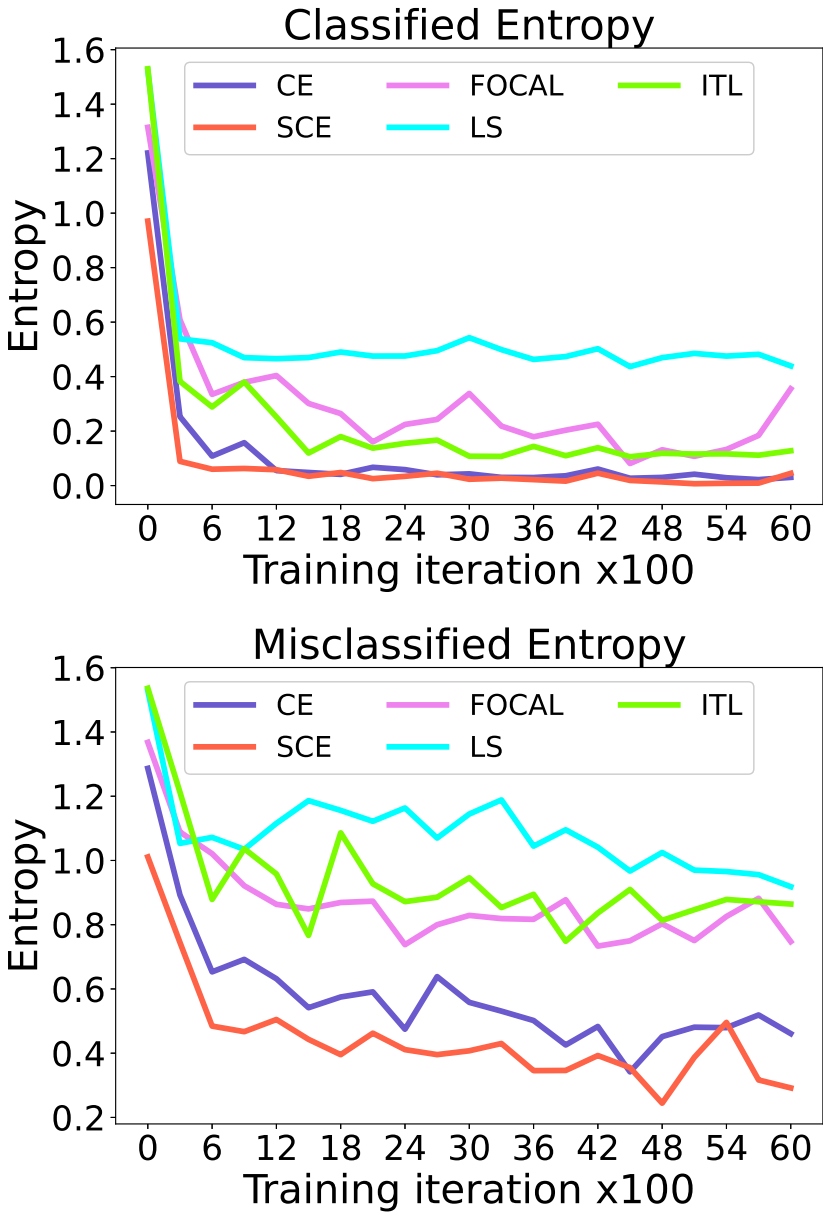

Figure B.1: The evolution of posterior entropy for target domain test samples during training. Top: Correctly classified samples. Bottom: Misclassified samples.

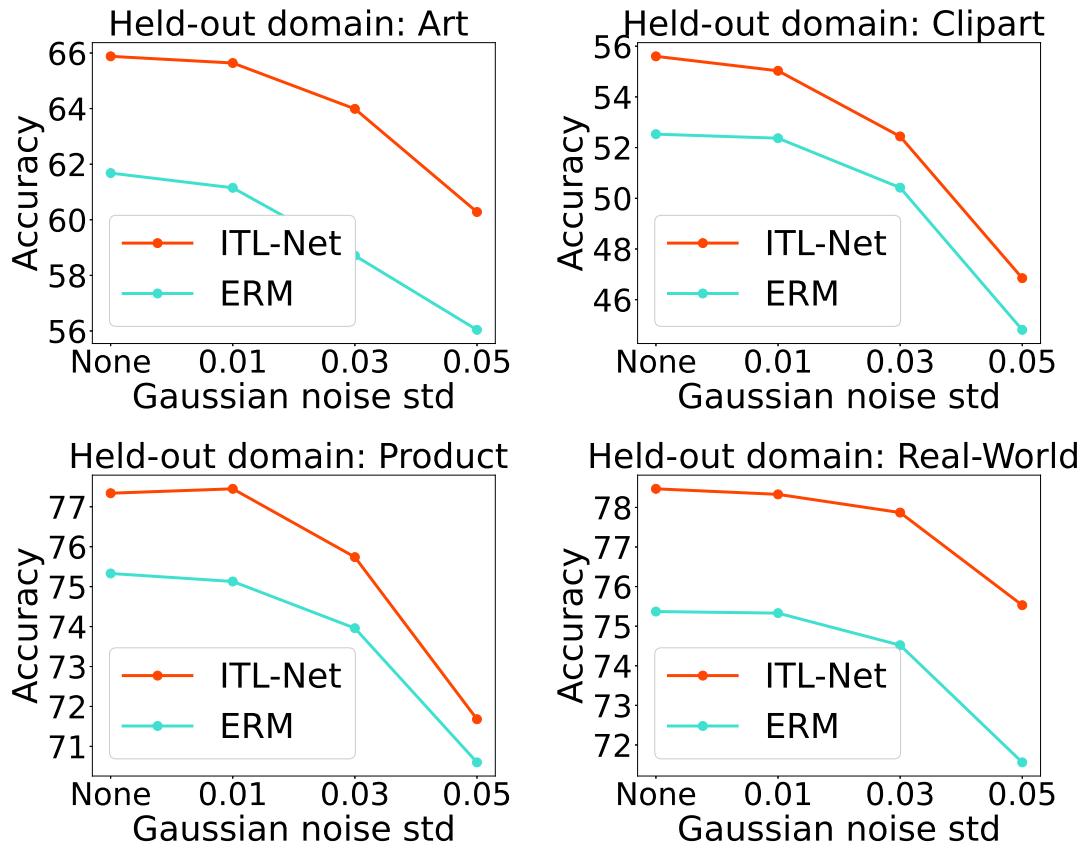

Figure B.3: Perturbation analysis on OfficeHome: ITL-Net vs ERM. Multiplicative Gaussian noise with mean 1 and std: 0.01, 0.05, 0.08 is added to network weights.

Table B.3: **DomainBed** Cross-domain recognition accuracy (%) with **ResNet50** on **VLCS**, **PACS**, **TerraIncognita** and **OfficeHome**.

	Target set	Caltech	Labelme	Sun	V-Pascal	Avg.
VLCS	ERM	97.7 ± 0.4	64.3 ± 0.9	73.4 ± 0.5	77.3 ± 1.3	77.5
	SagNet	97.9 ± 0.4	64.5 ± 0.5	71.4 ± 1.3	77.5 ± 0.5	77.8
	CORAL	98.3 ± 0.1	66.1 ± 1.2	73.4 ± 0.3	77.5 ± 1.2	78.8
	CDANN	97.3 ± 0.3	65.1 ± 1.2	70.7 ± 0.8	77.1 ± 1.5	77.5
	RSC	97.9 ± 0.1	62.5 ± 0.7	72.3 ± 1.2	75.6 ± 0.8	77.1
	ITL-Net	98.3 ± 0.4	65.4 ± 0.7	75.1 ± 0.6	76.8 ± 1.2	**78.9**
	Target set	Art	Cartoon	Photo	Sketch	Avg.
PACS	ERM	84.7 ± 0.4	80.8 ± 0.6	97.2 ± 0.3	79.3 ± 1.0	85.5
	SagNet	87.4 ± 1.0	80.7 ± 0.6	97.1 ± 0.1	80.0 ± 0.4	86.3
	CORAL	88.3 ± 0.2	80.0 ± 0.5	97.5 ± 0.3	78.8 ± 1.3	86.2
	CDANN	84.6 ± 1.8	75.5 ± 0.9	96.8 ± 0.3	73.5 ± 0.6	82.6
	RSC	85.4 ± 0.8	79.1 ± 0.6	96.9 ± 0.5	77.7 ± 1.7	84.9
	ITL-Net	87.1 ± 0.4	83.3 ± 0.6	96.1 ± 0.4	79.3 ± 0.6	**86.4**
	Target set	L100	L38	L43	L46	Avg.
TerraIncognita	ERM	49.8 ± 4.4	42.1 ± 1.4	56.9 ± 1.8	35.7 ± 3.9	46.1
	SagNet	53.0 ± 2.9	43.0 ± 2.5	57.9 ± 0.6	40.4 ± 1.3	48.6
	CORAL	51.6 ± 2.4	42.2 ± 1.0	57.0 ± 1.0	39.8 ± 2.9	47.6
	CDANN	47.0 ± 1.9	41.3 ± 4.8	54.9 ± 1.7	39.8 ± 0.8	45.8
	RSC	50.2 ± 2.2	39.2 ± 1.4	56.3 ± 1.4	40.8 ± 0.6	46.6
	ITL-Net	58.4 ± 3.7	46.2 ± 1.8	58.5 ± 0.9	40.9 ± 1.8	**51.0**
	Target set	Artistic	Clipart	Product	Real World	Avg.
OfficeHome	ERM	61.3 ± 0.7	52.4 ± 0.3	75.8 ± 0.1	76.6 ± 0.3	66.5
	SagNet	63.4 ± 0.2	54.8 ± 0.4	75.8 ± 0.4	78.3 ± 0.3	68.1
	CORAL	65.3 ± 0.4	54.4 ± 0.5	76.5 ± 0.1	78.4 ± 0.5	68.7
	CDANN	61.0 ± 1.4	50.4 ± 2.4	74.4 ± 0.9	76.6 ± 0.8	65.8
	RSC	60.7 ± 1.4	51.4 ± 0.3	74.8 ± 1.1	75.1 ± 1.3	65.5
	ITL-Net	65.6 ± 0.4	55.6 ± 0.4	77.5 ± 0.3	78.6 ± 0.4	**69.3**

Table B.4: **DomainBed** Cross-domain recognition accuracy (%) with **ResNet50** on **DomainNet**.

DomainNet	Target set	Clipart	Infograph	Painting	Quickdraw	Real	Sketch	Avg.
	ERM	58.1 ± 0.3	18.8 ± 0.3	46.7 ± 0.3	12.2 ± 0.4	59.6 ± 0.1	49.8 ± 0.4	40.9
	SagNet	57.7 ± 0.3	19.0 ± 0.2	45.3 ± 0.3	12.7 ± 0.5	58.1 ± 0.5	48.8 ± 0.2	40.3
	CORAL	59.2 ± 0.1	19.7 ± 0.2	46.6 ± 0.3	13.4 ± 0.4	59.8 ± 0.2	50.1 ± 0.6	41.5
	CDANN	54.6 ± 0.4	17.3 ± 0.1	43.7 ± 0.9	12.1 ± 0.7	56.2 ± 0.4	45.9 ± 0.5	38.3
	RSC	55.0 ± 1.2	18.3 ± 0.5	44.4 ± 0.6	12.2 ± 0.2	55.7 ± 0.7	47.8 ± 0.9	38.9
	ITL-Net	63.5 ± 0.3	19.4 ± 0.1	46.3 ± 0.1	13.7 ± 0.4	53.2 ± 0.6	53.5 ± 0.3	**41.6**

Table B.5: **DomainBed** Single source domain recognition accuracy (%) with **ResNet50** on **VLCS**. Each cell reports the accuracy for a set of target domains, and the source domain used for training corresponding to the column. The performance of target domains is separated by '/'. Average over target domains for a given source domain is given at the bottom of the cell.

VLCS	Source set	Caltech	Labelme	Sun	V-Pascal	Avg.
	ERM	47.81/55.58/59.72 54.37	70.00/57.61/65.90 64.5	52.93/62.27/60.30 58.5	96.75/63.29/76.81 78.95	**64.08**
	CORAL	47.81/55.58/59.72 54.37	70.00/57.61/65.90 64.5	52.93/62.27/60.30 58.5	96.75/63.29/76.81 78.95	**64.08**
	SagNet	48.76/53.14/56.93 52.94	35.69/53.53/63.33 50.85	67.28/62.05/66.70 65.34	96.96/62.88/75.20 78.35	61.87
	ITL-Net	43.19/39.85/51.01 44.68	89.82/55.18/60.63 68.54	48.90/61.78/60.01 56.9	97.31/60.84/77.51 78.55	62.17

Table B.6: **DomainBed** Single source domain recognition accuracy (%) with **ResNet50** on **PACS**. Each cell reports the accuracy for a set of target domains, and the source domain used for training corresponding to the column. The performance of target domains is separated by '/'. Average over target domains for a given source domain is given at the bottom of the cell.

	Source set	Art	Cartoon	Photo	Sketch	Avg.
PACS	ERM	65.36/96.23/45.41 69.0	70.17/86.17/66.02 74.12	68.07/20.05/16.62 34.91	24.76/36.05/27.25 29.35	51.85
	CORAL	65.36/96.21/45.41 68.99	70.20/86.15/66.01 74.12	68.03/20.04/16.62 34.9	24.70/36.05/27.24 29.33	51.84
	SagNet	66.60/93.41/55.61 71.87	61.42/79.76/64.93 68.7	69.04/30.38/25.88 41.77	26.17/36.86/25.93 29.65	53.0
	ITL-Net	66.30/94.67/57.29 72.75	74.85/86.52/75.06 78.81	62.60/45.82/51.44 53.29	17.87/26.45/19.64 21.32	**56.54**

Table B.7: **DomainBed** Single source domain recognition accuracy (%) with **ResNet50** on **TerraIncognita**. Each cell reports the accuracy for a set of target domains, and the source domain used for training corresponding to the column. The performance of target domains is separated by '/'. Average over target domains for a given source domain is given at the bottom of the cell.

	Source set	L100	L38	L43	L46	Avg.
TerraIncognita	ERM	43.82/60.93/69.15 57.97	39.97/51.27/54.40 48.55	40.96/39.11/64.70 48.26	58.26/46.53/73.73 59.51	53.57
	CORAL	43.80/60.90/69.17 57.96	40.00/51.12/54.20 48.44	40.87/40.12/64.23 48.41	58.70/45.43/73.62 59.25	53.51
	SagNet	40.71/56.95/68.19 55.28	37.95/50.44/53.71 47.37	33.99/34.41/59.01 42.47	59.54/45.93/74.72 60.06	51.3
	ITL-Net	44.79/55.71/67.62 56.04	44.66/53.62/58.07 52.12	40.97/39.29/66.35 48.87	61.68/51.36/76.41 63.15	**55.04**

Table B.8: **DomainBed** Single source domain recognition accuracy (%) with **ResNet50** on **OfficeHome**. Each cell reports the accuracy for a set of target domains, and the source domain used for training corresponding to the column. The performance of target domains is separated by '/'. Average over target domains for a given source domain is given at the bottom of the cell.

	Source set	Artistic	Clipart	Product	Real World	Avg.
OfficeHome	ERM	44.75/23.36/21.09 29.73	39.50/18.66/20.11 26.09	37.45/26.00/39.55 34.33	27.67/31.44/56.00 38.37	32.13
	CORAL	44.75/23.36/21.09 29.73	39.49/18.67/20.11 26.09	37.45/26.01/39.55 34.34	27.77/31.46/55.98 38.4	32.14
	SagNet	47.98/22.33/17.85 29.39	46.40/23.98/14.89 28.42	34.84/34.34/41.62 36.93	33.30/35.41/54.25 40.99	33.93
	ITL-Net	31.63/21.57/21.53 24.91	54.72/21.79/30.99 35.83	37.38/38.75/36.69 37.61	33.71/35.64/56.64 42.0	**35.09**

Appendix C

MetaMD

C.1 Derive of the closed form mirror loop

In our setting, the mirror loop is described as

$$\theta_{t+1} = \arg\min_{\theta} \langle \nabla_\theta \mathcal{L}(\theta_t), \theta \rangle + \frac{1}{2\eta} B_\phi(\theta || \theta_t) \tag{C.1}$$

or, equivalently,

$$\theta_{t+1} = \arg\min_{\theta} \eta \langle \nabla_\theta \mathcal{L}_{tr}(\theta_t), \theta \rangle + B_\phi(\theta || \theta_t). \tag{C.2}$$

Setting the gradient w.r.t. θ to zero, we have

$$\eta \nabla \mathcal{L}(\theta_t) + \nabla \phi(\theta_{t+1}) - \nabla \phi(\theta_t) = 0, \tag{C.3}$$

which when rearranged yields

$$\nabla \phi(\theta_{t+1}) = \nabla \phi(\theta_t) - \eta \nabla \mathcal{L}(\theta_t) \tag{C.4}$$

$$\theta_{t+1} = \nabla \phi^{-1}(\nabla \phi(\theta_t) - \eta \nabla \mathcal{L}(\theta_t)). \tag{C.5}$$

In our case

$$\phi(\theta) = \frac{1}{2} \theta^T M \theta, \tag{C.6}$$

where M is a block diagonal matrix. Therefore,

$$\nabla \phi(\theta) = M\theta \tag{C.7}$$

$$\nabla \phi^{-1}(\theta) = M^{-1}\theta. \tag{C.8}$$

As a result, we also have

$$\theta_{t+1} = \nabla\phi^{-1}(\nabla\phi(\theta_t) - \eta\nabla\mathcal{L}(\theta_t)) \tag{C.9}$$

$$= M^{-1}(M\theta_t - \eta\nabla\mathcal{L}(\theta_t)) \tag{C.10}$$

$$= \theta_t - \eta M^{-1}\nabla\mathcal{L}(\theta_t). \tag{C.11}$$

C.2 Gradient Computation for Diagonal Matrix

Diagonal matrix is a special case of block diagonal matrix, which we applied in the 2D quadratic setting in section 5.6.2. With this simple parameterisation, the exact hypergradient w.r.t. the target optimiser can be computed with Forward Mode Differential (FMD). The gradient of the second term in the proposed meta-objective in Eq.5.7 is easy to compute while the first term with respect to ϕ is expressed as:

$$\frac{\partial B_\phi(\theta_*, \theta_1)}{\partial\phi} \approx \frac{\partial B_\phi(\theta_T, \theta_1)}{\partial\phi} \tag{C.12}$$

$$= \underbrace{\frac{\partial B_\phi(\theta_T, \theta_1)}{\partial\phi}}_{\text{direct gradient}} + \underbrace{\frac{\partial B_\phi(\theta_T, \theta_1)}{\partial\theta_T}\frac{\partial\theta_T}{\partial\phi}}_{\text{indirect gradient}} \tag{C.13}$$

when T is large enough to satisfy that $\theta_* \approx \theta_T$. The computation of the direct gradient can be easily solved by the existing auto-differentiation library. The indirect gradient in Eq C.13, usually termed hypergradient, is much more computationally chanllenging as it is expressed in the form:

$$\frac{\partial\theta_T}{\partial\phi} = \sum_{t=1}^{T}\left(\prod_{t'=t+1}^{T} A_{t'}\right) B_t \tag{C.14}$$

$$\text{s.t.} A_t = \frac{\partial\pi_\phi(\theta_{t-1})}{\partial\theta_{t-1}} = I - \eta M^{-1}\frac{\partial^2}{\partial\theta^2}\mathcal{L}_{tr}(\theta_{t-1}), \tag{C.15}$$

$$B_t = \frac{\partial\pi_\phi(\theta_{t-1})}{\partial\phi} = 2\eta\frac{\partial}{\partial\theta}\mathcal{L}_{tr}(\theta_{t-1})M^{-2}. \tag{C.16}$$

Forward-Mode Differentiation (FMD) and Reverse-Mode Differentiation Franceschi et al. (2017) are two algorithms to compute Eq C.14. RMD computes the gradient from the last to the initial step, requiring one to store the entire optimisation trajectory in memory. When the optimisation trajectory is long, this computation is very expensive. In comparison, FMD updates the hypergradient in parallel with in inner loop optimisation by:

$$\frac{\partial\theta_t}{\partial\phi} = \frac{\partial\pi(\theta_{t-1})}{\partial\theta_{t-1}}\frac{\partial\theta_{t-1}}{\partial\phi} + \frac{\partial\pi(\theta_{t-1})}{\partial\phi}, \tag{C.17}$$

where it only requires the information from step $t-1$.

C.3 Hyperparameter Tuning

Grid Search For tuning the hyperparameters on 3-layer MLPs model settings in section 5.6.2, we sweep over the learning rates $\{0.1, 0.05, 0.01, 0.005, 0.001\}$ and weight decay parameters of $\{0.001, 0.0001, 0.0005\}$ for the SGD, SGD-M and RMSprop. In terms of Adam, we do grid search over the learn rates $\{0.3, 0.2, 0.1, 0.01, 0.001\}$ and weight decay $\{0.001, 0.0001, 0.0005\}$.

Bayesian Optimisation We implement our BayesOpt using (Balandat et al., 2020) for the hyperparameter tuning on the ResNet18 and CIFAR10 setting. The model the expected performance using a Gaussian process with RBF kernel, which maps the learning rate and weight decay to the estimated validation accuracy. This also provides uncertainty information to the Upper Confidence Bound (UCB) acquisition function for exploring/exploiting the hyperparameter space. For each model selection in the meta-test stage, we run the Bayesian optimisation for 25 iterations.

C.4 Training loss learning curve for DiverseDigits dataset

We give all the training loss learning curves on DiverseDigits in Fig C.1. It can be noticed that the conclusion we drew that MetaMD is clearly faster than SGD and SGD-M in training convergence in Section 5.6.2 is further supported.

C.5 Experiment details on High Resolution Image Datasets

We provide a summary of each dataset applied in our High Resolution tasks. All the datasets except Caltech-256 has standard training, testing splits. As a result, we build our own split with 60 and 20 examples for each classes in training and testing stage respectively. Both MetaMD and competitors are tuned on learning rates, $\{0.1, 0.01, 0.001, 0.0001\}$ and weight decays, $\{0.001, 0.0005, 0.0001, 0.00001, 0\}$. The learning curve for Caltech, DTD, Flowers and Pubfig are shown in Fig C.2, Fig C.3, Fig C.4 and Fig C.4 respectively.

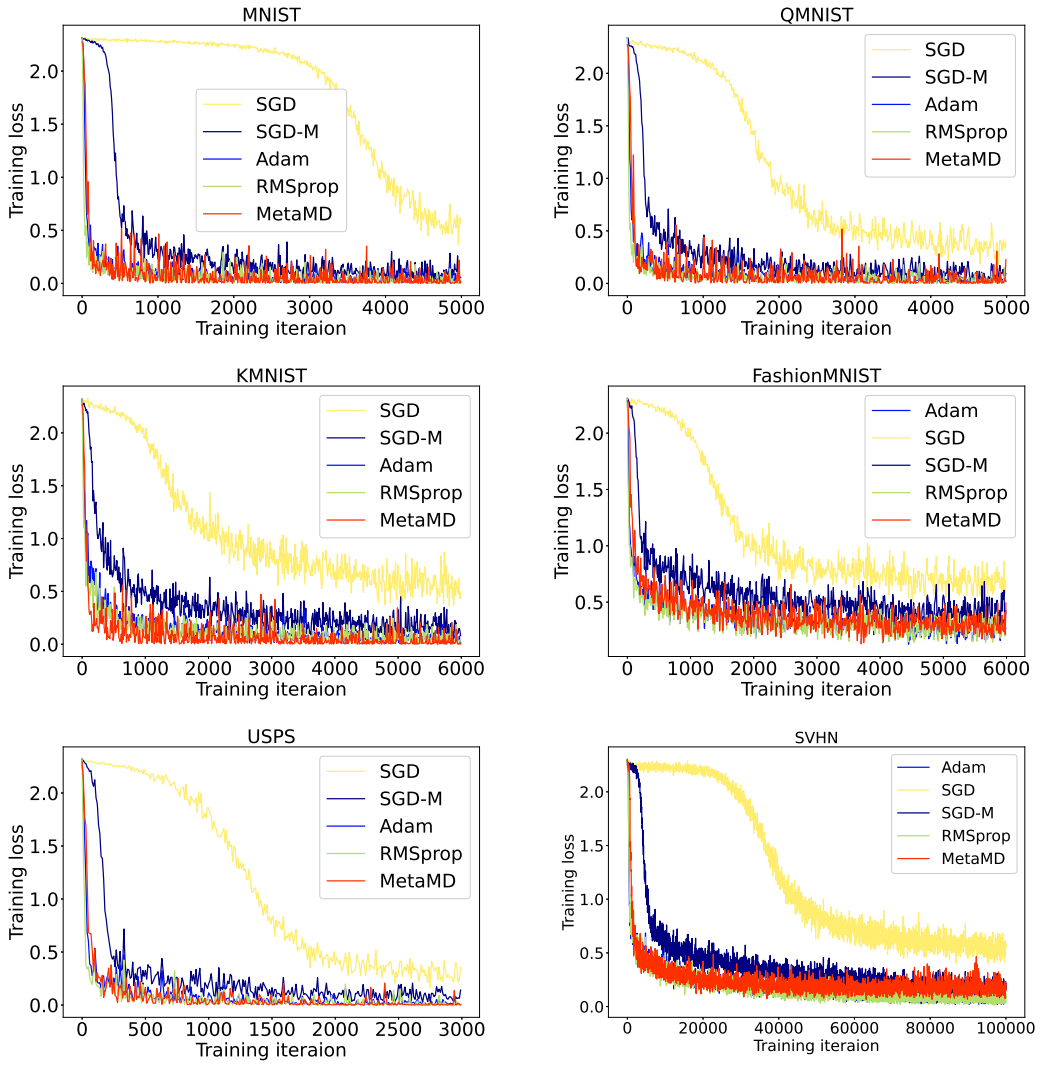

Figure C.1: Convergence comparison of different optimisers on DiverseDigits.

Table C.1: Statistics for the datasets used throughout the experiments. The Train, and Test columns contain the number of instances in each of the corresponding subsets.

Dataset	Train	Test	Classes
Aircraft (Maji et al., 2013)	6,667	3,333	100
Butterfly (Chen et al., 2018)	10,270	15,009	200
Flowers (Nilsback and Zisserman, 2008)	2,040	6,149	102
Pets (Parkhi et al., 2012)	4,000	3,390	37
PubFig (Pinto et al., 2011)	12,178	1,660	83
DTD (Cimpoi et al., 2014)	3,760	1,880	47
Caltech (Griffin et al., 2007)	12,800	5,120	256

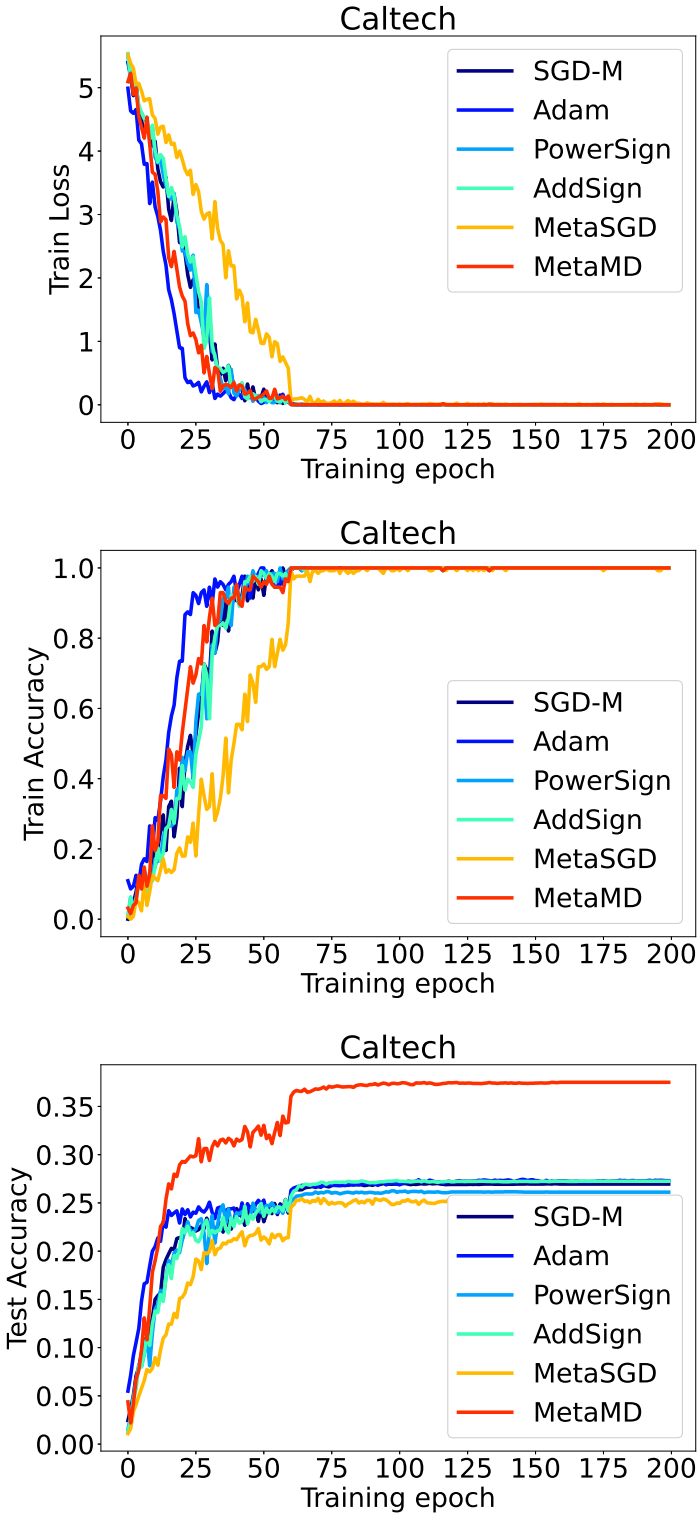

Figure C.2: Learning curves on Caltech. The Top, middle and bottom row represent Training loss, Training accuracy and Test accuracy respectively

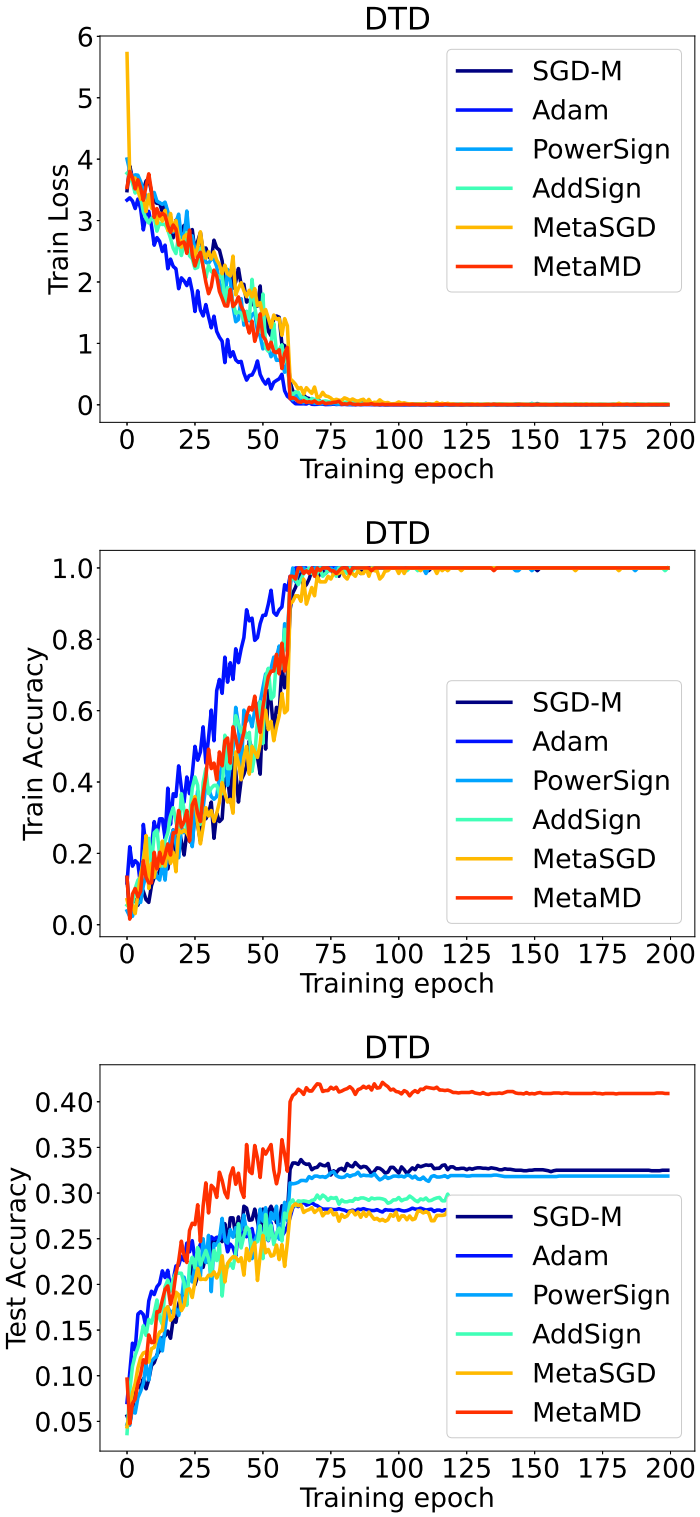

Figure C.3: Learning curves on DTD. The left, middle and right collumn represent Training loss, Training accuracy and Test accuracy respectively

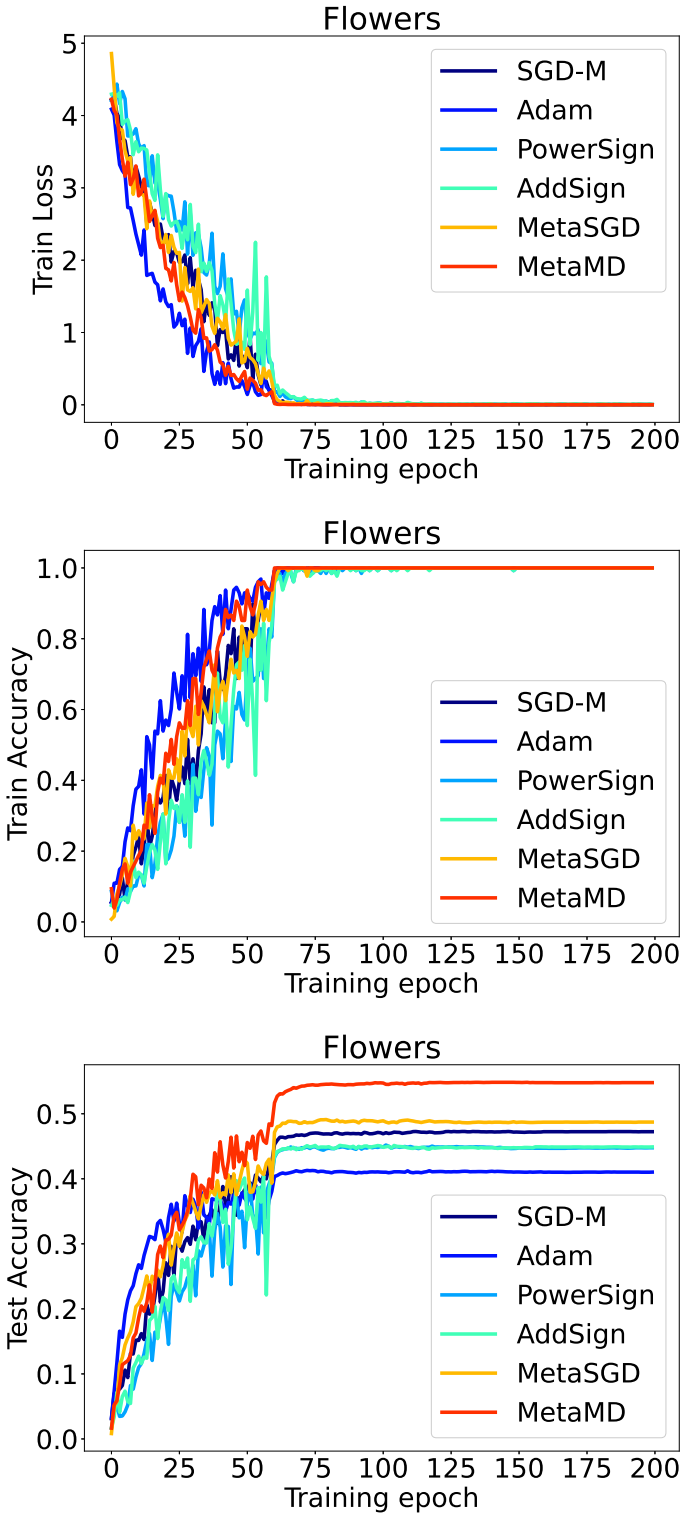

Figure C.4: Learning curves on Flowers. The left, middle and right collumn represent Training loss, Training accuracy and Test accuracy respectively

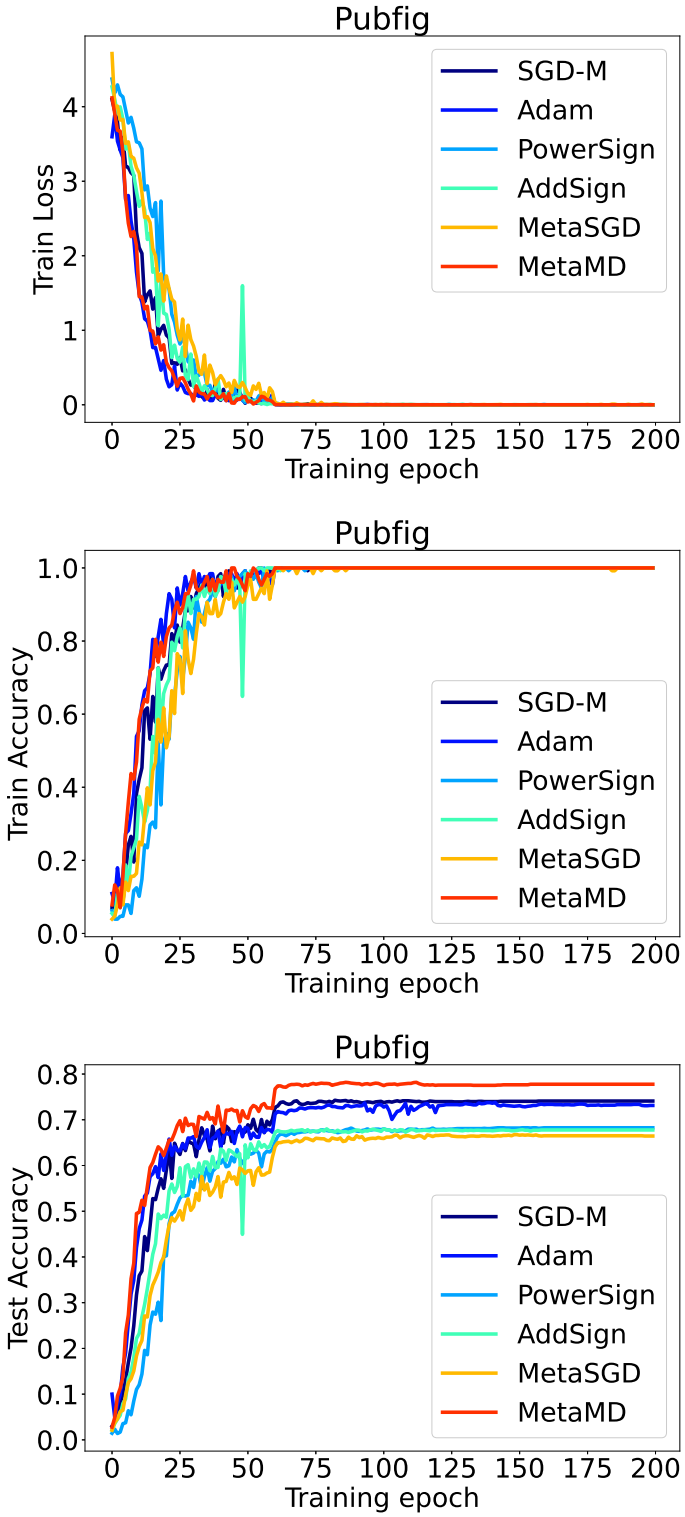

Figure C.5: Learning curves on Pubfig. The left, middle and right collumn represent Training loss, Training accuracy and Test accuracy respectively

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Amid, E., Warmuth, M. K., Anil, R., and Koren, T. (2019a). Robust bi-tempered logistic loss based on bregman divergences. In *NeurIPS*.

Amid, E., Warmuth, M. K., and Srinivasan, S. (2019b). Two-temperature logistic regression based on the tsallis divergence. In *AISTATS*.

Amit, R. and Meir, R. (2018). Meta-learning by adjusting priors based on extended pac-bayes theory. In *International Conference on Machine Learning*, pages 205–214. PMLR.

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. In *NeurIPS*.

Azizan, N., Lale, S., and Hassibi, B. (2021). Stochastic mirror descent on overparameterized nonlinear models. *IEEE Transactions on Neural Networks and Learning Systems*.

Balaji, Y., Sankaranarayanan, S., and Chellappa, R. (2018). Metareg: Towards domain generalization using meta-regularization. In *NeurIPS*.

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. (2020). BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*.

Balcan, M.-F., Khodak, M., and Talwalkar, A. (2019). Provable guarantees for gradient-based meta-learning. In *International Conference on Machine Learning*, pages 424–433. PMLR.

Bartlett, P. L., Foster, D. J., and Telgarsky, M. J. (2017). Spectrally-normalized margin bounds for neural networks. *Advances in neural information processing systems*, 30.

Bartlett, P. L. and Mendelson, S. (2002). Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482.

Bechtle, S., Molchanov, A., Chebotar, Y., Grefenstette, E., Righetti, L., Sukhatme, G., and Meier, F. (2020). Meta-learning via learned loss. *ICPR*.

Beery, S., Van Horn, G., and Perona, P. (2018). Recognition in terra incognita. In *ECCV*.

Bello, I., Zoph, B., Vasudevan, V., and Le, Q. V. (2017). Neural optimizer search with reinforcement learning. In *International Conference on Machine Learning*, pages 459–468. PMLR.

Bengio, Y. (2000). Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2).

Blanchard, G., Deshmukh, A. A., Dogan, Ü., Lee, G., and Scott, C. (2021). Domain generalization by marginal transfer learning. *The Journal of Machine Learning Research*, 22(1):46–100.

Blum, A., Haghtalab, N., Procaccia, A. D., and Qiao, M. (2017). Collaborative pac learning. *Advances in Neural Information Processing Systems*, 30.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Carlucci, F. M., D'Innocente, A., Bucci, S., Caputo, B., and Tommasi, T. (2019). Domain generalization by solving jigsaw puzzles. In *CVPR*.

Cha, J., Chun, S., Lee, K., Cho, H.-C., Park, S., Lee, Y., and Park, S. (2021). Swad: Domain generalization by seeking flat minima. *NeurIPS*.

Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. (2019). Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018.

Chen, P., Ye, J., Chen, G., Zhao, J., and Heng, P.-A. (2021a). Robustness of accuracy metric and its inspirations in learning with noisy labels. In *AAAI*.

Chen, Q., Shui, C., and Marchand, M. (2021b). Generalization bounds for meta-learning: An information-theoretic analysis. *Advances in Neural Information Processing Systems*, 34:25878–25890.

Chen, T., Wu, W., Gao, Y., Dong, L., Luo, X., and Lin, L. (2018). Fine-grained representation learning and recognition by exploiting hierarchical semantic embedding. In *Proceedings of the 26th ACM International Conference on Multimedia*, pages 2023–2031.

Chen, X. and Gupta, A. (2015). Webly supervised learning of convolutional networks. In *ICCV*.

Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillicrap, T. P., Botvinick, M., and de Freitas, N. (2017). Learning to learn without gradient descent by gradient descent. In *ICML*.

Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., , and Vedaldi, A. (2014). Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., and Ha, D. (2018). Deep learning for classical japanese literature. In *NeurIPS (Workshop)*.

Coates, A., Ng, A., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(Jan):1–30.

Dou, Q., Castro, D. C., Kamnitsas, K., and Glocker, B. (2019). Domain generalization via model-agnostic learning of semantic features. In *NeurIPS*.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

Fang, C., Xu, Y., and Rockmore, D. N. (2013). Unbiased metric learning: On the utilization of multiple datasets and web images for softening bias. In *ICCV*.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

Finn, C. and Levine, S. (2018). Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. In *International Conference on Learning Representations*.

Flennerhag, S., Rusu, A. A., Pascanu, R., Visin, F., Yin, H., and Hadsell, R. (2020). Meta-learning with warped gradient descent. In *ICLR*.

Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. (2021). Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*.

Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. (2017). Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*.

Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., and Pontil, M. (2018). Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1568–1577. PMLR.

Fu, J., Luo, H., Feng, J., Low, K. H., and Chua, T.-S. (2016). Drmad: Distilling

reverse-mode automatic differentiation for optimizing hyperparameters of deep neural networks. In *IJCAI*.

Gao, B., Gouk, H., and Hospedales, T. M. (2021a). Searching for robustness: Loss learning for noisy classification tasks. In *ICCV*.

Gao, B., Gouk, H., and Hospedales, T. M. (2021b). Searching for robustness: Loss learning for noisy classification tasks. In *ICCV*.

Gao, B., Gouk, H., Lee, H. B., and Hospedales, T. M. (2022a). Meta mirror descent: Optimiser learning for fast convergence. *arXiv preprint arXiv:2203.02711*.

Gao, B., Gouk, H., Yang, Y., and Hospedales, T. (2022b). Loss function learning for domain generalization by implicit gradient. In *International Conference on Machine Learning*, pages 7002–7016. PMLR.

Ghifary, M., Kleijn, W. B., Zhang, M., and Balduzzi, D. (2015). Domain generalization for object recognition with multi-task autoencoders. In *ICCV*.

Ghosh, A., Kumar, H., and Sastry, P. (2017). Robust loss functions under label noise for deep neural networks. In *AAAI*.

Gonzalez, S. and Miikkulainen, R. (2020a). Improved training speed, accuracy, and data utilization through loss function optimization. In *CEC*.

Gonzalez, S. and Miikkulainen, R. (2020b). Improved training speed, accuracy, and data utilization via loss function optimization.

Gonzalez, S. and Miikkulainen, R. (2021). Optimizing loss functions through multivariate taylor polynomial parameterization. In *GECCO*.

Gouk, H., Hospedales, T. M., and Pontil, M. (2021). Distance-based regularisation of deep networks for fine-tuning. In *ICLR*.

Griffin, G., Holub, A., and Perona, P. (2007). Caltech-256 object category dataset.

Gulrajani, I. and Lopez-Paz, D. (2021). In search of lost domain generalization. In *ICLR*.

Han, B., Yao, Q., Yu, X., Niu, G., Xu, M., Hu, W., Tsang, I., and Sugiyama, M. (2018). Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *NeurIPS*.

Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *CEC*.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*.

Hensel, K. (1889). Ueber gattungen, welche durch composititon aus zwei anderen gattungen entstehen.

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. (2020). Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*.

Hospedales, T. M., Antoniou, A., Micaelli, P., and Storkey, A. J. (2021). Meta-Learning in Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Houthooft, R., Chen, Y., Isola, P., Stadie, B., Wolski, F., Ho, O. J., and Abbeel, P. (2018). Evolved policy gradients. In *NeurIPS*.

Hu, S., Zhang, K., Chen, Z., and Chan, L. (2020). Domain generalization via multidomain discriminant analysis. In *Uncertainty in Artificial Intelligence*, pages 292–302. PMLR.

Huang, C., Zhai, S., Talbott, W., Bautista, M. A., Sun, S.-Y., Guestrin, C., and Susskind, J. (2019). Addressing the loss-metric mismatch with adaptive loss alignment. In *ICML*.

Huang, Z., Wang, H., Xing, E. P., and Huang, D. (2020). Self-challenging improves cross-domain generalization. In *ECCV*.

Hull, J. J. (1994). A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554.

Hutter, F., Kotthoff, L., and Vanschoren, J., editors (2019). *Automatic machine learning: methods, systems, challenges*. Challenges in Machine Learning. Springer.

Jenni, S. and Favaro, P. (2018). Deep bilevel learning. In *ECCV*.

Jiang, L., Zhou, Z., Leung, T., Li, L.-J., and Fei-Fei, L. (2018). Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *ICML*.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. In *ICLR*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*.

Kirsch, L., van Steenkiste, S., and Schmidhuber, J. (2020). Improving generalization in meta reinforcement learning using learned objectives. In *ICLR*.

Kivinen, J. and Warmuth, M. K. (1997). Exponentiated gradient versus gradient descent for linear predictors. *information and computation*, 132(1):1–63.

Koh, P. W., Sagawa, S., Marklund, H., Xie, S. M., Zhang, M., Balsubramani, A., Hu, W., Yasunaga, M., Phillips, R. L., Gao, I., Lee, T., David, E., Stavness, I., Guo, W., Earnshaw, B., Haque, I., Beery, S. M., Leskovec, J., Kundaje, A., Pierson, E., Levine, S., Finn, C., and Liang, P. (2021). Wilds: A benchmark of in-the-wild distribution shifts. In *ICML*.

Kontorovich, A. (2014). Concentration in unbounded metric spaces and algorithmic stability. In *International Conference on Machine Learning*, pages 28–36. PMLR.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Master's thesis.

Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Kolesnikov, A., et al. (2020). The open images dataset v4. *International Journal of Computer Vision*, 128(7):1956–1981.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

Lee, H. B., Lee, H., Shin, J., Yang, E., Hospedales, T., and Hwang, S. J. (2021). Online hyperparameter meta-learning with hypergradient distillation. In *International Conference on Learning Representations*.

Lee, S., Lee, H. B., Lee, J., and Hwang, S. J. (2022). Sequential reptile: Inter-task gradient alignment for multilingual learning. In *International Conference on Learning Representations*.

Li, C., Yuan, X., Lin, C., Guo, M., Wu, W., Yan, J., and Ouyang, W. (2019a). Am-lfs: Automl for loss function search. In *ICCV*.

Li, D., Gouk, H., and Hospedales, T. (2022). Finding lost dg: Explaining domain generalization via model complexity. In *arXiv 2202.00563*.

Li, D., Yang, Y., Song, Y.-Z., and Hospedales, T. (2018a). Learning to generalize: Meta-learning for domain generalization. In *AAAI*.

Li, D., Zhang, J., Yang, Y., Liu, C., Song, Y.-Z., and Hospedales, T. M. (2019b). Episodic training for domain generalization. In *ICCV*.

Li, H., Pan, S. J., Wang, S., and Kot, A. C. (2018b). Domain generalization with adversarial feature learning. In *CVPR*.

Li, J., Wong, Y., Zhao, Q., and Kankanhalli, M. S. (2019c). Learning to learn from noisy labeled data. In *CVPR*.

Li, K. and Malik, J. (2017). Learning to optimize. In *ICLR*.

Li, Y., Tian, X., Gong, M., Liu, Y., Liu, T., Zhang, K., and Tao, D. (2018c). Deep domain generalization via conditional invariant adversarial networks. In *ECCV*.

Li, Y., Yang, Y., Zhou, W., and Hospedales, T. M. (2019d). Feature-critic networks for heterogeneous domain generalization. In *ICML*.

Li, Z., Zhou, F., Chen, F., and Li, H. (2017). Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In *ICCV*.

Liu, P., Zhang, G., Wang, B., Xu, H., Liang, X., Jiang, Y., and Li, Z. (2021). Loss function discovery for object detection via convergence-simulation driven search. In *ICLR*.

Long, P. M. and Sedghi, H. (2020). Generalization bounds for deep convolutional neural networks. In *ICLR*.

Lorraine, J., Vicol, P., and Duvenaud, D. (2020). Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pages 1540–1552. PMLR.

Ma, X., Huang, H., Wang, Y., Romano, S., Erfani, S., and Bailey, J. (2020). Normalized loss functions for deep learning with noisy labels. In *ICML*.

Maclaurin, D., Duvenaud, D., and Adams, R. P. (2015). Gradient-based hyperparameter optimization through reversible learning. In *ICML*.

Maji, S., Kannala, J., Rahtu, E., Blaschko, M., Vedaldi, A., Maji, S., Rahtu, E., Kannala, J., Blaschko, M., and Vedaldi, A. (2013). Fine-grained visual classification of aircraft. Technical report.

Martens, J. and Grosse, R. (2015). Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR.

Micaelli, P. and Storkey, A. J. (2021). Gradient-based hyperparameter optimization over long horizons. *Advances in Neural Information Processing Systems*, 34:10798–10809.

Muandet, K., Balduzzi, D., and Schölkopf, B. (2013). Domain generalization via invariant feature representation. In *International Conference on Machine Learning*, pages 10–18. PMLR.

Mukhoti, J., Kulharia, V., Sanyal, A., Golodetz, S., Torr, P. H., and Dokania, P. K. (2020). Calibrating deep neural networks using focal loss. *NeurIPS*.

Müller, R., Kornblith, S., and Hinton, G. E. (2019). When does label smoothing help? In *NeurIPS*.

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., and Sutskever, I. (2021). Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003.

Nam, H., Lee, H., Park, J., Yoon, W., and Yoo, D. (2021). Reducing domain gap via style-agnostic networks. In *CVPR*.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning.

Nichol, A., Achiam, J., and Schulman, J. (2018). On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

Nilsback, M.-E. and Zisserman, A. (2008). Automated flower classification over a large number of classes. In *Proceedings of the 6th Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE.

Park, E. and Oliva, J. B. (2019). Meta-curvature. In *NeurIPS*.

Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. (2012). Cats and dogs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Patrini, G., Rozza, A., Krishna Menon, A., Nock, R., and Qu, L. (2017). Making deep neural networks robust to label noise: A loss correction approach. In *CVPR*.

Pedregosa, F. (2016). Hyperparameter optimization with approximate gradient. In *International conference on machine learning*, pages 737–746. PMLR.

Pereyra, G., Tucker, G., Chorowski, J., Kaiser, Ł., and Hinton, G. (2017). Regularizing neural networks by penalizing confident output distributions. In *ICLR*.

Pinto, N., Stone, Z., Zickler, T., and Cox, D. (2011). Scaling up biologically-inspired computer vision: A case study in unconstrained face recognition on facebook. In *IEEE CVPR Workshops*, pages 35–42. IEEE.

Rajeswaran, A., Finn, C., Kakade, S. M., and Levine, S. (2019). Meta-learning with implicit gradients. *Advances in neural information processing systems*, 32.

Ramachandran, P., Zoph, B., and Le, Q. V. (2018). Searching for activation functions. In *ICLR (Workshop)*.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *AAAI*.

Reed, S., Lee, H., Anguelov, D., Szegedy, C., Erhan, D., and Rabinovich, A. (2015). Training deep neural networks on noisy labels with bootstrapping. In *ICLR*.

Ren, M., Zeng, W., Yang, B., and Urtasun, R. (2018). Learning to reweight examples for robust deep learning. In *ICML*.

Rosenfeld, E., Ravikumar, P., and Risteski, A. (2022). An online learning approach to interpolation and extrapolation in domain generalization. In *International Conference on Artificial Intelligence and Statistics*, pages 2641–2657. PMLR.

Ryu, J. U., Shin, J., Lee, H. B., and Hwang, S. J. (2020). Metaperturb: Transferable regularizer for heterogeneous tasks and architectures. *Advances in Neural Information Processing Systems*, 33:11501–11512.

Sandler, M., Vladymyrov, M., Zhmoginov, A., Miller, N., Madams, T., Jackson, A., and Arcas, B. A. Y. (2021). Meta-learning bidirectional update rules. In *ICML*.

Schmidhuber, J. (1987). *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München.

Schmidhuber, J., Zhao, J., and Wiering, M. (1997). Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. *Machine Learning*, 28(1):105–130.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Shalizi, C. and Kontorovich, A. (2013). Predictive pac learning and process decompositions. *Advances in neural information processing systems*, 26.

Shankar, S., Piratla, V., Chakrabarti, S., Chaudhuri, S., Jyothi, P., and Sarawagi, S. (2018). Generalizing across domains via cross-gradient training. In *ICLR*.

Shin, J., Lee, H. B., Gong, B., and Hwang, S. J. (2021). Large-scale meta-learning with continual trajectory shifting. *arXiv preprint arXiv:2102.07215*.

Shu, J., Xie, Q., Yi, L., Zhao, Q., Zhou, S., Xu, Z., and Meng, D. (2019). Meta-weight-net: Learning an explicit mapping for sample weighting. In *NeurIPS*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25.

Song, H., Kim, M., Park, D., and Lee, J.-G. (2020). Learning from noisy labels with deep neural networks: A survey. *arXiv preprint arXiv:2007.08199*.

Sun, B. and Saenko, K. (2016). Deep coral: Correlation alignment for deep domain adaptation. In *ECCV*.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *CVPR*.

Tan, M. and Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*.

Thrun, S. and Pratt, L. (1998). Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*.

Tripuraneni, N., Jin, C., and Jordan, M. (2021). Provable meta-learning of linear representations. In *International Conference on Machine Learning*, pages 10434–10443. PMLR.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142.

Venkateswara, H., Eusebio, J., Chakraborty, S., and Panchanathan, S. (2017). Deep hashing network for unsupervised domain adaptation. In *CVPR*.

Wang, X., Wang, S., Chi, C., Zhang, S., and Mei, T. (2020). Loss function search for face recognition. In *ICML*.

Wang, Y., Ma, X., Chen, Z., Luo, Y., Yi, J., and Bailey, J. (2019). Symmetric cross entropy for robust learning with noisy labels. In *CVPR*.

Wei, H., Feng, L., Chen, X., and An, B. (2020). Combating noisy labels by agreement: A joint training method with co-regularization. In *CVPR*.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Wichrowska, O., Maheswaranathan, N., Hoffman, M. W., Colmenarejo, S. G., Denil, M., de Freitas, N., and Sohl-Dickstein, J. (2017). Learned optimizers that scale and generalize. In *ICML*.

Wu, L., Tian, F., Xia, Y., Fan, Y., Qin, T., Jian-Huang, L., and Liu, T.-Y. (2018a). Learning to teach with dynamic loss functions. In *NeurIPS*.

Wu, Y., Ren, M., Liao, R., and Grosse., R. (2018b). Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations*.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Xiao, T., Xia, T., Yang, Y., Huang, C., and Wang, X. (2015). Learning from massive noisy labeled data for image classification. In *CVPR*.

Xu, Y., Cao, P., Kong, Y., and Wang, Y. (2019). L_dmi: A novel information-theoretic loss function for training deep nets robust to label noise. In *NeurIPS*.

Yadav, C. and Bottou, L. (2019). Cold case: The lost mnist digits. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc.

Yao, Q., Yang, H., Han, B., Niu, G., and Kwok, J. (2020). Searching to exploit memorization effect in learning with noisy labels. In *ICML*.

Yao, Z., Gholami, A., Shen, S., Mustafa, M., Keutzer, K., and Mahoney, M. (2021). Adahessian: An adaptive second order optimizer for machine learning. In *AAAI*.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. (2020). Gradient surgery for multi-task learning. In *NeurIPS*.

Zehfuss, G. (1858). Über eine gewisse determinante. *Zeitschrift für Mathematik und Physik*, 3(1858):298–301.

Zhang, M., Lucas, J., Ba, J., and Hinton, G. E. (2019). Lookahead optimizer: k steps forward, 1 step back. In *NeurIPS*.

Zhang, Y., Xiang, T., Hospedales, T. M., and Lu, H. (2018). Deep mutual learning. In *CVPR*.

Zhang, Z. and Sabuncu, M. (2018). Generalized cross entropy loss for training deep neural networks with noisy labels. In *NeurIPS*.

Zhao, S., Gong, M., Liu, T., Fu, H., and Tao, D. (2020). Domain generalization via entropy regularization. *NeurIPS*.

Zhou, K., Liu, Z., Qiao, Y., Xiang, T., and Loy, C. C. (2021a). Domain generalization: A survey. In *arXiv*.

Zhou, K., Yang, Y., Hospedales, T., and Xiang, T. (2020). Learning to generate novel domains for domain generalization. In *ECCV*.

Zhou, K., Yang, Y., Qiao, Y., and Xiang, T. (2021b). Domain generalization with mixstyle. In *ICLR*.

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. *ICLR*.