Meta Learning via Learned Loss

Sarah Bechtle[*1]
sbechtle@tuebingen.mpg.de

Artem Molchanov[*2]
molchano@usc.edu

Yevgen Chebotar[*2]
yevgen.chebotar@gmail.com

Edward Grefenstette[3]
egrefen@fb.com

Ludovic Righetti[1,4]
ludovic.righetti@nyu.edu

Gaurav Sukhatme[2]
gaurav@usc.edu

Franziska Meier[3]
fmeier@fb.com

Abstract—Typically, loss functions, regularization mechanisms and other important aspects of training parametric models are chosen heuristically from a limited set of options. In this paper, we take the first step towards automating this process, with the view of producing models which train faster and more robustly. Concretely, we present a meta-learning method for learning parametric loss functions that can generalize across different tasks and model architectures. We develop a pipeline for "meta-training" such loss functions, targeted at maximizing the performance of the model trained under them. The loss landscape produced by our learned losses significantly improves upon the original task-specific losses in both supervised and reinforcement learning tasks. Furthermore, we show that our meta-learning framework is flexible enough to incorporate additional information at *meta-train* time. This information shapes the learned loss function such that the environment does not need to provide this information during *meta-test* time. We make our code available at https://sites.google.com/view/mlthree

Index Terms—meta learning, reinforcement learning, optimization, deep learning

I. INTRODUCTION

Inspired by the remarkable capability of humans to quickly learn and adapt to new tasks, the concept of learning to learn, or *meta-learning*, recently became popular within the machine learning community [2, 5, 7]. We can classify *learning to learn* methods into roughly two categories: approaches that learn representations that can generalize and are easily adaptable to new tasks [7], and approaches that learn how to optimize models [2, 5].

In this paper we investigate the second type of approach. We propose a learning framework that is able to learn any parametric loss function—as long as its output is differentiable with respect to its parameters. Such learned functions can be used to efficiently optimize models for new tasks.

Specifically, the purpose of this work is to encode learning strategies into a parametric loss function, or a *meta-loss*, which
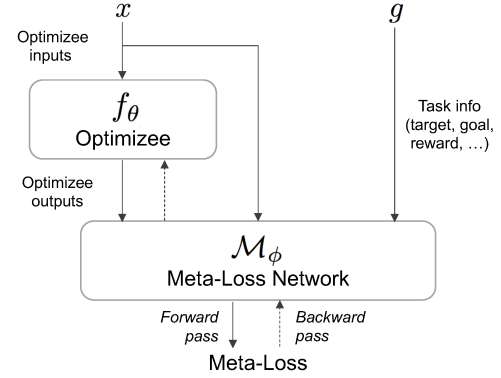
[*]Equal contributions.

[1]The authors are with the Max Planck Institute for Intelligent Systems, Tübingen, Germany

[2]The authors are with the Viterbi School of Engineering, University of Southern California, Los Angeles, CA 90089

[3]The authors are with Facebook AI Research

[4]The authors are with the Tandon School of Engineering, New York University, Brooklyn, NY 11201

Fig. 1: Framework overview: The learned meta-loss is used as a learning signal to optimize the optimizee f_θ, which can be a regressor, a classifier or a control policy.

generalizes across multiple training contexts or tasks. Inspired by *inverse reinforcement learning* [21], our work combines the *learning to learn* paradigm of meta-learning with the generality of learning loss landscapes. We construct a unified, fully differentiable framework that can learn optimizee-independent loss functions to provide a strong learning signal for a variety of learning problems, such as classification, regression or reinforcement learning. Our framework involves an inner and an outer optimization loops. In the inner loop, a model or an *optimizee* is trained with gradient descent using the loss coming from our learned meta-loss function. Fig. 1 shows the pipeline for updating the optimizee with the meta-loss. The outer loop optimizes the meta-loss function by minimizing a *task-loss*, such as a standard regression or reinforcement-learning loss, that is induced by the updated optimizee.

The contributions of this work are as follows: i) we present a framework for learning adaptive, high-dimensional loss functions through back-propagation that create the loss landscapes for efficient optimization with gradient descent. We show that our learned meta-loss functions improve over directly learning via the task-loss itself while maintaining the generality of the task-loss. ii) We present several ways our framework can incorporate extra information that helps shape the loss landscapes at *meta-train* time. This extra information can take on various forms, such as exploratory signals or

expert demonstrations for RL tasks. After training the meta-loss function, the task-specific losses are no longer required since the training of optimizees can be performed entirely by using the meta-loss function alone, without requiring the extra information given at *meta-train* time. In this way, our meta-loss can find more efficient ways to optimize the original task loss.

We apply our meta-learning approach to a diverse set of problems demonstrating our framework's flexibility and generality. The problems include regression problems, image classification, behavior cloning, model-based and model-free reinforcement learning. Our experiments include empirical evaluation for each of the aforementioned problems.

II. RELATED WORK

Meta-learning originates from the concept of learning to learn [25, 3, 30]. Recently, there has been a wide interest in finding ways to improve learning speeds and generalization to new tasks through meta-learning. Let us consider gradient based learning approaches, that update the parameters of an *optimizee* $f_\theta(x)$, with model parameters θ and inputs x as follows:

$$\theta_{\text{new}} = h_\psi(\theta, \nabla_\theta \mathcal{L}_\phi(y, f_\theta(x)); \quad (1)$$

where we take the gradient of a loss function $\mathcal{L}$, parametrized by ϕ, with respect to the optimizee's parameters θ and use a gradient transform h, parametrized by ψ, to compute new model parameters θ_{new}[1]. In this context, we can divide related work on meta-learning into learning model parameters θ that can be easily adapted to new tasks [7, 18, 10, 35], learning optimizer policies h that transform parameters updates with respect to known loss or reward functions [16, 2, 14, 8, 17, 5], or learning loss/reward function representations ϕ [27, 11, 36]. Alternatively, in unsupervised learning settings, meta-learning has been used to learn unsupervised rules that can be transferred between tasks [19, 12].

Our framework falls into the category of learning loss landscapes. Similar to works by Sung et al. [27] and Houthooft et al. [11], we aim at learning loss function parameters ϕ that can be applied to various optimizee models, e.g. regressors, classifiers or agent policies. Our learned loss functions are independent of the model parameters θ that are to be optimized, thus they can be easily transferred to other optimizee models. This is in contrast to methods that meta-learn model-parameters θ directly [e.g. 7, 18], which are orthogonal and complementary to ours, where the learned representation θ cannot be separated from the original model of the optimizee. The idea of learning loss landscapes or reward functions in the reinforcement learning (RL) setting can be traced back to the field of inverse reinforcement learning [21, 1, IRL]. However, in contrast to IRL we do not require expert demonstrations (however we can incorporate them). Instead we use task losses as a measure of the effectiveness of our loss function when using it to update an optimizee.

[1]For simple gradient descent:
$h(\theta, \nabla_\theta \mathcal{L}(y, f_\theta(x)) = \theta - \psi \nabla_\theta \mathcal{L}(y, f_\theta(x))$

Closest to our method are the works on *evolved policy gradients* [11], *teacher networks* [32], *meta-critics* [27] and *meta-gradient RL* [33]. In contrast to using an evolutionary approach [e.g. 11], we design a differentiable framework and describe a way to optimize the loss function with gradient descent in both supervised and reinforcement learning settings. Wu et al. [32] propose that instead of learning a differentiable loss function directly, a teacher network is trained to predict parameters of a manually designed loss function, whereas each new loss function class requires a new teacher network design and training. In Xu et al. [33], discount and bootstrapping parameters are learned online to optimize a task-specific meta-objective. Our method does not require manual design of the loss function parameterization or choosing particular parameters that have to be optimized, as our loss functions are learned entirely from data. Finally, in work by Sung et al. [27] a *meta-critic* is learned to provide a task-conditional value function, used to train an actor policy. Although training a meta-critic in the supervised setting reduces to learning a loss function as in our work, in the reinforcement learning setting we show that it is possible to use learned loss functions to optimize policies directly with gradient descent.

III. META-LEARNING VIA LEARNED LOSS

In this work, we aim to learn a loss function, which we call *meta-loss*, that is subsequently used to train an *optimizee*, e.g. a classifier, a regressor or a control policy. More concretely, we aim to learn a meta-loss function $\mathcal{M}_\phi$ with parameters ϕ, that outputs the loss value $\mathcal{L}_{\text{learned}}$ which is used to train an optimizee f_θ with parameters θ via gradient descent:

$$\theta_{\text{new}} = \theta - \alpha \nabla_\theta \mathcal{L}_{\text{learned}}, \quad (2)$$
$$\text{where } \mathcal{L}_{\text{learned}} = \mathcal{M}_\phi(y, f_\theta(x)) \quad (3)$$

where y can be ground truth target information in supervised learning settings or goal and state information for reinforcement learning settings. In short, we aim to learn a loss function that can be used as depicted in Algorithm 2. Towards this goal, we propose an algorithm to learn meta-loss function parameters ϕ via gradient descent.

The key challenge is to derive a training signal for learning the loss parameters ϕ. In the following, we describe our approach to addressing this challenge, which we call **M**eta-**L**earning via **L**earned **L**oss (ML[3]).

A. ML[3] for Supervised Learning

We start with supervised learning settings, in which our framework aims at learning a meta-loss function $\mathcal{M}_\phi(y, f_\theta(x))$ that produces the loss value given the ground truth target y and the predicted target $f_\theta(x)$. For clarity purposes we constrain the following presentation to learning a meta-loss network that produces the loss value for training a regressor f_θ via gradient descent, however the methodology trivially generalizes to classification tasks.

Our meta-learning framework starts with randomly initialized model parameters θ and loss parameters ϕ. The current loss parameters are then used to produce loss value $\mathcal{L}_{\text{learned}} =$

$\mathcal{M}_\phi(y, f_\theta(x))$. To optimize model parameters θ we need to compute the gradient of the loss value with respect to θ, $\nabla_\theta \mathcal{L} = \nabla_\theta \mathcal{M}_\phi(y, f_\theta(x))$. Using the chain rule, we can decompose the gradient computation into the gradient of the loss network with respect to predictions of model $f_\theta(x)$ times the gradient of model f with respect to model parameters[2],

$$\nabla_\theta \mathcal{M}_\phi(y, f_\theta(x)) = \nabla_f \mathcal{M}_\phi(y, f_\theta(x)) \nabla_\theta f_\theta(x). \quad (4)$$

Once we have updated the model parameters $\theta_{\text{new}} = \theta - \alpha \nabla_\theta \mathcal{L}_{\text{learned}}$ using the current meta-loss network parameters ϕ, we want to measure how much learning progress has been made with loss-parameters ϕ and optimize ϕ via gradient descent. Note, that the new model parameters θ_{new} are implicitly a function of loss-parameters ϕ, because changing ϕ would lead to different θ_{new}. In order to evaluate θ_{new}, and through that loss-parameters ϕ, we introduce the notion of a *task-loss* during *meta-train* time. For instance, we use the mean-squared-error (MSE) loss, which is typically used for regression tasks, as a task-loss $\mathcal{L}_\mathcal{T} = (y - f_{\theta_{\text{new}}}(x))^2$. We now optimize loss parameters ϕ by taking the gradient of $\mathcal{L}_\mathcal{T}$ with respect to ϕ as follows[2]:

$$\nabla_\phi \mathcal{L}_\mathcal{T} = \nabla_f \mathcal{L}_\mathcal{T} \nabla_{\theta_{\text{new}}} f_{\theta_{\text{new}}} \nabla_\phi \theta_{\text{new}} \quad (5)$$

$$= \nabla_f \mathcal{L}_\mathcal{T} \nabla_{\theta_{\text{new}}} f_{\theta_{\text{new}}} \nabla_\phi [\theta - \alpha \nabla_\theta \mathbb{E} \left[\mathcal{M}_\phi(y, f_\theta(x)) \right] \quad (6)$$

where we first apply the chain rule and show that the gradient with respect to the meta-loss parameters ϕ requires the new model parameters θ_{new}. We expand θ_{new} as one gradient step on θ based on meta-loss $\mathcal{M}_\phi$, making the dependence on ϕ explicit.

Optimization of the loss-parameters can either happen after *each inner gradient* step (where inner refers to using the current loss parameters to update θ), or after M *inner gradient steps* with the current meta-loss network $\mathcal{M}_\phi$.

The latter option requires back-propagation through a chain of all optimizee update steps. In practice we notice that updating the meta-parameters ϕ after each inner gradient update step works better. We reset θ after M *inner gradient steps*. We summarize the *meta-train* phase in Algorithm 1, with one *inner* gradient step.

B. ML³ Reinforcement Learning

In this section, we introduce several modifications that allow us to apply the ML³ framework to reinforcement learning problems. Let $\mathcal{M} = (S, A, P, R, p_0, \gamma, T)$ be a finite-horizon Markov Decision Process (MDP), where S and A are state and action spaces, $P : S \times A \times S \to \mathbb{R}_+$ is a state-transition probability function or system dynamics, $R : S \times A \to \mathbb{R}$ a reward function, $p_0 : S \to \mathbb{R}_+$ an initial state distribution, γ a reward discount factor, and T a horizon. Let $\tau = (s_0, a_0, \ldots, s_T, a_T)$ be a trajectory of states and actions and $R(\tau) = \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$ the trajectory return. The goal of reinforcement learning is to find parameters θ of a policy $\pi_\theta(a|s)$ that maximizes the expected discounted reward over trajectories induced by the policy: $\mathbb{E}_{\pi_\theta}[R(\tau)]$ where

---

[2]Alternatively this gradient computation can be performed using automatic differentiation

---

Algorithm 1 ML³ at (*meta-train*)

1: $\phi \leftarrow$ randomly initialize
2: **while** not done **do**
3:      $\theta \leftarrow$ randomly initialize
4:      $x, y \leftarrow$ Sample task samples from $\mathcal{T}$
5:      $\mathcal{L}_{\text{learned}} = \mathcal{M}(y, f_\theta(x))$
6:      $\theta_{\text{new}} \leftarrow \theta - \alpha \nabla_\theta \mathbb{E}_x \left[\mathcal{L}_{\text{learned}} \right]$
7:      $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_\mathcal{T}(y, f_{\theta_{\text{new}}})$
8: **end while**

Algorithm 2 ML³ at (*meta-test*)

1: $M \leftarrow$ # of optimizee updates
2: $\theta \leftarrow$ randomly initialize
3: **for** $j \in \{0, \ldots, M\}$ **do**
4:      $x, y \leftarrow$ Sample task samples from $\mathcal{T}$
5:      $\mathcal{L}_{\text{learned}} = \mathcal{M}(y, f_\theta(x))$
6:      $\theta \leftarrow \theta - \alpha \nabla_\theta \mathbb{E}_x \left[\mathcal{L}_{\text{learned}} \right]$
7: **end for**

$s_0 \sim p_0, s_{t+1} \sim P(s_{t+1}|s_t, a_t)$ and $a_t \sim \pi_\theta(a_t|s_t)$. In what follows, we show how to train a meta-loss network to perform effective policy updates in a reinforcement learning scenario. To apply our ML³ framework, we replace the optimizee f_θ from the previous section with a stochastic policy $\pi_\theta(a|s)$. We present two applications of ML³ to RL.

1) ML³ for Model-Based Reinforcement Learning: Model-based RL (MBRL) attempts to learn a policy π_θ by first learning a dynamic model P. Intuitively, if the model P is accurate, we can use it to optimize the policy parameters θ. As we typically do not know the dynamics model a-priori, MBRL algorithms iterate between using the current approximate dynamics model P, to optimize the policy π_θ such that it maximizes the reward R under P, then use the optimized policy π_θ to collect more data which is used to update the model P. In this context, we aim to learn a loss function that is used to optimize policy parameters through our meta-network $\mathcal{M}$.

Similar to the supervised learning setting we use current meta-parameters ϕ to optimize policy parameters θ under the current dynamics model P: $\theta_{\text{new}} = \theta - \alpha \nabla_\theta \left[\mathcal{M}_\phi(\tau, g) \right]$,

where $\tau = (s_0, a_0, \ldots, s_T, a_T)$ is the sampled trajectory and the variable g captures some task-specific information, such as the goal state of the agent. To optimize ϕ we again need to define a task loss, which in the MBRL setting can be defined as $\mathcal{L}_\mathcal{T}(g, \pi_{\theta_{\text{new}}}) = -\mathbb{E}_{\pi_{\theta_{\text{new}}}, P}[R_g(\tau_{\text{new}})]$, denoting the reward that is achieved under the current dynamics model P. To update ϕ, we compute the gradient of the task loss $\mathcal{L}_\mathcal{T}$ wrt. ϕ, which involves differentiating all the way through the reward function, dynamics model and the policy that was updated using the meta-loss $\mathcal{M}_\phi$. The pseudo-code in Algorithm 3 (Appendix A) illustrates the MBRL learning loop. In Algorithm 5 (Appendix A), we show the policy optimization procedure during meta-test time. Notably, we have found that in practice, the model of the dynamics P is not needed anymore for policy optimization at meta-test time. The meta-network learns to implicitly represent the gradients of the dynamics model and can produce a loss to optimize the policy directly.

2) ML³ for Model-Free Reinforcement Learning: Finally, we consider the model-free reinforcement learning (MFRL)

case, where we learn a policy without learning a dynamics model. In this case, we can define a surrogate objective, which is independent of the dynamics model, as our task-specific loss [31, 28, 26]:

$$\mathcal{L}_{\mathcal{T}}(g, \pi_{\theta_{\text{new}}}) = -\mathbb{E}_{\pi_{\theta_{\text{new}}}}\left[R_g(\tau_{\text{new}}) \log \pi_{\theta_{\text{new}}}(\tau_{\text{new}})\right] \quad (7)$$

$$= -\mathbb{E}_{\pi_{\theta_{\text{new}}}}\left[R_g(\tau_{\text{new}}) \sum_{t=0}^{T-1} \log \pi_{\theta_{\text{new}}}(a_t|s_t)\right] \quad (8)$$

Similar to the MBRL case, the task loss is indirectly a function of the meta-parameters ϕ that are used to update the policy parameters. Although we are evaluating the task loss on full trajectory rewards, we perform policy updates from Eq. 2 using stochastic gradient descent (SGD) on the meta-loss with mini-batches of experience (s_i, a_i, r_i) for $i \in \{0, \ldots, B-1\}$ with batch size B, similar to Houthooft et al. [11]. The inputs of the meta-loss network are the sampled states, sampled actions, task information g and policy probabilities of the sampled actions: $\mathcal{M}_\phi(s, a, \pi_\theta(a|s), g)$. In this way, we enable efficient optimization of very high-dimensional policies with SGD provided only with trajectory-based rewards. In contrast to the above MBRL setting, the rollouts used for task-loss evaluation are real system rollouts, instead of simulated rollouts. At test time, we use the same policy update procedure as in the MBRL setting, see Algorithm 5 (Appendix A).

C. Shaping ML³ loss by adding extra loss information during meta-train

So far, we have discussed using standard task losses, such as MSE-loss for regression or reward functions for RL settings. However, it is possible to provide more information about the task at *meta-train* time, which can influence the learning of the loss-landscape. We can design our task-losses to incorporate extra penalties; for instance we can extend the MSE-loss with $\mathcal{L}_{\text{extra}}$ and weight the terms with β and γ:

$$\mathcal{L}_{\mathcal{T}} = \beta(y - f_\theta(x))^2 + \gamma\mathcal{L}_{\text{extra}} \quad (9)$$

In our work, we experiment with 4 different types of extra loss information at *meta-train* time: for supervised learning we show that adding extra information through $\mathcal{L}_{\text{extra}} = (\theta - \theta^*)^2$, where θ^* are the optimal regression parameters, can help shape a convex loss-landscape for otherwise non-convex optimization problems; we also show how we can use $\mathcal{L}_{\text{extra}}$ to induce a physics prior in robot model learning. For reinforcement learning tasks we demonstrate that by providing additional rewards in the task loss during meta-train time, we can encourage the trained meta-loss to learn exploratory behaviors; and finally also for reinforcement learning tasks, we show how expert demonstrations can be incorporated to learn loss functions which can generalize to new tasks. In all settings, the additional information shapes the learned loss function such that the environment does not need to provide this information during meta-test time.

IV. EXPERIMENTS

In this section we evaluate the applicability and the benefits of the learned meta-loss from two different view points. First, we study the benefits of using standard task losses, such as the mean-squared error loss for regression, to train the meta-loss in Section IV-A. We analyze how a learned meta-loss compares to using a standard task-loss in terms of generalization properties and convergence speed. Second, we study the benefit of adding extra information at *meta-train* time to shape the loss landscape in Section IV-B.

A. Learning to mimic and improve over known task losses

First, we analyze how well our meta-learning framework can learn to mimic and improve over standard task losses for both supervised and reinforcement learning settings. For these experiments, the meta-network is parameterized by a neural network with two hidden layers of 40 neurons each.

1) Meta-Loss for Supervised Learning: In this set of experiments, we evaluate how well our meta-learning framework can learn loss functions $\mathcal{M}_\phi$ for regression and classification tasks. In particular, we perform experiments on sine function regression and binary classification of digits (see details in Appendix A). At meta-train time, we randomly draw one task for meta-training (see Fig. 2 (a)), and at meta-test time we randomly draw 10 test tasks for regression, and 4 test tasks for classification (Fig. 2(b)). For the sine regression, tasks are drawn according to details in Appendix A, and we initialize our model f_θ to a simple feedforward NN with 2 hidden layers and 40 hidden units each, for the binary classification task f_θ is initialized via the *LeNet* architecture [13]. For both experiments we use a fixed learning rate $\alpha = \eta = 0.001$ for both inner (α) and outer (η) gradient update steps. We average results across 5 random seeds, where each seed controls the initialization of both initial model and meta-network parameters, as well as the the random choice of meta-train/test task(s), and visualize them in Fig. 2. We compare the performance of using SGD with the task-loss $\mathcal{L}$ directly (in orange) to SGD using the learned meta-network $\mathcal{M}_\phi$ (in blue), both using a learning rate $\alpha = 0.001$. In Fig. 2 (c) we show the average performance of the meta-network $\mathcal{M}_\phi$ as it is being learned, as a function of (outer) meta-train iterations in blue. In both regression and classification tasks, the meta-loss eventually leads to a better performance on the meta-train task as compared to the task loss. In Fig. 2 (d) we evaluate SGD using $\mathcal{M}_\phi$ vs SGD using $\mathcal{L}$ on previously unseen (and out-of-distribution) meta-test tasks as a function of the number of gradient steps. Even on these novel test tasks, our learned $\mathcal{M}_\phi$ leads to improved performance as compared to the task-loss.

2) Learning Reward functions for Model-based Reinforcement Learning: In the MBRL example, the tasks consist of a free movement task of a point mass in a 2D space, we call this environment PointmassGoal, and a reaching task with a 2-link 2D manipulator, which we call the ReacherGoal environment (see Appendix A for details). The task distribution $p(\mathcal{T})$ consists of different target positions that either the point mass or the arm should reach. During meta-train time, a model of the system dynamics, represented by a neural network, is learned from samples of the currently optimal policy. The task loss during meta-train time is $\mathcal{L}_{\mathcal{T}}(\theta) = \mathbb{E}_{\pi_\theta, P}[R(\tau)]$, where $R(\tau)$

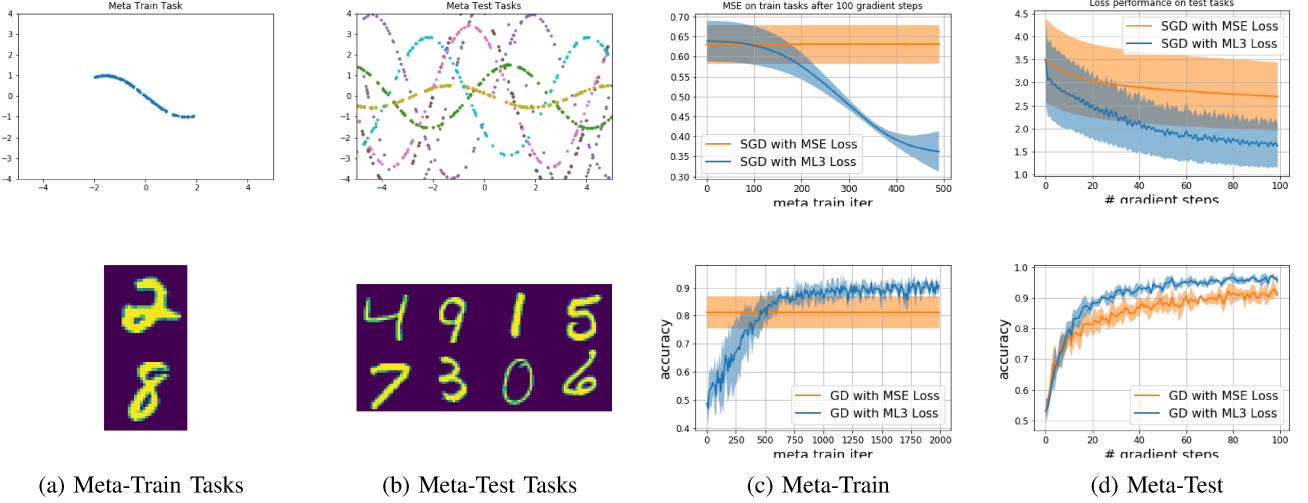| (a) Meta-Train Tasks | (b) Meta-Test Tasks | (c) Meta-Train | (d) Meta-Test |

Fig. 2: Meta-learning for regression (top) and binary classification (bottom) tasks. (a) meta-train task, (b) meta-test tasks, (c) performance of the meta-network on the meta-train task as a function of (outer) meta-train iterations in blue, as compared to SGD using the task-loss directly in orange, (d) average performance of meta-loss on meta-test tasks as a function of the number of gradient update steps
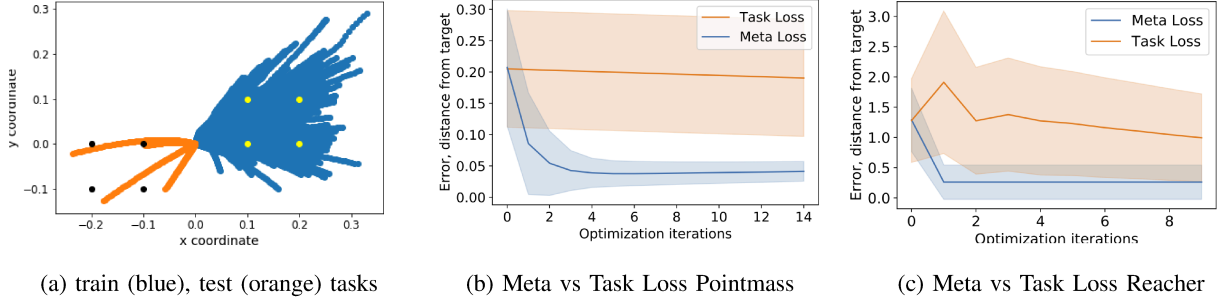

| (a) train (blue), test (orange) tasks | (b) Meta vs Task Loss Pointmass | (c) Meta vs Task Loss Reacher |

Fig. 3: ML^3 for MBRL: results are averaged across 10 runs. We can see in (a) that the ML^3 loss generalizes well, the loss was trained on the blue trajectories and tested on the orange ones for the PointmassGoal task. ML^3 loss also significantly speeds up learning when compared to the task loss at meta-test time on the PointmassGoal (b) and the ReacherGoal (c) environments.

is the final distance from the goal g, when rolling out $\pi_{\theta_{\text{new}}}$ in the dynamics model P. Taking the gradient $\nabla_\phi \mathbb{E}_{\pi_{\theta_{\text{new}}, P}}[R(\tau)]$ requires the differentiation through the learned model P (see Appendix 3). The input to the meta-network is the state-action trajectory of the current roll-out and the desired target position. The meta-network outputs a loss signal together with the learning rate to optimize the policy. Fig. 3a shows the qualitative reaching performance of a policy optimized with the meta loss during meta-test on PointmassGoal. The meta-loss network was trained only on tasks in the right quadrant (blue trajectories) and tested on the tasks in the left quadrant (orange trajectories) of the x, y plane, showing the generalization capability of the meta loss. Figure 3b and 3c show a comparison in terms of final distance to the target position at test time. The performance of policies trained with the meta-loss is compared to policies trained with the task loss, in this case final distance to the target. The curves show results for 10 different goal positions (including goal positions where the meta-loss needs to generalize). When optimizing with the task loss, we use the dynamics model learned during the meta-train time, as in this case the differentiation through the model is required during

test time. As mentioned in Section III-B1, this is not needed when using the meta-loss.

3) Learning Reward functions for Model-free Reinforcement Learning: In the following, we move to evaluating on model-free RL tasks. Fig. 4 shows results when using two continuous control tasks based on OpenAI Gym MuJoCo environments [22]: ReacherGoal and AntGoal (see Appendix A for details)[3] Fig. 4a and Fig. 4b show the results of the meta-test time performance for the ReacherGoal and the AntGoal environments respectively. We can see that ML^3 loss significantly improves optimization speed in both scenarios compared to PPO. In our experiments, we observed that on average ML^3 requires 5 times fewer samples to reach 80% of task performance in terms of our metrics for the model-free tasks.

To test the capability of the meta-loss to generalize across different architectures, we first meta-train $\mathcal{M}_\phi$ on an architecture with two layers and meta-test the same meta-loss

---

[3]Our framework is implemented using open-source libraries *Higher* [9] for convenient second-order derivative computations and *Hydra* [34] for simplified handling of experiment configurations.
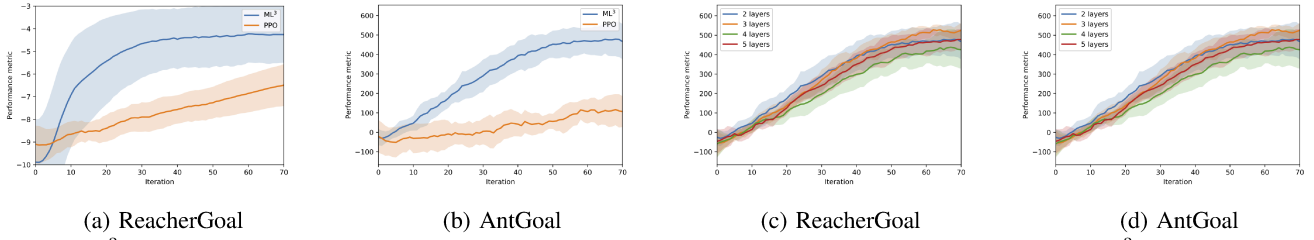
| (a) ReacherGoal | (b) AntGoal | (c) ReacherGoal | (d) AntGoal |

Fig. 4: ML^3 for model-free RL: results are averaged across 10 tasks. (a+b) Policy learning on new task with ML^3 loss compared to PPO objective performance during *meta-test* time. The learned loss leads to faster learning at meta-test time. (c+d) Using the same ML^3 loss, we can optimize policies of different architectures, showing that our learned loss maintains generality.

on architectures with varied number of layers. Fig. 4 (c+d) show meta-test time comparison for the ReacherGoal and the AntGoal environments in a model-free setting for four different model architectures. Each curve shows the average and the standard deviation over ten different tasks in each environment. Our comparison clearly indicates that the meta-loss can be effectively re-used across multiple architectures with a mild variation in performance compare to the overall variance of the corresponding task optimization.

B. Shaping loss landscapes by adding extra information at meta-train time

This set of experiments shows that our meta-learner is able to learn loss functions that incorporate extra information available only during meta-train time. The learned loss will be shaped such that optimization is faster when using the meta-loss compared to using a standard loss.

1) Illustration: Shaping loss: We start by illustrating the loss shaping on an example of sine frequency regression where we fit a single parameter for the purpose of visualization simplicity.

For this illustration we generate training data $\mathcal{D} = \{x_n, y_n\}^N, N = 1000$, by drawing data samples from the ground truth function $y = sin(\nu x)$, for $x = [-1, 1]$. We create a model $f_\omega(x) = \sin(\omega x)$, and aim to optimize parameter ω on $\mathcal{D}$, with the goal of recovering value ν. Fig. 5a (bottom) shows the loss landscape for optimizing ω, when using the MSE loss. The target frequency ν is indicated by a vertical red line. As noted by Parascandolo et al. [23], the landscape of this loss is highly non-convex and difficult to optimize with conventional gradient descent.

Here, we show that by utilizing additional information about the ground truth value of the frequency at meta-train time, we can learn a better shaped loss. Specifically, during meta-train time, our task-specific loss is the squared distance to the ground truth frequency: $(\omega - \nu)^2$ that we later call the *shaping loss*. The inputs of the meta-network $\mathcal{M}_\phi(y, \hat{y})$ are the training targets y and predicted function values $\hat{y} = f_\omega(x)$, similar to the inputs to the mean-squared loss. After meta-train time commences our learned loss function $\mathcal{M}_\phi$ produces a convex loss landscapes as depicted in Fig. 5a(top).

To analyze how the shaping loss impacts model optimization at meta-test time, we compare 3 loss functions: 1) directly using standard MSE loss (orange), 2) ML^3 loss that was trained via the MSE loss as task loss (blue), and 3) ML^3 loss trained via

the shaping loss, Fig. 5b. When comparing the performance of these 3 losses, it becomes evident that without shaping the loss landscape, the optimization is prone to getting stuck in a local optimum.

2) Shaping loss via physics prior for inverse dynamics learning: Next, we show the benefits of shaping our ML^3 loss via ground truth parameter information for a robotics application. Specifically, we aim to learn and shape a meta-loss that improves sample efficiency for learning (inverse) dynamics models, i.e. a mapping $u = f(q, \dot{q}, \ddot{q}_{\text{des}})$, where: q, $\dot{q}$, $\ddot{q}_{\text{des}}$ are vectors of joint angular positions, velocities and desired accelerations; u is a vector of joint torques.

Rigid body dynamics (RBD) provides an analytical solution to computing the (inverse) dynamics and can generally be written as:

$$M(q)\ddot{q} + F(q, \dot{q}) = u \qquad (10)$$

where the inertia matrix $M(q)$, and $F(q, \dot{q})$ are computed analytically [6]. Learning an inverse dynamics model using neural networks can increase the expressiveness compared to RBD but requires many data samples that are expensive to collect. Here we follow the approach in [15], and attempt to learn the inverse dynamics via a neural network that predicts the inertia matrix $M_\theta(q)$. To improve upon sample efficiency we apply our method by shaping the loss landscape during meta-train time using the ground truth inertia matrix $M(q)$ provided by a simulator. Specifically, we use the task loss $\mathcal{L}_\mathcal{T} = (M_\theta(q) - M(q))^2$ to optimize our meta-loss network. During meta-test time we use our trained meta-loss shaped with the physics prior (the inertia matrix exposed by the simulator) to optimize the inverse dynamics neural network. In Fig. 5-c we show the prediction performance of the inverse dynamics model during meta-test time on new trajectories of the ReacherGoal environment. We compare the optimization performance during meta-test time when using the meta-loss trained with physics prior, the meta loss trained without physics prior (i.e via MSE loss) to the optimization with MSE loss. Fig. 5-d shows a similar comparison for the Sawyer environment - a simulator of the 7 degrees-of-freedom Sawyer anthropomorphic robot arm. Inverse dynamics learning using the meta loss with physics prior achieves the best prediction performance on both robots. ML^3 without physics prior performs worst on the ReacherGoal environment, in this case the task loss formulated only in the action space did not provide enough information to learn
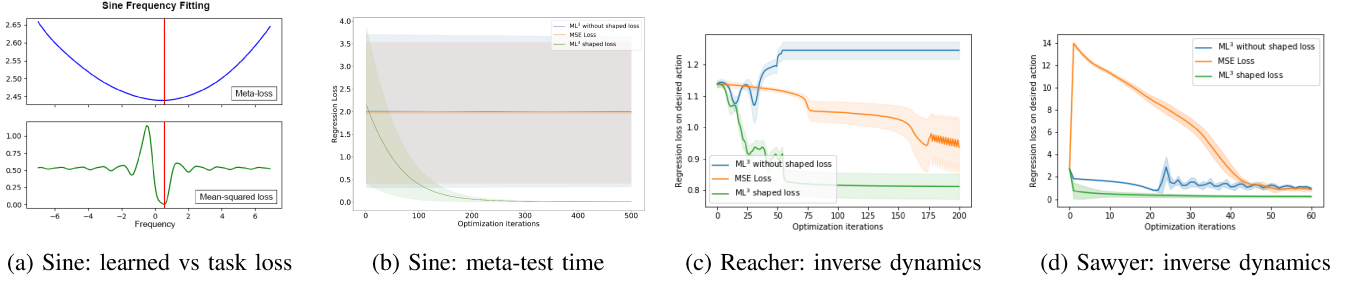
(a) Sine: learned vs task loss   (b) Sine: meta-test time   (c) Reacher: inverse dynamics   (d) Sawyer: inverse dynamics

Fig. 5: Meta-test time evaluation of the shaped meta-loss (ML^3), i.e. trained with shaping ground-truth (extra) information at meta-train time: a) Comparison of learned ML^3 loss (top) and MSE loss (bottom) landscapes for fitting the frequency of a sine function. The red lines indicate the ground-truth values of the frequency. b) Comparing optimization performance of: ML^3 loss trained with (green), and without (blue) ground-truth frequency values; MSE loss (orange). The ML^3 loss learned with the ground-truth values outperforms both the non-shaped ML^3 loss and the MSE loss. c-d) Comparing performance of inverse dynamics model learning for ReacherGoal (c) and Sawyer arm (d). ML^3 loss trained with (green) and without (blue) ground-truth inertia matrix is compared to MSE loss (orange). The shaped ML^3 loss outperforms the MSE loss in all cases.

(a) Trajectory ML^3 vs. iLQR   (b) MountainCar: meta-test time   (c) Train and test targets   (d) ML^3 vs. Task loss at test
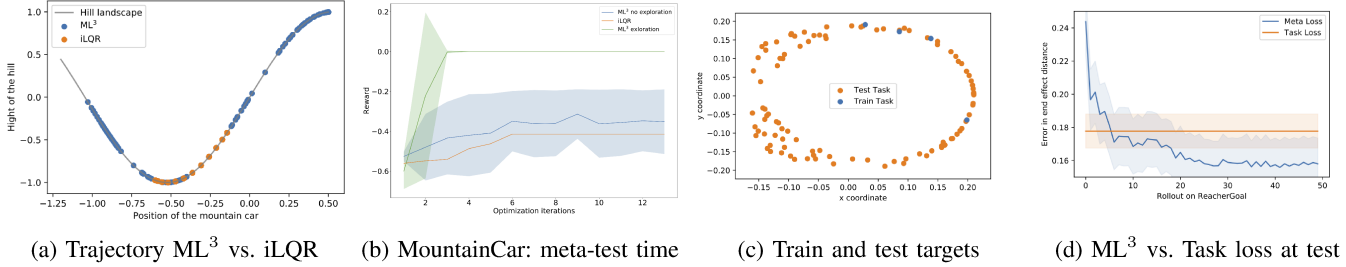
Fig. 6: (a) MountainCar trajectory for policy optimized with iLQR compared to ML^3 loss with extra information. (b) optimization performance during meta-test time for policies optimized with iLQR compared to ML^3 with and without extra information. (c+d) ReacherGoal with expert demonstrations available during meta-train time. (c) shows the targets in end-effector space. The four blue dots show the training targets for which expert demonstrations are available, the orange dots show the meta-test targets. In (d) we show the reaching performance of a policy trained with the shaped ML^3 loss at meta-test time, compared to the performance of training simply on the behavioral cloning objective and testing on test targets.

a $\mathcal{L}_{\text{learned}}$ useful for optimization. For the Sawyer training with MSE loss leads to a slower optimization, however the asymptotic performance of MSE and ML^3 is the same. Only ML^3 with shaped loss outperforms both.

3) Shaping Loss via intermediate goal states for RL: We analyze loss landscape shaping on the MountainCar environment [20], a classical control problem where an under-actuated car has to drive up a steep hill. The propulsion force generated by the car does not allow steady climbing of the hill, thus greedy minimization of the distance to the goal often results in a failure to solve the task. The state space is two-dimensional consisting of the position and velocity of the car, the action space consists of a one-dimensional torque. In our experiments, we provide intermediate goal positions during meta-train time, which are not available during the meta-test time. The meta-network incorporates this behavior into its loss leading to an improved exploration during the meta-test time as can be seen in Fig. 6-a, when compared to a classical iLQR-based trajectory optimization [29]. Fig. 6-b shows the average distance between the car and the goal at last rollout time step over several iterations of policy updates with ML^3 with and without extra information and iLQR. As we observe, ML^3 with extra information can successfully bring the car to the goal in

a small amount of updates, whereas iLQR and ML^3 without extra information is not able to solve this task.

4) Shaping loss via expert information during meta-train time: Expert information, like demonstrations for a task, is another way of adding relevant information during meta-train time, and thus shaping the loss landscape. In *learning from demonstration (LfD)* [24, 21, 4], expert demonstrations are used for initializing robotic policies. In our experiments, we aim to mimic the availability of an expert at meta-test time by training our meta-network to optimize a behavioral cloning objective at meta-train time. We provide the meta-network with expert state-action trajectories during train time, which could be human demonstrations or, as in our experiments, trajectories optimized using iLQR. During meta-train time, the task loss is the behavioral cloning objective $\mathcal{L}_{\mathcal{T}}(\theta) = \mathbb{E}\left[\sum_{t=0}^{T-1}[\pi_{\theta_{\text{new}}}(a_t|s_t) - \pi_{\text{expert}}(a_t|s_t)]^2\right]$. Fig. 6d shows the results of our experiments in the ReacherGoal environment.

V. CONCLUSIONS

In this work we presented a framework to meta-learn a loss function entirely from data. We showed how the meta-learned loss can become well-conditioned and suitable for an

efficient optimization with gradient descent. When using the learned meta-loss we observe significant speed improvements in regression, classification and benchmark reinforcement learning tasks. Furthermore, we showed that by introducing additional guiding information during training time we can train our meta-loss to develop exploratory strategies that can significantly improve performance during the meta-test time.

We believe that the ML[3] framework is a powerful tool to incorporate prior experience and transfer learning strategies to new tasks. In future work, we plan to look at combining multiple learned meta-loss functions in order to generalize over different families of tasks. We also plan to further develop the idea of introducing additional curiosity rewards during training time to improve the exploration strategies learned by the meta-loss.

REFERENCES

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

[2] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, pages 3981–3989, 2016.

[3] Y. Bengio and S. Bengio. Learning a synaptic learning rule. Technical Report 751, Département d'Informatique et de Recherche Opérationelle, Université de Montréal, Montreal, Canada, 1990.

[4] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In *Springer Handbook of Robotics*, pages 1371–1394. Springer, 2008.

[5] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016.

[6] R. Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[7] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[8] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil. Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1165–1173. JMLR. org, 2017.

[9] E. Grefenstette, B. Amos, D. Yarats, P. M. Htut, A. Molchanov, F. Meier, D. Kiela, K. Cho, and S. Chintala. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*, 2019.

[10] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems*, pages 5302–5311, 2018.

[11] R. Houthooft, Y. Chen, P. Isola, B. C. Stadie, F. Wolski, J. Ho, and P. Abbeel. Evolved policy gradients. In *NeurIPS*, pages 5405–5414, 2018.

[12] K. Hsu, S. Levine, and C. Finn. Unsupervised learning via meta-learning. *CoRR*, abs/1810.02334, 2018.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[14] K. Li and J. Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[15] M. Lutter, C. Ritter, and J. Peters. Deep lagrangian networks: Using physics as model prior for deep learning. *arXiv preprint arXiv:1907.04490*, 2019.

[16] D. Maclaurin, D. Duvenaud, and R. Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

[17] F. Meier, D. Kappler, and S. Schaal. Online learning of a memory for learning rates. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2425–2432. IEEE, 2018.

[18] R. Mendonca, A. Gupta, R. Kralev, P. Abbeel, S. Levine, and C. Finn. Guided meta-policy search. *arXiv preprint arXiv:1904.00956*, 2019.

[19] L. Metz, N. Maheswaranathan, B. Cheung, and J. Sohl-Dickstein. Learning unsupervised learning rules. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HkNDsiC9KQ.

[20] A. Moore. Efficient memory-based learning for robot control. *PhD thesis, University of Cambridge*, 1990.

[21] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[22] OpenAI Gym, 2019.

[23] G. Parascandolo, H. Huttunen, and T. Virtanen. Taming the waves: sine as activation function in deep neural networks. 2017.

[24] D. A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[25] J. Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook. Institut für Informatik, Technische Universität München, 1987.

[26] J. Schulman, N. Heess, T. Weber, and P. Abbeel. Gradient estimation using stochastic computation graphs. In *NeurIPS*, pages 3528–3536, 2015.

[27] F. Sung, L. Zhang, T. Xiang, T. Hospedales, and Y. Yang. Learning to learn: Meta-critic networks for sample efficient learning. *arXiv preprint arXiv:1706.09529*, 2017.

[28] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, 2000.

[29] Y. Tassa, N. Mansard, and E. Todorov. Control-limited differential dynamic programming. *IEEE International Conference on Robotics and Automation, ICRA*, 2014.

[30] S. Thrun and L. Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[31] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8: 229–256, 1992.

[32] L. Wu, F. Tian, Y. Xia, Y. Fan, T. Qin, J.-H. Lai, and T.-Y. Liu. Learning to teach with dynamic loss functions. In *NeurIPS*, pages 6467–6478, 2018.

[33] Z. Xu, H. van Hasselt, and D. Silver. Meta-gradient reinforcement learning. In *NeurIPS*, pages 2402–2413, 2018.

[34] O. Yadan. A framework for elegantly configuring complex applications. Github, 2019. URL https://github.com/facebookresearch/hydra.

[35] T. Yu, C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint arXiv:1802.01557*, 2018.

[36] H. Zou, T. Ren, D. Yan, H. Su, and J. Zhu. Reward shaping via meta-learning. *arXiv preprint arXiv:1901.09330*, 2019.

Algorithm 3 ML^3 for MBRL *(meta-train)*

1: $\phi, \leftarrow$ randomly initialize parameters
2: Randomly initialize dynamics model P
3: **while** not done **do**
4:    $\theta \leftarrow$ randomly initialize parameters
5:    $\tau \leftarrow$ forward unroll π_θ using P
6:    $\pi_{\theta_{\text{new}}} \leftarrow$ optimize$(\tau, \mathcal{M}_\phi, g, R)$
7:    $\tau_{\text{new}} \leftarrow$ forward unroll $\pi_{\theta_{\text{new}}}$ using P
8:    *Update ϕ to maximize reward under P:*
9:    $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_\mathcal{T}(\tau_{\text{new}})$
10:    $\tau_{real} \leftarrow$ roll out $\pi_{\theta_{\text{new}}}$ on real system
11:    $P \leftarrow$ update dynamics model with τ_{real}
12: **end while**

Algorithm 4 ML^3 for MFRL *(meta-train)*

1: $I \leftarrow$ # of inner steps
2: $\phi \leftarrow$ randomly initialize parameters
3: **while** not done **do**
4:    $\theta_0 \leftarrow$ randomly initialize policy
5:    $\mathcal{T} \leftarrow$ sample training tasks
6:    $\tau_0, R_0 \leftarrow$ roll out policy π_{θ_0}
7:    **for** $i \in \{0, \ldots, I\}$ **do**
8:       $\pi_{\theta_{i+1}} \leftarrow$ optimize$(\pi_{\theta_i}, \mathcal{M}_\phi, \tau_i, R_i)$
9:       $\tau_{i+1}, R_{i+1} \leftarrow$ roll out policy $\pi_{\theta_{i+1}}$
10:       $\mathcal{L}_\mathcal{T}^i \leftarrow$ compute task-loss $\mathcal{L}_\mathcal{T}^i(\tau_{i+1}, R_{i+1})$
11:    **end for**
12:    $\mathcal{L}_\mathcal{T} \leftarrow \mathbb{E}\left[\mathcal{L}_\mathcal{T}^i\right]$
13:    $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_\mathcal{T}$
14: **end while**

Algorithm 5 ML^3 for RL *(meta-test)*

1: $\theta \leftarrow$ randomly initialize policy
2: **for** $j \in \{0, \ldots, M\}$ **do**
3:    $\tau, R \leftarrow$ roll out π_θ
4:    $\pi_\theta \leftarrow$ optimize$(\pi_\theta, \mathcal{M}_\phi, \tau, R)$
5: **end for**

We notice that in practice, including the policy's distribution parameters directly in the meta-loss inputs, e.g. mean μ and standard deviation σ of a Gaussian policy, works better than including the probability estimate $\pi_\theta(a|s)$, as it provides a direct way to update the distribution parameters using back-propagation through the meta-loss.

The forward model of the dynamics is represented in both cases by a neural network, the input to the network is the current state and action, the output is the next state of the environment.

The Pointmass state space is four-dimensional. For PointmassGoal $(x, y, \dot{x}, \dot{y})$ are the 2D positions and velocities, and the actions are accelerations $(\ddot{x}, \ddot{y})$.

The ReacherGoal environment for the MBRL experiments is a lower-dimensional variant of the MFRL environment. It has a four dimensional state, consisting of position and angular velocity of the joints $[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]$ the torque is two dimensional $[\tau_1, \tau_2]$ The dynamics model P is updated once every 100 outer iterations with the samples collected by the policy from the last inner optimization step of that outer optimization step, i.e. the latest policy.

The ReacherGoal environment is a 2-link 2D manipulator that has to reach a specified goal location with its end-effector. The task distribution (at meta-train and meta-test time) consists of an initial link configuration and random goal locations within the reach of the manipulator. The performance metric for this environment is the mean trajectory sum of negative distances to the goal, averaged over 10 tasks. As a trajectory reward $R_g(\tau)$ for the task-loss (see Eq. 7) we use $R_g(\tau) = -d + 1/(d + 0.001) - |a_t|$, where d is the distance of the end-effector to the goal g specified as a 2-d Cartesian position. The environment has eleven dimensions specifying angles of each link, direction from the end-effector to the goal, Cartesian coordinates of the target and Cartesian velocities of the end-effector.

The AntGoal environment requires a four-legged agent to run to a goal location. The task distribution consists of random goals initialized on a circle around the initial position. The performance metric for this environment is the mean trajectory sum of differences between the initial and the current distances to the goal, averaged over 10 tasks. Similar to the previous environment we use $R_g(\tau) = -d + 5/(d + 0.25) - |a_t|$, where d is the distance from the center of the creature's torso to the goal g specified as a 2D Cartesian position. In contrast to the ReacherGoal this environment has 33 [4] dimensional state space that describes Cartesian position, velocity and orientation of the torso as well as angles and angular velocities of all eight joints. Note that in both environments, the meta-network receives the goal information g as part of the state s in the corresponding environments. Also, in practice, including the policy's distribution parameters directly in the meta-loss inputs, e.g. mean μ and standard deviation σ of a Gaussian policy, works better than including the probability estimate $\pi_\theta(a|s)$, as it provides a more direct way to update θ using back-propagation through the meta-loss.

For the sine task at meta-train time, we draw 100 data points from function $y = \sin(x - \pi)$, with $x \in [-2.0, 2.0]$. For meta-test time we draw 100 data points from function $y = A\sin(x - \omega)$, with $A \sim [0.2, 5.0]$, $\omega \sim [-\pi, pi]$ and $x \in [-2.0, 2.0]$. We initialize our model f_θ to a simple feedforward NN with 2 hidden layers and 40 hidden units each, for the binary classification task f_θ is initialized via the *LeNet* architecture. For both regression and classification experiments we use a fixed learning rate $\alpha = \eta = 0.001$ for both inner (α) and outer (η) gradient update steps. We average results across 5 random seeds, where each seed controls the initialization of both initial model and meta-network parameters, as well as the the random choice of meta-train/test task(s), and visualize them in Fig. 2. Task losses are $\mathcal{L}_{\text{Regression}} = (y - f_\theta(x))^2$ and $\mathcal{L}_{\text{BinClass}} = CrossEntropyLoss(y, f_\theta(x))$ for regression and classification meta-learning respectively.

---

[4]In contrast to the original Ant environment we remove external forces from the state.