META LEARNING VIA LEARNED LOSS

Anonymous authors
Paper under double-blind review

ABSTRACT

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 training such loss functions, targeted at maximizing the performance of model learning with them. We observe that 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.

1 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 (Andrychowicz et al., 2016; Duan et al., 2016; Finn et al., 2017). We can classify learning-to-learn methods into roughly 2 categories: approaches that lead to learning representations that can generalize and are easily adaptable to new tasks (Finn et al., 2017), and learning approaches that attempt to learn how to optimize models (Andrychowicz et al., 2016; Duan et al., 2016). In this paper we investigate the second type of approach and propose a learning framework that is able to learn loss function representations that can then be used to optimize models for new tasks.

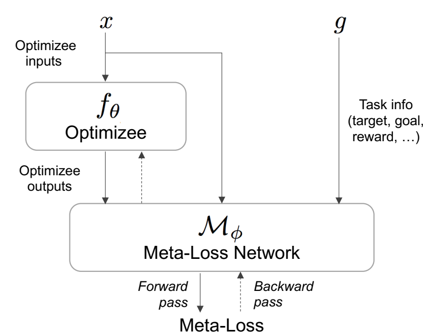


Figure 1: Framework overview

Specifically, the purpose of this work is to encode learning strategies into an adaptive high-dimensional loss function, or a *meta-loss*, which generalizes across multiple training contexts or tasks. Inspired by *inverse reinforcement learning* (Ng et al., 2000), 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 model-agnostic loss functions to provide a strong learning signal for a large range of model classes, such as classifiers, regressors or control policies.

The contributions of this work are as follows: i) we present a framework for learning adaptive, high-dimensional loss functions through back-propagation that shape the loss landscape such that it can be efficiently optimized with gradient descent. This framework involves an inner and an outer loop. 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. We show that our learned meta-loss functions improves over directly learning via the task-loss itself while maintaining the generality of the task-loss. ii) We show how we can utilize 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.

2 RELATED WORK

Meta-learning originates in the concept of learning to learn (Schmidhuber, 1987; Bengio & Bengio, 1990; Thrun & Pratt, 2012). Recently, there has a been a wide interest in finding ways to improve learning speeds and generalization to new tasks through meta-learning. The main directions of the research in this area can be divided into learning representations that can be easily adapted to new tasks (Finn et al., 2017), learning unsupervised rules that can be transferred between tasks (Metz et al., 2019; Hsu et al., 2018), learning optimizer policies that transform policy updates with respect to known loss or reward functions (Maclaurin et al., 2015; Andrychowicz et al., 2016; Li & Malik, 2016; Franceschi et al., 2017; Meier et al., 2018; Duan et al., 2016), or learning loss/reward landscapes (Sung et al., 2017; Houthooft et al., 2018).

Our framework falls into the category of learning loss landscapes; similar to work by Andrychowicz et al. (2016), we aim at learning a loss function that can be applied to various optimizee models. However, in contrast to the work of Andrychowicz et al. (2016) and Duan et al. (2016), our framework does not require a specific recurrent architecture of the optimizer and can operate without an explicit external loss or reward function during test time. Furthermore, as our learned loss functions are independent of the models to be optimized, they can be easily transferred to other optimizee models, in contrast to MAML (Finn et al., 2017), where the learned representation can not 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 (Ng et al., 2000; Abbeel & Ng, 2004, IRL). However, in contrast to the original goal of IRL of inferring reward functions from expert demonstrations, in our work we aim at extending this idea and learning loss functions that can improve learning speeds and generalization for a wider range of applications. Furthermore, we design our framework to be fully differentiable, facilitating the training of both the learned meta-loss and optimizee models.

A range of recent works demonstrate advantages of meta-learning for improving exploration strategies in RL settings, especially in the presence of sparse rewards. Mendonca et al. (2019) propose training an agent to mimic expert demonstrations while only having access to a sparse reward signal during test time. In the work of Hausman et al. (2018) and Gupta et al. (2018), a structured latent exploration space is learned from prior experience, which enables fast exploration in novel tasks. Zou et al. (2019) propose a method for automatically learning potential-based reward shaping by learning the Q-function parameters during the meta-training phase, such that at meta-test time the Q-function can adapt quickly to new tasks. In our work, we also demonstrate that we can significantly improve the RL sample efficiency by training our meta-loss to optimize an actor policy, even when providing only limited or no reward information to the learned loss function at test time.

Closest to our method are the works on *evolved policy gradients* (Houthooft et al., 2018), *teacher networks* (Wu et al., 2018), *meta-critics* (Sung et al., 2017) and *meta-gradient RL* (Xu et al., 2018). In contrast to using an evolutionary approach (e.g. Houthooft et al., 2018), 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. (2018) 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. (2018), 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. (2017) 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.

3 META-LEARNING VIA LEARNED LOSS

In this work, we aim to learn a loss function, which we call *meta-loss*, that is used to train an *optimizee*, e.g. a classifier, a regressor or an agent policy. In the following, we describe our approach, which we

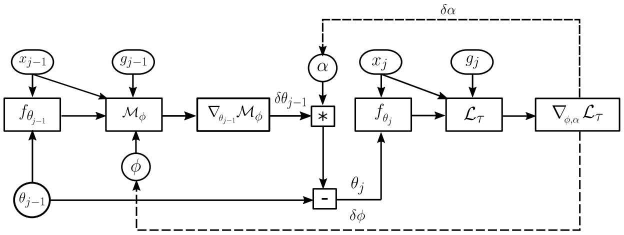


Figure 2: Meta-Learning via Learned Loss (ML^3) framework overview. The parameters of an optimizee are first updated using the meta-loss. Then parameters of the meta-network are updated using the task-specific loss calculated on the updated optimizee. The dashed lines show the gradients for the meta-network parameters with respect to the task-specific loss.

Algorithm 1 ML^3 at (*meta-train*)

$\phi, \theta \leftarrow$ randomly initialize
while not done **do**
  $x, y \leftarrow$ Sample task samples
  $f_{\theta_{\text{new}}} \leftarrow$ optimize$(f_\theta, \mathcal{M}_\phi, x, y)$
  $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_\mathcal{T}(y, f_{\theta_{\text{new}}})$

Algorithm 2 ML^3 at (*meta-test*)

$M \leftarrow$ # of optimizee updates
Randomly initialize f_θ
for $j \in \{0, \dots, M\}$ **do**
  $x, y \leftarrow$ Sample task samples
  $\theta \leftarrow \theta - \alpha \nabla_\theta \, \mathbb{E}_x \left[\mathcal{M}_\phi(y, f_\theta(x))\right]$

call **M**eta-**L**earning via **L**earned **L**oss (ML^3). Let f_θ be an optimizee with parameters θ. Let $\mathcal{M}_\phi$ be the meta-loss model with parameters ϕ. Let x be the inputs of the optimizee, $f_\theta(x)$ the outputs of the optimizee and g information about the task, such as a regression or classification target y, a reward function R, etc.

Let $\mathcal{T}$ be a task and $\mathcal{L}_\mathcal{T}(\theta)$ be the task-specific loss of the optimizee f_θ for the task $\mathcal{T}$. Fig. 2 shows the diagram of our framework architecture for a single step of the optimizee update. The optimizee is connected to the meta-loss network, which allows the gradients from the meta-loss to flow through the optimizee. The meta-loss additionally takes the inputs of the optimizee and the task information variable g. In our framework, we represent the meta-loss function using a neural network, which is subsequently referred to as a meta-loss network. Next, we make the implementation of our meta-learning framework concrete for supervised learning in Section 3.1, and for model-based and model-free reinforcement learning in Section 3.2.

3.1 ML^3 for Supervised Learning

For supervised learning, our framework aims at learning a meta-loss function $\mathcal{M}_\phi(y, f_\theta(x))$ that predicts the loss given the ground truth target y and the predicted target $f_\theta(x)$. This is achieved as follows: a single update of the optimizee $f_\theta(x)$ is performed using gradient descent on the meta-loss by back-propagating the output of the meta-loss network through the optimizee keeping the parameters of the meta-loss network fixed:

$$\theta_{\text{new}} = \theta - \alpha \nabla_\theta \mathbb{E}_x \left[\mathcal{M}_\phi(y, f_\theta(x))\right], \tag{1}$$

where α is the learning rate, which can be either fixed or learned jointly with the meta-loss network. This *inner optimization* uses the current meta-loss network parameters to optimize the model for the supervised learning task $\mathcal{T}$. For simplicity, we illustrate the process for one inner gradient step, but in principle (and in practice) multiple gradient steps can be taken. The inner loop optimization updates the meta-loss parameters ϕ by evaluating the task-loss, for instance a Mean-Squared-Error Loss in the context of regression, on the new model $f_{\theta_{\text{new}}}$: $\mathcal{L}_\mathcal{T}(y, f_{\theta_{\text{new}}}(x)) = \mathcal{L}_\mathcal{T}\left(y, f_{\theta - \alpha\nabla_\theta \mathbb{E}_x\left[\mathcal{M}_\phi(y, f_{\theta(x)})\right]}(x)\right)$, which now is a function of the meta-parameters ϕ. This makes it possible to compute the gradient of the task-loss $\mathcal{L}$ with respect to meta-parameters ϕ and propagate the error back through the meta-loss network, as visualized in Fig. 2. Optimization of the meta-parameters can either happen after *each inner gradient* step, or after M *inner gradient steps* with the current meta-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. Algorithm 1 summarizes the training procedure of the meta-loss network, which we later refer to as *meta-train*. Algorithm 2 shows the optimizee training with the learned meta-loss at test time, on a new task, which we call *meta-test*[1].

3.2 ML^3 Reinforcement Learning

In this section, we introduce several modifications that allow us to apply the ML^3 framework to reinforcement learning problems. Let $\mathcal{M} = (S, A, P, R, p_0, \gamma, T)$ be a finite-horizon Markov

[1]The source code will be available at the time of the final submission.

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 $\tau = (s_0, a_0, \dots, s_T, a_T)$ 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 $s_0 \sim p_0, s_{t+1} \sim P(s_{t+1}|s_t, a_t)$ 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 ML^3 framework, we replace the optimizee f_θ from the previous section with a stochastic policy $\pi_\theta(a|s)$. We present two applications of ML^3 to RL.

3.2.1 ML^3 for Model-Based Reinforcement Learning

Model-based RL (MBRL) attempts to learn a policy π 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 such that it maximizes the reward R under P, then use the optimized policy π 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 $\tau = (s_0, a_0, \dots, s_T, a_T)$ 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.1) illustrates the MBRL learning loop. In Algorithm 5 (Appendix A.1), 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.

3.2.2 ML^3 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 (Williams, 1992; Sutton et al., 2000; Schulman et al., 2015):

$$\mathcal{L}_\mathcal{T}(g, \pi_{\theta_{\text{new}}}) = -\mathbb{E}_{\pi_{\theta_{\text{new}}}}\left[R_g(\tau_{\text{new}}) \log \pi_{\theta_{\text{new}}}(\tau_{\text{new}})\right] = -\mathbb{E}_{\pi_{\theta_{\text{new}}}}\left[R_g(\tau_{\text{new}}) \sum_{t=0}^{T-1} \log \pi_{\theta_{\text{new}}}(a_t|s_t)\right] \quad (2)$$

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. 1 using stochastic gradient descent (SGD) on the meta-loss with mini-batches of experience (s_i, a_i, r_i) for $i \in \{0, \dots, B-1\}$ with batch size B, similar to Houthooft et al. (2018). The inputs of the meta-loss network are the sampled states, sampled actions, task information g and policy probabilities of the sampled actions: $\mathcal{M}_\phi\left(s, a, \pi_\theta(a|s), g\right)$. 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.1).

3.3 The task loss and adding extra 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. In our work, we experiment with 3 different types of extra information at *meta-train* time: for supervised learning we show that providing ground truth information about the optimal parameters can help shape a convex loss-landscape for otherwise non-convex optimization problems; 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 also show how expert demonstrations can be incorporated to learn loss functions which can generalize to new tasks. In all these settings, the additional information shapes the learned loss function such that the environment does not need to provide this information during meta-test time.

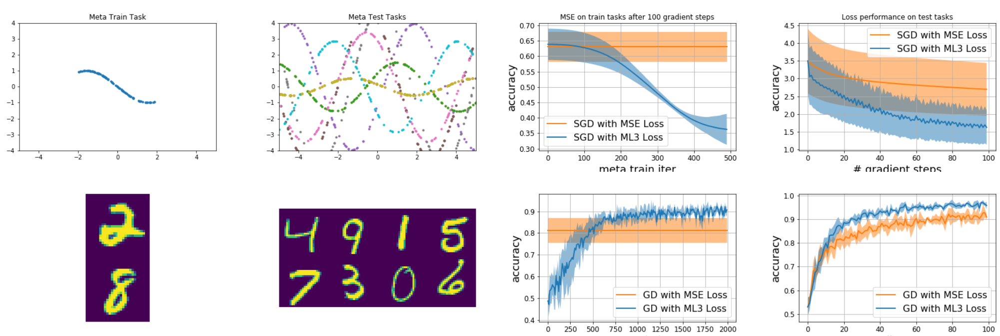
4 Experiments

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 4.1. 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 in Section 4.2.

4.1 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.

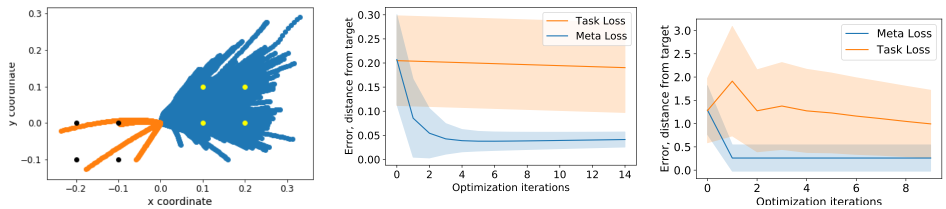
4.1.1 Meta-Loss for Supervised Learning



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

Figure 3: 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

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.4). At meta-train time, we randomly draw one task for meta-training (see Fig. 3 (a)), and at meta-test time we randomly draw 10 test tasks for regression, and 4 test tasks for classification (Fig. 3(b)). We compare the performance of using SGD with the task-loss $\mathcal{L}$ directly (in orange) to SGD using the learned meta-network $\mathcal{M}$ (in blue), both using a learning rate $\alpha = 0.001$. In Fig. 3 (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. 3 (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.



(a) Generalization Pointmass (b) Meta vs Task Loss Pointmass (c) Meta vs Task Loss Reacher

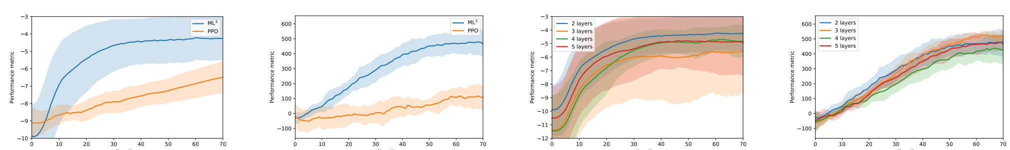
Figure 4: Results of ML^3 for MBRL. We can see that, for MBRL the meta-loss generalizes well (a) and speeds up learning when compared to the task-specific loss during meta test. The results are shown for the PointmassGoal (a and b) and the ReacherGoal (c) environments.

4.1.2 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.2 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 $R(\tau)$ 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. 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. 4a shows the qualitative reaching performance of a policy optimized with the meta loss during test on PointmassGoal. The meta-loss network was trained only on tasks in the right quadrant and tested on the tasks in the left quadrant of the x, y plane, showing the generalization capability of the meta loss. Figure 4b and 4c 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 just 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 using the task loss alone, 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 3.2.1, this is not needed when using the meta-loss.

4.1.3 Learning Reward functions for Model-free Reinforcement Learning

In the following, we move to evaluating on model-free RL tasks. Fig. 5 shows results when using two continuous control tasks based on OpenAI Gym MuJoCo environments (Gym, 2019): ReacherGoal and AntGoal (see Appendix A.3 for details).



(a) ReacherGoal (b) AntGoal (c) ReacherGoal (d) AntGoal

Figure 5: (a+b) Policy learned with ML^3 loss compared to PPO performance during *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. Each curve is an average over ten different tasks.

Fig. 5a and Fig. 5b 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 our meta-loss on an architecture with two layers and meta-test the same meta-loss on architectures with varied number of layers. Fig. 5 (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.

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

This set of experiments shows that our meta-learner is able to learn convex loss functions for tasks with inherently non-convex or difficult to optimize loss landscapes. Effectively, the meta-loss allows eliminating local minima for gradient-based optimization and creates well-conditioned loss landscapes.

4.2.1 Shaping loss for regression

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.

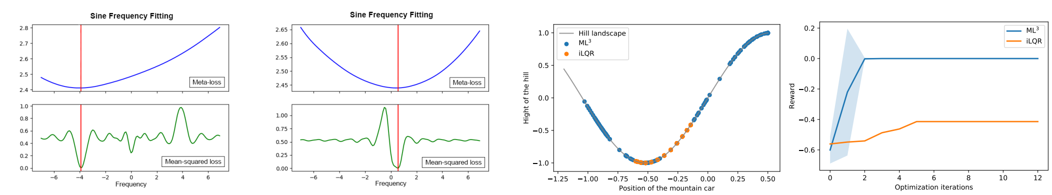


Figure 6: *Left:* Comparison of learned meta-loss (top) and mean-squared loss (bottom) landscapes for fitting the frequency of a sine function. The red lines indicate the target values of the frequency. *Right:* Improved exploration behavior in the MountainCar environment when using ML^3 with intermediate goals during meta-train time and average distance to the goal at the final timestep.

Fig. 6 (left) shows loss landscapes for fitting the frequency parameter ω of the sine function $f(x) = \sin(\omega x)$. Below, we show the landscape of optimization with mean-squared loss on the outputs of the sine function using 1000 samples from the target function. The target frequency ν is indicated by a vertical red line, and the mean-squared loss is computed as $\frac{1}{N}\sum_{i=0}^{N}(sin(\omega x_i) - sin(\nu x_i))^2$. As noted by Parascandolo et al. (2017), the landscape of this loss is highly non-convex and difficult to optimize with conventional gradient descent. In our work, we can circumvent this problem by introducing additional information about the ground truth value of the frequency at meta-train time, however only using samples from the sine function at inputs to the meta-network. That is, during the meta-train time, our task-specific loss is the squared distance to the ground truth frequency: $(\omega - \nu)^2$. The inputs of the meta-network are the target values of the sine function: $sin(\nu x_i)$, similar to the information available in the mean-squared loss. Effectively, during the meta-test time we can use the same samples as in the mean-squared loss, however achieve convex loss landscapes as depicted in Fig. 6 (left) at the top.

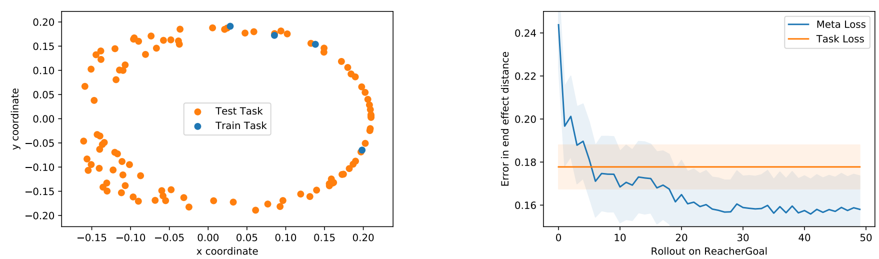
By providing additional reward information during meta-train time, as pointed out in Section 3.3, it is possible to shape the learned reward signal such that it improves the optimization during policy training. By having access to additional information during meta-training, the meta-network can learn a loss function that provides exploratory strategies to the agent or allows the agent to learn in a self-supervised setting.

4.2.2 Shaping Loss for RL

We analyze loss landscape shaping on the MountainCar environment (Moore, 1990), 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. 6-3, when compared to a classical iLQR-based trajectory optimization (Tassa et al., 2014). Fig. 6-4 shows the average distance between the car and the goal at last rollout time step over several iterations of policy updates with ML^3 and iLQR. As we observe, ML^3 can successfully bring the car to the goal in a small amount of updates, whereas iLQR is not able to solve this task.

4.2.3 Expert information during 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)* (Pomerleau, 1991; Ng et al., 2000; Billard et al., 2008), 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 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. 7 shows the results of our experiments in the ReacherGoal environment.



(a) Train and test targets

(b) Meta vs. Task Loss at test time

Figure 7: ReacherGoal with expert demonstrations available during meta-train time. (a) 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 (b) we show the reaching performance of a policy trained with ML^3 at meta-test time, compared to the performance of training simply on the behavioral cloning objective and testing on test targets.

5 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

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004. URL `citeseer.ist.psu.edu/662229.html`.

Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, pp. 3981–3989, 2016. URL `http://dblp.uni-trier.de/db/conf/nips/nips2016.html#AndrychowiczDCH16`.

Yoshua Bengio and Samy Bengio. Learning a synaptic learning rule. Technical Report 751, Département d'Informatique et de Recherche Opérationelle, Université de Montréal, Montreal, Canada, 1990.

Aude Billard, Sylvain Calinon, Rüdiger Dillmann, and Stefan Schaal. Robot programming by demonstration. In *Springer Handbook of Robotics*, pp. 1371–1394. Springer, 2008.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016. URL `http://dblp.uni-trier.de/db/journals/corr/corr1611.html#DuanSCBSA16`.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017. URL `http://dblp.uni-trier.de/db/conf/icml/icml2017.html#FinnAL17`.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1165–1173. JMLR. org, 2017.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems*, pp. 5302–5311, 2018.

OpenAI Gym, 2019.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rk07ZXZRb`.

Rein Houthooft, Yuhua Chen, Phillip Isola, Bradly C. Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. In *NeurIPS*, pp. 5405–5414, 2018. URL `http://dblp.uni-trier.de/db/conf/nips/nips2018.html#HouthooftCISWHA18`.

Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning. *CoRR*, abs/1810.02334, 2018. URL `http://dblp.uni-trier.de/db/journals/corr/corr1810.html#abs-1810-02334`.

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.

Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pp. 2113–2122, 2015.

Franziska Meier, Daniel Kappler, and Stefan Schaal. Online learning of a memory for learning rates. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2425–2432. IEEE, 2018.

Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Guided meta-policy search. *arXiv preprint arXiv:1904.00956*, 2019.

Luke Metz, Niru Maheswaranathan, Brian Cheung, and Jascha Sohl-Dickstein. Learning unsupervised learning rules. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HkNDsiC9KQ`.

Andrew Moore. Efficient memory-based learning for robot control. *PhD thesis, University of Cambridge*, 1990.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pp. 663–670, 2000.

Giambattista Parascandolo, Heikki Huttunen, Tao Xiang, Timothy Hospedales, and Tuomas Virtanen. Taming the waves: sine as activation function in deep neural networks. *Submitted to ICLR*, 2017.

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

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

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *NeurIPS*, pp. 3528–3536, 2015. URL `http://dblp.uni-trier.de/db/conf/nips/nips2015.html#SchulmanHWA15`.

Flood Sung, Li Zhang, Tao Xiang, Timothy Hospedales, and Yongxin Yang. Learning to learn: Meta-critic networks for sample efficient learning. *arXiv preprint arXiv:1706.09529*, 2017.

Richard Sutton, David McAllester, Satinder Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, 2000.

Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. *IEEE International Conference on Robotics and Automation, ICRA*, 2014.

Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

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

Lijun Wu, Fei Tian, Yingce Xia, Yang Fan, Tao Qin, Jian-Huang Lai, and Tie-Yan Liu. Learning to teach with dynamic loss functions. In *NeurIPS*, pp. 6467–6478, 2018. URL `http://dblp.uni-trier.de/db/conf/nips/nips2018.html#WuTXFQLL18`.

Zhongwen Xu, Hado van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *NeurIPS*, pp. 2402–2413, 2018.

Haosheng Zou, Tongzheng Ren, Dong Yan, Hang Su, and Jun Zhu. Reward shaping via meta-learning. *arXiv preprint arXiv:1901.09330*, 2019.

A Appendix

A.1 MFRL and MBRL algorithms details

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

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

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

$\phi, \theta \leftarrow$ randomly initialize parameters
while not done **do**
  $\pi_{\theta_{\text{new}}} \leftarrow$ optimize$(\pi_\theta, \mathcal{M}_\phi, g, R)$
  $\tau_{new} \leftarrow$ roll out new policy $\pi_{\theta_{\text{new}}}$
  $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_\mathcal{T}(\tau_{new})$

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

Randomly initialize policy π_θ
for $j \in \{0, \dots, M\}$ **do**
  $\tau \leftarrow$ roll out π_θ in environment
  $\theta \leftarrow \theta - \alpha \nabla_\theta \mathbb{E}\left[\mathcal{M}_\phi(s, \tau, R)\right]$

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.

A.2 EXPERIMENTS: MBRL

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 $[\tau_1, \tau_2]$

A.3 EXPERIMENTS: MFRL

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 initial random link configurations 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. 2) 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 [2] 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.

A.4 EXPERIMENTS: REGRESSION AND CLASSIFICATION DETAILS

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 (Lecun et al., 1998). 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. 3. Task losses are $\mathcal{L}_{\text{Regression}} = (y - f_\theta(x))^2$ and $\mathcal{L}_{\text{BinClass}} = \textit{CrossEntropyLoss}(y, f_\theta(x))$ for regression and classification meta-learning respectively.

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