Meta-Learning via Learned Loss

Yevgen Chebotar [*1] Artem Molchanov [*1] Sarah Bechtle [*2] Ludovic Righetti [2 3] Franziska Meier [4]
Gaurav Sukhatme [1]

Abstract

We present a meta-learning approach based on learning an adaptive, high-dimensional loss function that can generalize across multiple tasks and different model architectures. We develop a fully differentiable pipeline for learning a loss function targeted at maximizing the performance of an optimizee trained using this loss function. We observe that the loss landscape produced by our learned loss significantly improves upon the original task-specific loss. We evaluate our method on supervised and reinforcement learning tasks. Furthermore, we show that our pipeline is able to operate in sparse reward and self-supervised reinforcement learning scenarios.

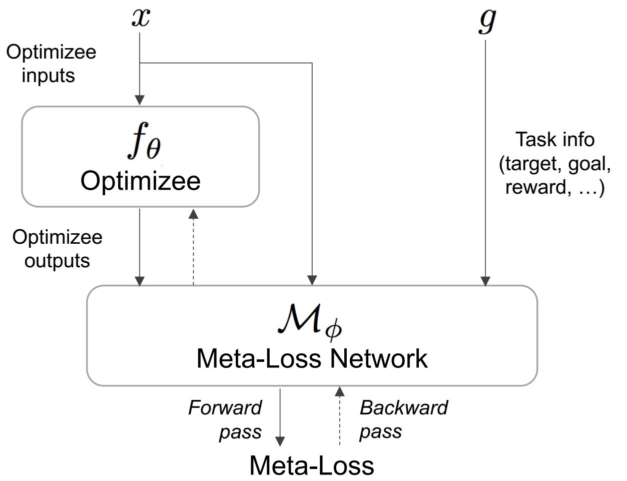


Figure 1. Using a learned meta-loss to update an optimizee model.

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). When thinking about optimizing a policy for a reinforcement learning agent or learning a classification task, it appears sensible to not approach each individual task from scratch but to learn a learning mechanism that is common across a variety of tasks and can be reused. The purpose of this work is to encode these learning strategies into an adaptive high-dimensional loss function, or a *meta-loss*, which generalizes across multiple tasks and can be utilized to optimize models with different architectures. 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 shape the loss function to provide a strong learning signal for a range of various models, such as classifiers, regressors or control policies. As the loss function is independent of the model being optimized, it is agnostic to the particular model architecture. Furthermore, by training our loss function to optimize different tasks, we can achieve generalization across multiple problems. The meta-learning framework presented in this work 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 the task-specific losses of updated optimizees. 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. In this way, our meta-loss can find more efficient ways to optimize the original task loss. Furthermore, since we can choose which information to provide to our meta-loss, we can train it to work in scenarios with sparse information by only providing inputs that we expect to have at test time. The contributions of this work are as follows: 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; we show that our learned meta-loss functions are agnostic to the archi-

[*]Equal contribution [1]University of Southern California, Los Angeles, CA, USA [2]Max Planck Institute for Intelligent Systems, Tübingen, Germany [3]New York University, New York, NY USA [4]Facebook AI Research, Menlo Park, CA, USA. Correspondence to: Yevgen Chebotar <ychebota@usc.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

tecture of optimizee models; and we present a reinforcement learning framework that significantly improves the speed of policy training and enables learning in self-supervised and sparse reward settings.

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 (Andrychowicz et al., 2016; Li & Malik, 2016; 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 (Andrychowicz et al., 2016), we aim at learning a separate optimization procedure that can be applied to various optimizee models. However, in contrast to (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 (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 (IRL) (Ng et al., 2000; Abbeel & Ng, 2004). 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. In (Mendonca et al., 2019), an agent is trained to mimic expert demonstrations while only having access to a sparse reward signal during test time. In (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) proposes 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) and *meta-critics* (Sung et al., 2017). In contrast to using an evolutionary approach as in (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. In (Wu et al., 2018), 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. Our method does not require manual design of the loss function parameterization as our loss functions are learned entirely from data. Finally, in (Sung et al., 2017) a *meta-critic* is learned to provide a value function conditioned on a task, used to train an actor policy. Although training a meta-critic in the supervised setting reduces to learning a loss function similar to 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 an adaptive 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 the general architecture of our framework, which we call **M**eta-**L**earning via **L**earned **L**oss (ML^3).

3.1. ML^3 framework

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, $f_\theta(x)$ outputs of the optimizee and g information about the task, such as a regression target, a classification target, a reward function, etc. Let $p(\mathcal{T})$ be a distribution of tasks and $\mathcal{L}_{\mathcal{T}_i}(\theta)$ be the task-specific loss of the optimizee f_θ for the task $\mathcal{T}_i \sim p(\mathcal{T})$.

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.

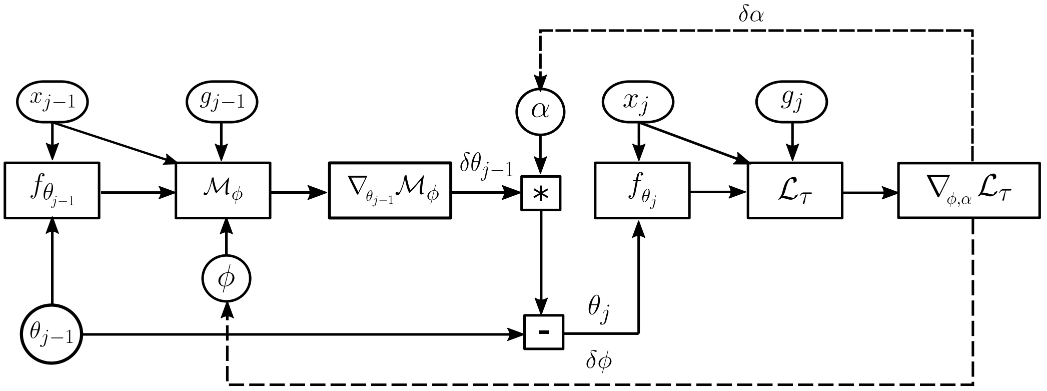


Figure 2. Meta-Learning via Learned Loss (ML^3) framework overview. The parameters of an optimizee are first updated using the meta-loss. Afterwards, the parameters of the meta-loss network and the learning rate are updated using the task-specific loss calculated on the updated optimizee. The dashed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

It is worth noting that it is possible to train the meta-loss to perform self-supervised learning by not including g in the meta-loss network inputs. A single update of the optimizee 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:

$$\theta_j = \theta_{j-1} - \alpha\nabla_{\theta_{j-1}}\mathbb{E}\left[\mathcal{M}_\phi(x, f_{\theta_{j-1}}(x), g)\right], \quad (1)$$

where α is the learning rate, which can be either fixed or learned jointly with the meta-loss network. The objective of learning the meta-loss network is to minimize the task-specific loss over a distribution of tasks $\mathcal{T}_i \sim p(\mathcal{T})$ and over multiple steps of optimizee training with the meta-loss:

$$\mathcal{L}(\phi,\alpha) = \sum_{i=0}^{N}\sum_{j=1}^{M}\mathcal{L}_{\mathcal{T}_i}(\theta_{i,j}) = \sum_{i=0}^{N}\sum_{j=1}^{M}\mathcal{L}_{\mathcal{T}_i}(\theta_{i,j-1} - \alpha\nabla_{\theta_{i,j-1}}\mathbb{E}[\mathcal{M}_\phi(x_i, f_{\theta_{i,j-1}}(x_i), g_i)]), \quad (2)$$

where N is the number of tasks and M is the number of steps of updating the optimizee using the meta-loss. The task-specific objective $\mathcal{L}(\phi,\alpha)$ depends on the updated optimizee parameters θ_j and hence on the parameters of the meta-loss network ϕ, making it possible to connect the meta-loss network to the task-specific loss and propagate the error back through the meta-loss network. Another variant of this objective would be to only optimize for the final performance of the optimizee at the last step M of applying the meta-loss: $\mathcal{L}(\phi,\alpha) = \sum_{i=0}^{N}\mathcal{L}_{\mathcal{T}_i}(\theta_{i,M})$. However, this requires relying on back-propagation through a chain of all optimizee update steps. As we noticed in our experiments, including the task loss from each step and avoiding propagating it through the chain of updates by stopping the gradients at each optimizee update step works better in practice.

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

```
$p(\mathcal{T}) \leftarrow$ Distribution of tasks
$N \leftarrow$ Number of tasks per batch
$M \leftarrow$ Number of optimizee updates
$K \leftarrow$ Number of unrolls per iteration
while not done do
    Sample a batch of tasks $\mathcal{T}_0, \ldots, \mathcal{T}_N \in p(\mathcal{T})$
    Randomly initialize optimizees $f_{\theta_0}, \ldots, f_{\theta_N}$
    for unroll $k \in \{0, \ldots, K\}$ do
        $\phi, \alpha \leftarrow \min_{\phi,\alpha}\sum_{i=0}^{N}\sum_{j=1}^{M}\mathcal{L}_{\mathcal{T}_i}(\theta_{i,j-1} - \alpha\nabla_{\theta_{i,j-1}}\mathbb{E}[\mathcal{M}_\phi(x_i, f_{\theta_{i,j-1}}(x_i), g_i)])$
```

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

```
$\mathcal{T} \in p(\mathcal{T}) \leftarrow$ Sample a new task
$M \leftarrow$ Number of optimizee updates
Randomly initialize optimizee $f_\theta$
for $j \in \{0, \ldots, M\}$ do
    $x, g \leftarrow$ Sample a batch of task samples
    $\theta \leftarrow \theta - \alpha\nabla_\theta\mathbb{E}\left[\mathcal{M}_\phi(x, f_\theta(x), g)\right]$
```

In order to facilitate the optimization of the meta-loss network for long optimizee update horizons, we split the optimization of $\mathcal{L}(\phi,\alpha)$ into several steps with smaller horizons, which we denote $unrolls$ similar to (Andrychowicz et al., 2016). 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, which we call *meta-test*

3.2. ML^3 for 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 Decision Process (MDP), where S and A are state and action spaces, $P : S \times A \times S \rightarrow \mathbb{R}_+$ is a state-transition probability function or system dynamics,

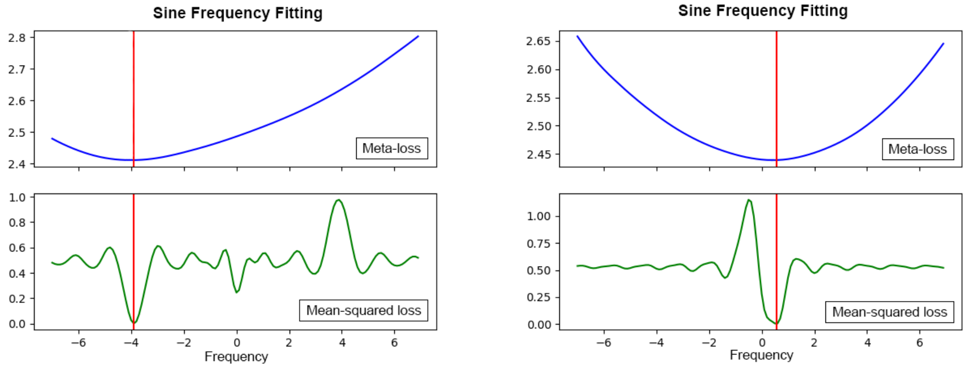


Figure 3. Comparison of learned meta-loss (top) and mean-squared loss (bottom) landscapes for fitting the frequency of a sine function. The red lines indicate the target values of the frequency.

$R : S \times A \rightarrow \mathbb{R}$ a reward function, $p_0 : S \rightarrow \mathbb{R}_+$ 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 $R(\tau) = \sum_{t=0}^{T} \gamma^t R(s_t, a_t)$ the trajectory reward. 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.

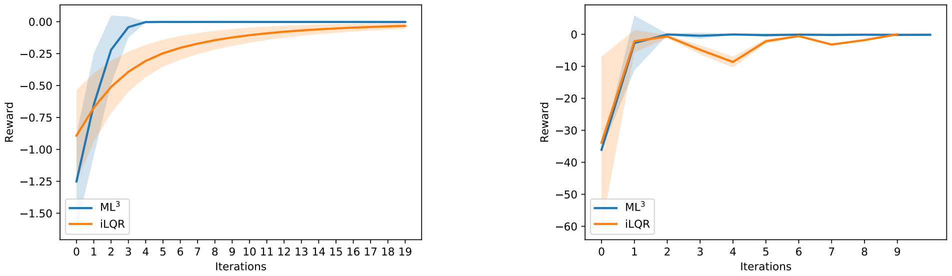
To apply our ML^3 framework, we replace the optimizee f_θ from the previous section with a stochastic policy $\pi_\theta(a|s)$. We present two cases for applying ML^3 to RL tasks. In the first case, we assume availability of a differentiable system dynamics model and a reward function. In the second case, we assume a fully model-free scenario with a non-differentiable reward function. In the case of an available differentiable system dynamics model P and a reward function R, the ML^3 objective derived in Eq. 2 can be applied directly by setting the task loss to $\mathcal{L}_\mathcal{T}(\theta) = -\mathbb{E}_{\pi_\theta}[R(\tau)]$ and differentiating all the way through the reward function, dynamics model and the policy that was updated using the meta-loss $\mathcal{M}_\phi$. In many realistic scenarios, we have to assume unknown system dynamics models and non-differentiable reward functions. 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):

$$\mathcal{L}_\mathcal{T}(\theta) = -\mathbb{E}_{\pi_\theta}\left[R(\tau)\sum_{t=0}^{T}\log \pi_\theta(a_t|s_t)\right]$$

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 $i \in \{0, \ldots, B\}$ with batch size B, similar to (Houthooft et al., 2018). The inputs of the meta-loss network are the sampled states, sampled actions, rewards and policy probabilities of the sampled actions: $\mathcal{M}_\phi\left(s, a, \pi_\theta(a|s), r\right)$. 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. As we mentioned before, it is possible to provide different information about the task during meta-train and meta-test times. In our work, we show that by providing additional rewards in the task loss during meta-train time, we can encourage the trained meta-loss to learn exploratory behaviors. This additional information shapes the learned loss function such that the environment does not need to provide this information during meta-test time. It is also possible to train the meta-loss in a fully self-supervised fashion, where the task related input g is excluded from the meta-network input.

4. Experiments

In this section we evaluate the applicability and the benefits of the learned meta-loss under a variety of aspects. The questions we seek to answer are as follows. (1) Can we learn a loss model that improves upon the original task-specific loss functions, i.e. can we shape the loss landscape to achieve better optimization performance during test time? With an example of a simple regression task, we demonstrate that our framework can generate convex loss landscapes suitable for fast optimization. (2) Can we improve the learning speed when using our ML^3 loss function as a learning signal in complex, high-dimensional tasks? We concentrate on reinforcement learning tasks as one of the most challenging benchmarks for learning performance. (3) Can we learn a loss function that can leverage additional information during meta-train time and can operate in sparse reward or self-supervised settings during meta-test time? (4)



(a) PointmassGoal environment

(b) InvertedPendulum environment

Figure 4. Policy learned with ML^3 loss compared to trajectories optimized with iLQR

Can we learn a loss function that generalizes over different optimizee model architectures?

Throughout all of our experiments, the meta network is parameterized by a feed-forward neural network with two hidden layers of 40 neurons each with $tanh$ activation function. The learning rate for the optimizee network was learned together with the loss.

4.1. Learned Loss Landscape

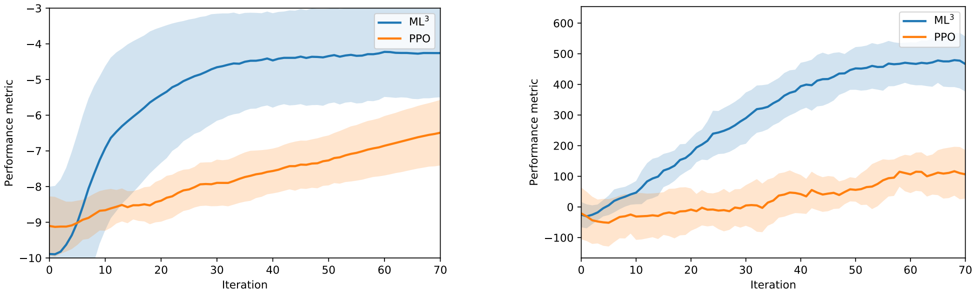
For visualization and illustration purposes, this set of experiments shows that our meta-learner is able to learn convex loss functions for tasks with inherently non-convex or difficult to optimize loss landscapes. Effectively, the meta-loss allows eliminating local minima for gradient-based optimization and creates well-conditioned loss landscapes. We illustrate this on an example of sine frequency regression where we fit a single parameter for the purpose of visualization simplicity. Fig. 3 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 in (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-loss 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-loss 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. 3 at the top.

4.2. Reinforcement Learning

For the remainder of the experimental section, we focus on reinforcement learning tasks. Reinforcement learning still remains one of the most challenging problems when it comes to learning performance and learning speed. In this section, we present our experiments on a variety of policy optimization problems. We use ML^3 for model-based and model-free reinforcement learning, thus demonstrating applicability of our approach in both settings. In the former, as mentioned in Section 3.2, we assume access to a differentiable reward function and dynamics model that could be available either a priori or learned from samples with differentiable function approximators, such as neural networks. This scenario formulates the task loss as a function of differentiable trajectories enabling direct gradient based optimization of the policy, similar to the trajectory optimization methods such as the iterative Linear-Quadratic Regulators (iLQR) (Tassa et al., 2014). In the model-free setting, we treat the dynamics of the system as a black box. In this case, the direct differentiation of the task loss is not possible and we formulate the learning signal for the meta-loss network as a surrogate policy gradient objective. See Section 3.2 for the detailed description. The policy $\pi_\theta(a|s)$ is represented by a feed-forward neural network in all experiments.

4.2.1. Sample efficiency

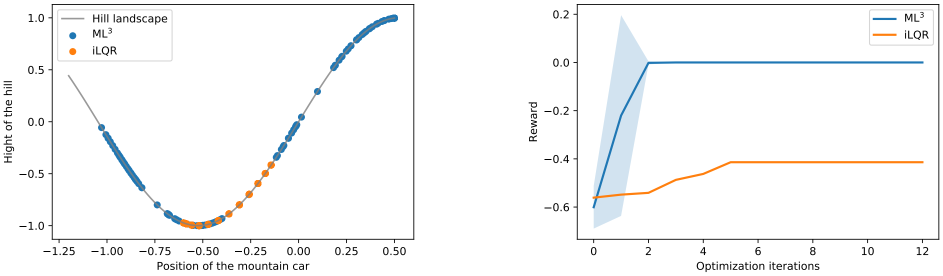
We are now presenting our results for continuous control reinforcement learning tasks, by comparing task performance of a policy trained with our meta-loss, to a policy optimized with an appropriate comparison method. When a model is available, we compare the performance with a gradient based optimizer, in this case iLQR (Tassa et al., 2014). iLQR has wide-spread application in robotics (Levine & Koltun, 2013; Koenemann et al., 2015) and is therefore a suitable comparison method for approaches that require the knowledge of a model. In the model-free setting, we use a popular policy gradient method - Proximal Policy Optimization (PPO) (Schulman et al., 2017) for comparison.



(a) ReacherGoal environment

(b) AntGoal environment

Figure 5. Policy learned with ML^3 loss compared to PPO performance

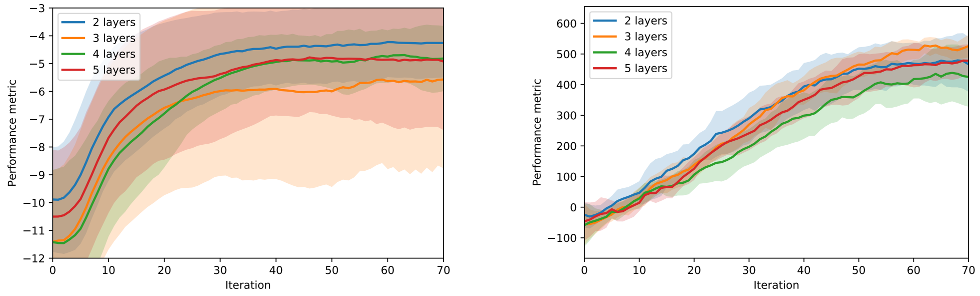


(a) MountainCar exploration behavior.

(b) Average distance to the goal at the last time-step.

Figure 6. Improved exploration behavior in the MountainCar environment when using ML^3 with intermediate goals during meta-train time, compared to iLQR.

We first evaluate our method on simple, classical continuous control problems where the dynamics are known and then continue with higher-dimensional problems where we do not have full knowledge of the model. In Fig. 4a, we compare a policy optimized with the learning signal coming from the meta-loss network to trajectories optimized with iLQR. The task is a free movement task of a point mass in a 2D space with known dynamics parameters, we call this environment PointmassGoal. The state space is four-dimensional where $(x, y, \dot{x}, \dot{y})$ are the 2D positions and velocities, and the actions are accelerations $(\ddot{x}, \ddot{y})$. The task distribution $p(\mathcal{T})$ consists of different target positions that the point mass should reach. The task-specific loss at training time is defined by the distance from the target at the last time step during the rollout. In Fig. 4a, we average the learning performance over ten random goals. We observe that the policies optimized with the learned meta-loss converge faster and can get closer to the targets compared to the trajectories optimized with iLQR. We would like to point out that on top of the improvement in convergence rates, in contrast to iLQR our trained meta-loss does not require a differentiable dynamics model nor a differentiable reward function as its input at meta-test time as it updates the policy directly through gradient descent. In Fig. 4b, we provide a similar comparison on the task that requires to swing up and balance an inverted pendulum. In this task, the state space is three dimensional: $(sin(\theta), cos(\theta), \dot{\theta})$, where θ is the angle of the pendulum. The action is a one dimensional torque. The task distribution consists of different initial angle configurations the pendulum starts in. The plot shows the averaged result over ten different initial configurations of the pendulum. From the figure we can see that the policy optimized with ML^3 is able to swing up and balance, whereas the iLQR trajectory struggles to keep the pendulum upright after swinging up the pendulum, and oscillates around the vertical configuration. In the following, we continue with the model-free evaluation. In Fig. 5, we show the performance of our framework using two continuous control tasks based on OpenAI Gym MuJoCo environments (Gym, 2019): ReacherGoal and AntGoal. The ReacherGoal environment is a 2-link 2D manipulator that has to reach a specified goal location with its end-effector. The task distribution consists of initial random link configurations and random goal locations. The performance metric for this environment is the mean trajectory sum of negative distances to the goal, averaged over 10 tasks. 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. 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,



(a) ReacherGoal environment

(b) AntGoal environment

Figure 7. Optimization curves for policies with different number of layers that are optimized with the same meta-loss pre-trained on a 2-layer policy. Each curve is an average over ten different tasks.

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.

4.2.2. Sparse Rewards and Self-Supervision

By providing additional reward information during meta-train time, as pointed out in Section 3.2, 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-loss network can learn a loss function that provides exploratory strategies to the agent or allows the agent to learn in a self-supervised setting. In Fig. 6, we show results from 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. To solve the task, the car has to accumulate energy by repeatedly climbing the hill forth and back. In this environment, 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 a not available during the meta-test time. The meta-loss network incorporates this behavior into its loss leading to an improved exploration during the meta-test time as can be seen in Fig. 6a. Fig. 6b 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. The meta-loss network can also be trained in a fully self-supervised fashion, by removing the task related input g (i.e. rewards) from the meta-loss input. We successfully apply this setting in our experiments with the continuous control MuJoCo environments: the ReacherGoal and the AntGoal (see Fig. 5). In both cases, during meta-train time, the meta-loss network is still optimized using the rewards provided by the environments. However, during meta-test time, no external reward signal is provided and the meta-loss calculates the loss signal for the policy based solely on its environment state input.

4.2.3. Generalization across different model architectures

One key advantage of learning the loss function is its re-usability across different policy architectures that is impossible for the frameworks aiming to meta-train the policy directly (Finn et al., 2017; Duan et al., 2016). 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. 7a and Fig. 7b 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.

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. We observed significant speed improvements in benchmark reinforcement learning tasks on a variety of environments. Furthermore, we showed that by introducing additional guiding rewards during training time we can train our meta-loss to develop exploratory strategies that can significantly improve performance during the meta-test time, even in sparse reward and self-supervised settings. Finally, we presented experiments that demonstrated that the learned meta-loss transfers well to unseen model architectures and therefore can be applied to new policy classes. 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

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

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

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

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

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

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

Gym, O., 2019.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

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

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

Koenemann, J., Del Prete, A., Tassa, Y., Todorov, E., Stasse, O., Bennewitz, M., and Mansard, N. Whole-body model-predictive control applied to the hrp-2 humanoid. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3346–3351. IEEE, 2015.

Levine, S. and Koltun, V. Guided policy search. In *International Conference on Machine Learning*, pp. 1–9, 2013.

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

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

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

Metz, L., Maheswaranathan, N., Cheung, B., and Sohl-Dickstein, J. Learning unsupervised learning rules. In *International Conference on Learning Representations*, 2019.

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

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

Parascandolo, G., Huttunen, H., Xiang, T., Hospedales, T., and Virtanen, T. Taming the waves: sine as activation function in deep neural networks. *Submitted to ICLR*, 2017.

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

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

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

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

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

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

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

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

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