ulm university universität
uulm

Faculty of Engineering, Computer Science and Psychology
Institute of Neural Information Processing

Deep Reinforcement Learning: Model-Agnostic Meta-Learning

Project Report at Ulm University

Presented by:
Carolin Schindler
carolin.schindler@uni-ulm.de

Instructors:
Prof. Dr. Dr. Daniel Alexander Braun
M. Sc. Heinke Hihn

winter term 2021/2022

Last updated February 26, 2022

Satz: PDF-LaTeX 2$_\varepsilon$

Abstract

In the (deep) reinforcement learning setting, an agent learns a behavioural policy by acting upon an initially unknown environment. By this means, the agent becomes able to perform a specific task. However, it is effortful and time-consuming to train an agent on every potential task from scratch. A possible approach to this problem is the paradigm of meta-learning, where a “meta-policy” is learned that can adapt to new tasks quickly with only a few samples. In this replication study, we focus on the Model-Agnostic Meta-Learning (MAML) algorithm and its first-order approximation by Finn et al. [9] in the 2D Navigation environment. MAML aims to optimize the parameters of the policy such that they are a good starting point for learning a new task rapidly. Overall we were able to replicate the results, albeit not in their entirety. We reason that this is due to MAML being sensitive to the selected hyperparameters and thus prone to instabilities.

Contents

1 Introduction

While humans are efficient, general-purpose learners, most current artificial intelligence systems rely on a huge amount of data and hardware resources in order to learn very specific tasks [12]. This poses artificial intelligence inflexible compared to human intelligence. To address this issue, Griffiths et al. [12] suggest to combine meta-reasoning [6] and meta-learning [30] approaches. The former deals with "reasoning about reasoning", which helps to enhance efficient, real-time usage of limited computational resources. The latter aims at adapting to new tasks quickly with only a few samples, i. e. it is concerned with "learning to learn".

A prominent example of meta-learning is the Model-Agnostic Meta-Learning (MAML) algorithm by Finn et al. [9]. Its basic idea is to optimize the model parameters during training in such a way that they are a good starting point for learning a new task with only a few steps of gradient descent, i. e. the parameters of the trained model allow for rapid progress towards the new task. On the one hand, MAML is a simple algorithm that is applicable to any gradient-based learning setup and yields strong performance [14]. On the other hand, however, it leads to expenses regarding runtime and memory. Moreover, it is known that training with MAML can be tricky as it is prone to instabilities due to its sensitivity to neural network architectures and hyperparameters [2].

In this replication study, we focus on the Model-Agnostic Meta-Learning algorithm in the domain of deep reinforcement learning [17]. This domain is challenged with solving the gap between human and artificial intelligence just like any other domain [31]. Furthermore, deep reinforcement learning itself faces difficulties in terms of reproducibility [13]. Hence, we are interested in investigating to which extend the results of the MAML paper are reproducible for a simple reinforcement learning agent.

All in all, our goal is

- to implement the MAML algorithm (and the 2D Navigation environment) as true to the original work [9] as possible and
- to replicate the results achieved by MAML in comparison to regular pretraining for the 2D navigation task [9, Figure 4].
- Additionally, we compare the performance of MAML to its first-order approximation FOMAML for the 2D navigation task as Finn et al. [9] did only provide results for this approximation in the domain of supervised classification.

The remainder of this work is structured as follows: In the next Chapter, we give some background information in the area of Deep Reinforcement Learning and Meta-Learning before we introduce the algorithm under investigation, namely Model-Agnostic Meta-Learning. Chapter 3 explains the setup of the experiments conducted during the replication study and evaluates their results, which are further discussed in Chapter 4. We conclude in Chapter 5 with a short summary of our findings.

2 Methods

In the following, we detail the Model-Agnostic Meta-Learning [9] algorithm focusing on the domain of (deep) reinforcement learning. In order to have a common understanding of the concepts relevant for this work, we first provide a short introduction into the areas of Deep Reinforcement Learning and Meta-Learning.

2.1 Background

2.1.1 Deep Reinforcement Learning

In the Reinforcement Learning (RL) [28, 17] setting, an agent sequentially performs actions a_t upon an initially unknown environment and gets feedback from it in terms of transitions to states s_{t+1} and rewards r_{t+1}. The goal of the agent is to learn a behavioural policy $\pi(a_t|s_t)$ mapping states to actions such that it maximizes the expected total future reward, e. g. the expected finite horizon discounted cumulative reward $J(\pi) = \mathbb{E}_\pi\big[\sum_{t=1}^{H} \gamma^t r_t\big]$ with discount factor $\gamma \in [0, 1)$ and horizon H.

There exist several RL algorithms to solve this learning problem in a model-free manner. They can be roughly grouped into two categories: value-based and policy-based methods.
The former aim to estimate the value function $V^\pi(s)$ or $Q^\pi(s, a)$ respectively, which indicates how much reward we can expect to get in the future when we are in a certain state and continue acting according to π. We then choose the action that most likely leads to a transition into states with high value function estimates, i. e. the value function is used to derive the policy. Well-known examples are Temporal-Difference Learning [27] (for $V^\pi(s)$) and Q-Learning [32] (for $Q^\pi(s, a)$).
In contrast, policy-based methods estimate the policy directly. A prominent approach

is to assume a parametrized policy $\pi_\theta(a_t|s_t)$ and apply policy gradient methods in order to find the best parameter θ^*. The most basic update equation originates from the REINFORCE algorithm [34] alias Vanilla Policy Gradient (VPG):

$$\theta \leftarrow \theta + \alpha \mathbb{E}_{\pi_\theta} \left[\sum_{t=1}^{H} \big(\mathcal{R}_t - b(s_t)\big) \nabla_\theta \log \pi_\theta(a_t|s_t) \right], \tag{2.1}$$

where α is the learning rate, $\mathcal{R}_t = \sum_{t'=t}^{H} r_{t'+1}$ is the reward-to-go and $b(s_t)$ is a baseline reward introduced in order to reduce the variance of the gradient estimates. A common choice for the baseline reward is $b(s_t) = V^{\pi_\theta}(s_t)$. The expectation can be implemented by Monte Carlo approximation sampling K trajectories $(s_1, a_1, r_2, ..., s_H, a_H, r_{H+1})$. More recent algorithms include Trust Region Policy Optimization (TRPO) [23] and Proximal Policy Optimization (PPO) [25], for instance. Both introduce a constraint on the size of the policy update in terms of performance in order to guarantee monotonic improvement during the optimization procedure. Further, it is noteworthy that policy-gradients as presented here are on-policy methods, meaning that we have to sample trajectories applying the policy and hence the actual actions executed by the agent.
An extension of policy gradient is the actor-critic architecture, where the actor updates its policy based on the value function estimated by the critic. By applying the Policy Gradient Theorem [29] to Equation (2.1) we obtain

$$\theta \leftarrow \theta + \alpha \mathbb{E}_{\pi_\theta} \left[\sum_{t=1}^{H} \big(Q^{\pi_\theta}(s_t, a_t) - b(s_t)\big) \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

as the update equation for the actor, where the state-action value function $Q^{\pi_\theta}(s_t, a_t)$ is approximated by the critic. When selecting $b(s_t) = V^{\pi_\theta}(s_t)$, the critic can directly estimate the advantage function $A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$, for example with the method of Generalized Advantage Estimation (GAE) [24].

Deep Reinforcement Learning [17] utilizes deep neural networks to represent required components of RL, e. g. the value function and the policy. These representations are learned by updating the parameter θ of the network f via gradient-descent with the respective loss $\mathcal{L}$. To exemplify, $\mathcal{L} = -J(\pi) = -J(f_\theta)$ when maximizing the expected total future reward. One of the simplest deep networks are fully-connected feed-forward networks, also called multi-layer perceptrons. We

will not dive deeper into the area of Deep Learning in this overview and refer to Zhang et al. [36] and Goodfellow et al. [11] for more details.

2.1.2 Meta-Learning

The most distinctive property of Meta-Learning [14] compared to related fields, such as transfer learning and multi-task learning, is its meta-objective. This objective explicitly takes the distribution over multiple tasks $p(\mathcal{T})$ into account and tries to optimize for fast learning of a single task $\mathcal{T}_i$ later on. In the domain of reinforcement learning, tasks may vary in terms of their goal definition or even the environment in which the goal needs to be achieved [9]. During meta-training we sample tasks $\mathcal{T}_j \sim p(\mathcal{T})$ and use these collectively to update our model based on the meta-objective. In the meta-testing phase, we train the model in a regular fashion for single tasks $\mathcal{T}_i$ which are normally others than the ones deployed for meta-training. With this evaluation setup, one can verify if the adaptation to a new task is possible with only a few samples and hence meta-learning was attained.

The approaches in the field of meta-learning can be grouped into three main categories [14]: Metric-based techniques assume an instance-based learning model containing a kernel and hence try to meta-learn this kernel, i. e. they learn a feature space in which the similarity between two inputs can be represented well across different tasks. Model-based techniques require an adaptive, internal state which learns to represent relevant task-specific information. As the representation is updated sequentially, a memory component is needed in order not to forget previous information. The hidden internal representation is then used for making predictions, which is why this technique is also known as black-box. Optimization-based techniques adjust the optimization algorithm itself in order to achieve fast learning. Commonly, this is formulated as a bi-level optimization problem: The base-learner applies a regular optimization strategy for each single task $\mathcal{T}_j$ and is nested within a meta-learner that optimizes for the meta-objective over the complete task distribution $p(\mathcal{T})$.

2.2 Model-Agnostic Meta-Learning

Model-Agnostic Meta-Learning (MAML) [9] belongs to the class of optimization-based meta-learning techniques (for more details, see Section 2.1.2). In general, MAML is applicable to any model and any learning task. The only assumptions made by the algorithm are that the model f is parametrized by some θ and that the parameters can be learned with gradient-based learning techniques, i. e. the loss function needs to be smooth enough in θ. This – in the light of neural networks – non-restrictive set of assumptions leads to a wide field of application for MAML.

The meta-objective to be optimized is defined by MAML as follows:

$$\min_{\theta} \sum_{\mathcal{T}_j \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_k^j)}(f_{\theta_k^j}) \,, \tag{2.2}$$

where θ_k^j is the parameter updated by the base-learner for task $\mathcal{T}_j$ and $\mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_k^j)}$ is the loss for task $\mathcal{T}_j$ evaluated with the data $\mathcal{D}_k^j$. It is important to note that the minimization is performed with respect to the model parameter θ, whereas the loss is evaluated with the parameter θ_k^j. Hence, the meta-objective explicitly searches for a parameter θ such that the updated parameters θ_k^j can be gained fast for all tasks $\mathcal{T}_j \sim p(\mathcal{T})$, which implies that θ is task-sensitive.

Algorithm 1 sketches the practical implementation of MAML in the reinforcement learning setting. In general, the base-learner can perform any amount $k \geq 1$ of parameter updates. If we set $k = 0$, there are no updates by the base-learner and hence we would not perform MAML any more but regular training with batching over several tasks. Further, we highlight that it is mandatory to sample new trajectories $\mathcal{D}_i^j$ for each parameter update with the updated reinforcement learner $f_{\theta_i^j}$ when deploying on-policy methods. A reuse of previously sampled trajectories would cause the agent to perform different actions as if its current policy had been applied and thus would violate the notion of being on-policy. The specific formulation of the loss function $\mathcal{L}$ depends on the choice of the reinforcement learning algorithm (for more details, see Section 2.1.1).

Compared to regular training on multiple tasks without a meta-objective, there is a memory and computational overhead. The additional memory consumption arises from the need to initially copy the current model f_θ for each sampled task $\mathcal{T}_j$ and to

Algorithm 1 MAML for Reinforcement Learning (practitioner perspective)
based on Finn et al. [9, Algorithm 3]

Require:
$p(\mathcal{T})$: distribution over tasks
α, β: learning rates (α: inner leaning rate, β meta learning rate)
K, M: batch sizes (K: sample batch size, M: meta batch size)
k: amount of inner gradient steps / base-learner updates to be performed ($k \geq 1$)
$iterations$ and H: amount of iterations to train and horizon
$\mathcal{L}$: loss function
f_θ: reinforcement learner parametrized with θ

randomly initialize θ
for iteration $\in [0, iterations)$ **do**
  Sample batch of M tasks $\mathcal{T}_j \sim p(\mathcal{T})$
  for all $\mathcal{T}_j$ **do**
    Copy f_θ into $f_{\theta_0^j}$
    for $i \in [0, k)$ **do**
      Sample K trajectories $\mathcal{D}_i^j = \{(\mathbf{s}_1, \mathbf{a}_1, ..., \mathbf{s}_H, \mathbf{a}_H)\}$ using $f_{\theta_i^j}$ in $\mathcal{T}_j$
      Evaluate $\nabla_{\theta_i^j} \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_i^j)}(f_{\theta_i^j})$ using $\mathcal{D}_i^j$
      Update $\theta_{i+1}^j := \theta_i^j \leftarrow \theta_i^j - \alpha \nabla_{\theta_i^j} \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_i^j)}(f_{\theta_i^j})$ ▷ base-learner update
    end for
    Sample K trajectories $\mathcal{D}_k^j = \{(\mathbf{s}_1, \mathbf{a}_1, ..., \mathbf{s}_H, \mathbf{a}_H)\}$ using $f_{\theta_k^j}$ in $\mathcal{T}_j$
    Evaluate and Store $\nabla_\theta \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_k^j)}(f_{\theta_k^j})$ using $\mathcal{D}_k^j$
  end for
  Update $\theta \leftarrow \theta - \beta \frac{1}{M} \sum_{\mathcal{T}_j} \nabla_\theta \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_k^j)}(f_{\theta_k^j})$ with stored $\nabla_\theta \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_k^j)}(f_{\theta_k^j})$ ▷ meta-learner update
end for

store the calculated gradients before they get applied. Moreover, the calculation of the gradient for the meta-learner update with the meta-objective from Equation (2.2) contains second-order derivatives with respect to different parameters θ_i^j, which is expensive in terms of computation:

$$\begin{aligned}
\nabla_\theta \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_k^j)}(f_{\theta_k^j}) &= \nabla_{\theta_k^j} \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_k^j)}(f_{\theta_k^j}) \cdot \left(\nabla_{\theta_{k-1}^j} \theta_k^j\right) \dots \left(\nabla_{\theta_0^j} \theta_1^j\right) \cdot \left(\nabla_\theta \theta_0^j\right) \\
&= \nabla_{\theta_k^j} \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_k^j)}(f_{\theta_k^j}) \cdot \left(\prod_{i=0}^{k-1} \nabla_{\theta_i^j} \theta_{i+1}^j \right) \cdot I \\
&= \nabla_{\theta_k^j} \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_k^j)}(f_{\theta_k^j}) \cdot \prod_{i=0}^{k-1} \nabla_{\theta_i^j} \left(\theta_i^j - \alpha \nabla_{\theta_i^j} \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_i^j)}(f_{\theta_i^j})\right) \\
&= \nabla_{\theta_k^j} \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_k^j)}(f_{\theta_k^j}) \cdot \prod_{i=0}^{k-1} \left(I - \alpha \nabla_{\theta_i^j} \left(\nabla_{\theta_i^j} \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_i^j)}(f_{\theta_i^j})\right) \right)
\end{aligned} \tag{2.3}$$

In the context of the above Calculation (2.3) [33], we point out that if $k > 1$, one would in practice store and directly accumulate the second-order derivatives after each base-learner update in Algorithm (1) and then only multiply $\nabla_{\theta_k^j} \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_k^j)}(f_{\theta_k^j})$ in line 12.

To avoid this computational effort, the authors of the MAML paper [9] suggested to apply a first-order approximation of the gradient, which leads to the so-called FOMAML (first-order MAML) algorithm. This means, we apply the same steps as in the MAML procedure but approximate the meta-gradient with

$$\nabla_{\theta} \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_k^j)}(f_{\theta_k^j}) \approx \nabla_{\theta_k^j} \mathcal{L}_{\mathcal{T}_j}^{(\mathcal{D}_k^j)}(f_{\theta_k^j}) \,. \tag{2.4}$$

3 Experiments and Evaluation

3.1 Experimental Apparatus

Here, we detail our experimental setup for replicating the results of the MAML paper [9] in the domain of deep reinforcement learning. All experiments were conducted in the 2D Navigation environment. With the chosen meta-learning setup, we try to stay as close as possible to the experimental evaluation as described in the original work. However, there are minor deviations due to the underlying framework selected for implementation, that are named in the respective section below. Moreover, not all hyperparameters are given, which is why we applied proposed default values or tuned them by ourselves in case of learning rates.

3.1.1 2D Navigation Environment

In this environment [9], a point agent moves in a 2D coordinate system which is not bounded. At every time step, the agent observes its current x- and y-coordinates as a state and performs actions in terms of movement in the x- and y-direction restricted to $[-0.1, 0.1]^2$. We normalize the action space to the interval $[-1, 1]$ for each dimension. This means, when the environment receives an action, it first downscales and then clips it to the allowed range. The goal of the agent is to move from the point of origin to a goal position that is sampled randomly for each task within a unit square, i. e. $p(\mathcal{T}(x)) = p(\mathcal{T}(y)) = U(-0.5, 0.5)$. Thus, the reward to be maximized is the negative euclidean distance to the goal. The goal is viewed as reached when the agent deviates less than 0.01 from it in both directions.
The oracle version of this environment additionally informs the agent about the aimed goal position in each state.

We implemented the 2D Navigation environments as a new custom environment in the OpenAI Gym [5]. Thereby, we used the implementation[1] by Finn et al. [9] as a reference.

3.1.2 Meta-Learning Setup

The setup employed for the reinforcement learning experiments in the 2D Navigation environment in the original MAML paper [9] is noted down in Table 3.1. Further, the authors pointed out that "the model with the best average return during training was used for evaluation" [9]. Moreover, we miss some information regarding the hyperparameters of the RL algorithms, for instance the value of the discount factor γ and the values of the tuned learning rates, the amount of tasks used for the evaluation and the seeds applied to the random number generators. Finn et al. [9] made their implementation publicly available[1]. However, some of the parameters given in their exemplary experiment scripts (such as the inner learning rate α, the learning rate for evaluation and the sample batch size K for evaluation) deviate from the ones stated in their paper. Thus, it is questionable to what extend we can assume the parameters in the implementation to be the ones applied for the results presented in the paper. Due to this, we make use of default values for the parameters in the RL algorithms and manually tune the learning rates as described in Appendix A.

We based our implementation according to Algorithm 1 upon the PyTorch version of the Spinning Up [1] framework. The details of the applied setup are specified in Table 3.2. All reinforcement learning algorithms in Spinning Up deploy an actor-critic architecture with the GAE method. Therefore, we perform equivalent update steps for the critic as we do for the actor. The actor network additionally has a $\tanh$ activation in the output layer in order to directly map into the allowed range of the activity space and hence avoid clipping of actions by the environment. Unfortunately, TRPO is only available in the Tensorflow version of the framework. Therefore, we applied VPG to all learner updates. We were not able to train the meta-learner with stochastic gradient descent due to either vanishing or exploding gradients. This is why we utilized an Adam Optimizer [16] (with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$) and

[1] `https://github.com/cbfinn/maml_rl/`

Table 3.1: The experimental setup of the original MAML paper [9] for evaluating their algorithm in the 2D Navigation environment.

MAML (second-order only)	
policy f_θ	neural network two hidden layers of size 100 $ReLU$ nonlinearities
loss function $\mathcal{L}$	VPG for base-learner update and for evaluation TRPO for meta-learner update
baseline reward $b(s_t)$	standard linear feature baseline [7]
horizon H	100
$iterations$	up to 500 for training, up to 4 for evaluation
amount of base-learner updates k	1 for training, 0 for evaluation
meta batch size M	20 for training, 1 for evaluation
sample batch size K	20 for training, 40 for evaluation
inner learning rate α	0.1
learning rate for evaluation	0.1 for the first update, then 0.05
Baseline Methods (pretrained, random, oracle)	
learning rates	manually tuned

on recommendation of Antoniou et al. [2], Cosine Annealing [19], which reduces the learning rate over the whole training or evaluation process respectively.

The baseline methods we compare the MAML algorithm to are the same as in the original work [9]: *random* and *oracle* are the lower and upper boundary for the evaluation, where *random* is evaluated with a randomly initialized (with the seed for training) actor-critic network and the *oracle* is a model trained in a regular fashion over the different tasks on the oracle version of the 2D Navigation environment. The third and most interesting baseline method in terms of comparison to MAML is *pretrained*, which is trained on the same 2D Navigation environment as MAML but does not perform any base-learner update steps, i. e. *pretrained* performs regular training on several different tasks without a meta-objective.
In order to make sure that the effects in the experiments actually arise from the different algorithms themselves, we utilize the same hyperparameter setting for all methods as we are not guaranteed to find the best setup for each of them individually.

Table 3.2: Our experimental setup for replicating the results of the original MAML paper [9] in the 2D Navigation environment.

MAML (second-order as well as first-order approximation)	
critic v_ω and actor f_θ	separate neural networks two hidden layers of size 100 $ReLU$ nonlinearities actor with additional $\tanh$ output activation
loss function $\mathcal{L}(f_\theta)$	VPG (for any update)
loss function $\mathcal{L}(v_\omega)$	unconstrained non-linear regression problem (as in GAE [24])
baseline reward $b(s_t)$	GAE is applied with critic
horizon H	100
$iterations$	100 for training (as in official implementation), 3 for evaluation
discount factor γ	0.99 (proposed default value)
λ for GAE	0.97 (proposed default value)
amount of base-learner updates k	1 for training, 0 for evaluation
meta batch size M	20 for training, 1 for evaluation
sample batch size K	20 for training, 40 for evaluation
amount of tasks for evaluation	40 (as in official implementation)
learning rates	manually tuned $\rightarrow$ specified in Appendix A base-learner: Stochastic Gradient Descent meta-learner: Adam Optimizer with Cosine Annealing
seed for training	0
seed for evaluation	1 (such that: evaluation tasks $\not\subseteq$ training tasks)
Baseline Methods (pretrained, random, oracle)	
we apply the same setup as for MAML, only the amount of base-learner updates k is set to 0 for training, as well	

3.1.3 Supplementary Experiments

As the results of the original MAML paper [9] were not entirely reproducible with the above meta-learning setup, we conducted further experiments always changing a single parameter in order to identify possible reasons. To find out how stable the methods are to different random number initializations, we changed the seed for the training from 0 to 2. We also increased the meta batch size M to 40, since the official implementation points out that this setting is more stable. Further, we suspect the simple VPG algorithm that we applied to all learner updates to be a factor as the original work applied the more advanced TRPO algorithm for the meta-learner update. As mentioned earlier, TRPO is not available in the employed framework. Thus, we experimented with PPO for the base-learner update (with the proposed default values $\epsilon = 0.2$ for clipping and $d_{targ} = 0.01$ for the penalty). We cannot apply PPO to the meta-learner update easily as PPO operates on the loss and not on the gradient like TRPO.

3.2 Results

We aim to replicate the quantitative results achieved in the 2D Navigation environment by the original MAML paper [9]. These are shown in Figure 3.1, for simpler comparison. To this end, we train and evaluate the three baseline methods and MAML as described in Section 3.1.2 and plot the average return per iteration during the evaluation phase. Our results are presented in Figure 3.2 and Table 3.3.

Initially, we had assumed that the original plot depicted the mean and the standard deviation over the tasks utilized for the evaluation since there was no further information on what the plot actually shows. With hindsight, however, we came to the conclusion that it most likely shows the 95% confidence interval due to two reasons: First, our plots with the standard deviation in Figure 3.2a have much wider error regions than the original plot, whereas the plots with the 95% confidence interval in Figure 3.2b more closely resemble the ones in the original work. Second, Finn et al. [9] provided the 95% confidence interval information for their few-shot classification task and it is not too far-fetched to assume that the same information was provided in the reinforcement learning task.

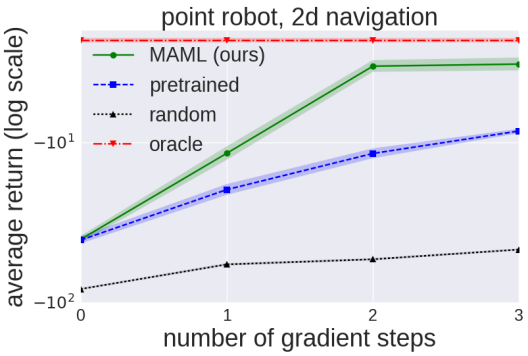
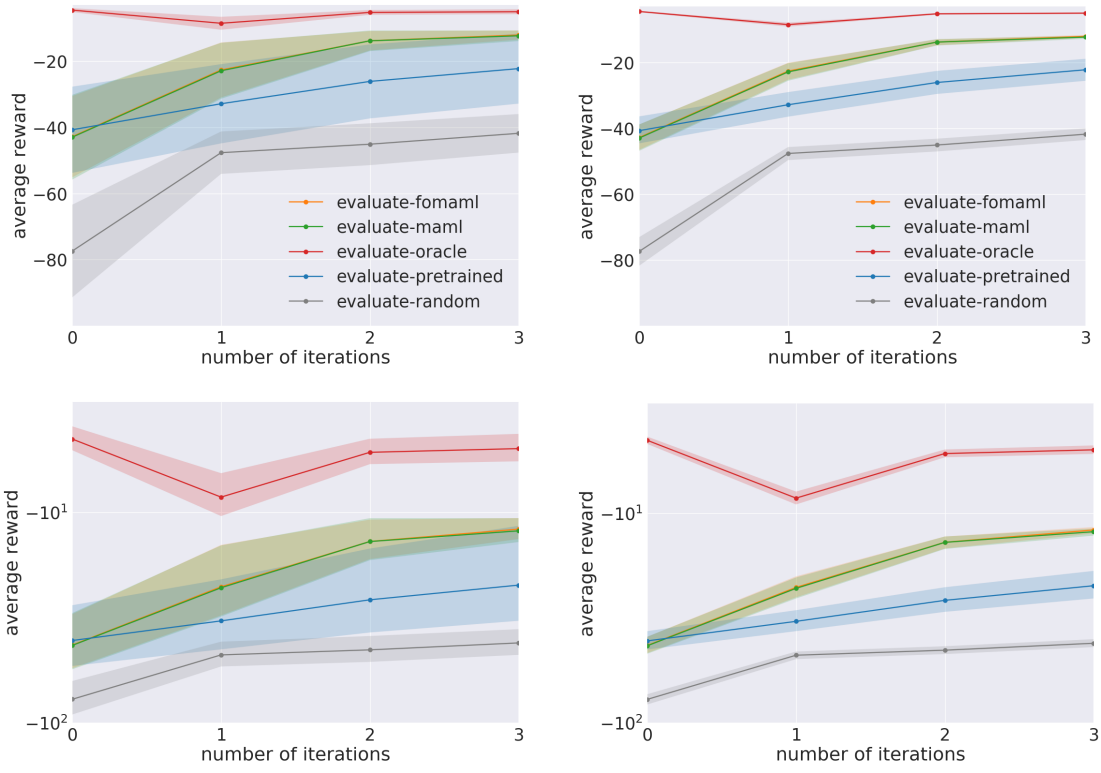


Figure 3.1: Quantitative results of the original MAML paper in the 2D Navigation environment.
see Finn et al. [9, Figure 4]



(a) Mean and standard deviation.

(b) Mean and 95% confidence interval.

Figure 3.2: Results achieved with our meta-learning setup in the 2D Navigation environment.
top: linear scale, bottom: same plot with logarithmic scale

Table 3.3: Mean and standard deviation of the average return achieved in the evaluation phase for all methods in the 2D Navigation environment.

results of our meta-learning setup								
iteration	0		1		2		3	
oracle	-4.49	± 0.58	-8.48	± 1.59	-5.19	± 0.72	-4.99	± 0.74
FOMAML	-42.79	± 12.29	-22.61	± 8.31	-13.77	± 2.93	-12.03	± 1.37
MAML	-42.95	± 12.81	-22.85	± 8.47	-13.79	± 3.11	-12.26	± 1.59
pretrained	-40.70	± 13.04	-32.84	± 11.98	-26.08	± 11.19	-22.20	± 10.59
random	-77.40	± 14.01	-47.64	± 6.40	-45.08	± 6.32	-41.79	± 5.82
results of the original MAML paper [9] (measured from the plot, except for MAML)								
iteration	0		1		2		3	
oracle	≈ -2.50		≈ -2.50		≈ -2.50		≈ -2.50	
MAML	-40.41		-11.68		-3.33		-3.32	
pretrained	≈ -40.00						≈ -9.00	
random	≈ -80.00						≈ -40.00	

Table 3.4: Two-sided independent samples t-test [4] for comparing our evaluation results of the MAML and pretrained method.

iteration	0	1	2	3
	$t(39)=0.78$ $p=0.44$	$t(39)=4.31$ $p=0.0001$	$t(39)=6.69$ $p=0.00000006$	$t(39)=5.87$ $p=0.0000008$

All methods improve during the evaluation phase, except the *oracle* baseline in our results which for unknown reasons slightly decreases in performance. The original work did only present exact numbers for the mean average reward per iteration in the case of MAML. By visual estimation and measuring from the plot as noted in Table 3.3, we see that the curve of the *random* baseline is quite close to the original plot and the *oracle* method is around 2.5 points worse in terms of mean average reward. Without any task-specific tuning, MAML and *pretrained* achieve approximately the same mean average reward as in the original plot. However, the two methods do not increase their average reward as much during the evaluation as in the original work and there is still room for improvement compared to our *oracle* baseline. Yet, MAML performs significantly better than the *pretrained* method after only one step of gradient descent (see Table 3.4 for t-test values).

As mentioned earlier, we performed supplementary experiments in order to identify possible reasons for the lower increase in performance during the evaluation for the MAML and the *pretrained* method. None of our proposed changes to the meta-learning setup was able to increase the average reward. Hence, we include the results of these experiments in Appendix B. Nevertheless, there is a noteworthy

result when changing the seed for training from 0 to 2: Contrary to our previous results, MAML does not perform significantly different from the *pretrained* method, while the first-order approximation FOMAML outperforms MAML. These findings, among others, are discussed in more detail in the next Chapter.

All in all, our results in the meta-learning setup support the hypotheses formulated by Finn et al. [9], namely:

- MAML enables fast learning of new tasks.
  (Although we should be careful not to over-generalize this statement in the light of the supplementary experiments, where a change of the training seed led to MAML not performing significantly different from the *pretrained* method.)
- MAML is applicable for meta-learning in the domain of (deep) reinforcement learning.
- A model trained with MAML continues to improve during the evaluation.

Furthermore, our evaluation shows that FOMAML is performing at least as well as MAML.

4 Discussion

The replication of the (deep) reinforcement learning experiments in the original MAML paper [9] turned out to be more challenging than initially expected. Especially, due to the lack of information on hyperparameters and even specifications of shown results. The official implementation, that is publicly available, was of no help in this regard as there are inconsistencies with the values noted in the paper.

Moreover, Raghu et al. [21] faced challenges[1] when running this implementation for a baseline comparison, causing them to apply a different one[2]. However, they find the latter not to achieve as good results as the original work. Even though their experimental setup is equivalent, judged by the given hyperparameters and procedure. This resembles our results, where MAML does not improve as much as in the original work during the testing phase. Raghu et al. [21] reported an average reward of -20.3 ± 3.2 for the 2D Navigation environment with the standard deviation calculated over three different random model initializations. Unfortunately, they did not report the amount of iterations performed in the meta-testing for the reinforcement learning tasks. In case they used only one iteration, our results are comparable. If they conformed with the original setup and reported the average reward after more than one iteration in the evaluation, we even outperform this replication. Just like us, the authors suspected the random initialization of the model to be a factor for this comparably diminished performance. They observed large variance in results depending on the model initialization. Our supplementary experiment where the training seed is changed from 0 to 2 on a sample basis further supports this statement. Only due to the changed randomness, we cannot conclude MAML to be performing better than the *pretrained* method anymore.
One may argue that it is not the MAML method but deep reinforcement learning

[1] The challenges faced when running the official MAML implementation for the domain of reinforcement learning are not further specified by Raghu et al. [21].
[2] `https://github.com/tristandeleu/pytorch-maml-rl`

itself that is prone to such changes [13, 15] like the seed, hyperparameters and even the underlying framework. While this might be true when comparing the results of the MAML method among different publications, it is not when comparing the individual results in our work: All methods (FOMAML, MAML, *pretrained*, *oracle* and *random*) employ the same framework, the same hyperparameters and the same seed. Thus, when comparing the results between them, all reinforcement learning algorithms are affected in a similar manner by the overall RL setup and it is reasonable to conclude that the effects arise due to the different nature of the methods themselves.
Our MAML implementation could possibly be improved by modelling the inner learning rate α as a trainable parameter as already suggested by Finn et al. [9], which reduces the amount of fixed hyperparameters. Incooperating the notion of Gradient Agreement [8] into the meta-objective, seems to be a promising extension as well. Additionally, we may experiment with increasing the amount of inner gradient steps k. Doing so, we would accept higher demands on memory and computation. Moreover, it is also not unlikely that a larger k makes the setup even more unstable [2].

Consequently, we suggest to perform more thorough experiments with the MAML method in the domain of deep reinforcement learning investigating stochasticity and hyperparameter selection. In this way, we expect to be able to provide guidelines for training RL agents in a MAML fashion successfully, similarly to the ones by Antoniou et al. [9] who verified their improvements only in the domain of supervised classification.
Furthermore, our replication study only covered the simple 2D Navigation environment, whereas usual reinforcement learning tasks are way more complex. Among them are locomotion tasks where simulated robots have to walk into a certain direction or with a predefined speed [9]. However, the variety of the tasks is still narrow compared to practical application, which is why Yu et al. [35] introduced Meta-World, a collection of environments with diverse tasks suited for benchmarking. It would be interesting to study the replicability of the results achieved with MAML in these more challenging environments, likewise in comparison to other variants of MAML as shortly discussed below.

A surprising finding is that FOMAML performs as well as MAML in the 2D Navigation environment or even better in case of setting the training seed to 2. Finn et al. [9] re-

ported equal performance for MAML and FOMAML in the supervised classification setting, while speeding up the computation with FOMAML. They reasoned that the second-order derivative terms in Equation 2.3 might be close to zero anyway as the applied $ReLU$ neural networks are locally almost linear. Hence, the update equation for MAML and the first-order approximation are nearly the same. However, this does not explain FOMAML outperforming MAML in one of our supplementary experiments. An explanation for these results in the RL domain is the following: When implementing the second-order derivation primitively (as done by us and the authors of the MAML paper), the second-order gradient terms are incorrect [10]. This is due to the expectation in the loss calculation containing the model parameter θ (see Equation 2.1 for reference). By applying the Monte Carlo approximation, this dependency of the expectation is omitted and leads to wrong higher-order gradient calculations. An in depth mathematical derivation is provided in Appendix C. Accordingly, MAML implemented primitively for RL is only an approximation as well and in case of the changed training seed, the FOMAML approximation performed better than the MAML one.
The issue of higher-order derivatives can be tackled by applying the DiCE [10] method to the gradient calculation. However, this leads to the drawback of high variance in the gradients. Taming MAML [18] solves the latter by additionally introducing a proper meta baseline reward $b(s_t, \mathcal{T}_i)$. For a more complete and detailed overview of different methods that arose from MAML, we refer to Huisman et al. [14]. There are also different variants of first-order MAML approximations, e. g. Reptile [20] and implicit MAML [22].

As a last point, we note that during our literature research it came to our attention that advances and variants in model-agnostic meta-learning tend to be only evaluated in the domain of supervised learning. Closing this evaluation gap in the deep reinforcement learning domain, is another opportunity for future work.

5 Conclusion

In the herein presented work, we replicated the results achieved by Model-Agnostic Meta-Learning [9] in the domain of deep reinforcement learning within the 2D Navigation environment. To this end, we implemented the MAML algorithm and the 2D Navigation environment as true to the original work as possible. With our meta-learning setup, we were able to confirm the hypotheses by Finn et al. [9] although the results were not reproducible in their entirety. One of the supplementary experiments, however, raised concerns in terms of the generalisability of the statement that MAML is outperforming the regularly trained *pretrained* baseline method. Therefore, we suggested to further investigate the stability of the MAML method in the context of reinforcement learning.

In addition, we provided evaluation results for the FOMAML approximation and showed that the authors' results in the domain of supervised classification also hold for the point agent in the 2D Navigation environment: FOMAML performs as well as MAML. Unexpectedly, FOMAML could even outperform MAML in a supplementary experiment. We reasoned that this is possible due to the primitive implementation of the second-order derivatives in the meta-learner update, which is likewise only an approximation in the domain of reinforcement learning.

Bibliography

[1] Joshua Achiam. *Spinning Up in Deep Reinforcement Learning*. 2018. URL: https://spinningup.openai.com/.

[2] Antreas Antoniou, Harrison Edwards, and Amos Storkey. *How to train your MAML*. 2019. arXiv: 1810.09502 [cs.LG].

[3] James Bergstra, Daniel Yamins, and David Cox. "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". In: *Proceedings of the 30th International Conference on Machine Learning*. Vol. 28. 1. PMLR, 2013, pp. 115–123. URL: https://proceedings.mlr.press/v28/bergstra13.html.

[4] Allan G. Bluman. *Elementary Statistics: A Step by Step Approach*. 7th ed. McGraw-Hill Higher Education, 2009.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[6] Michael T Cox and Anita Raja. *Metareasoning: Thinking about thinking*. MIT Press, 2011. DOI: 10.7551/mitpress/9780262014809.001.0001.

[7] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning*. Vol. 48. ICML. 2016, pp. 1329–1338.

[8] Amir Erfan Eshratifar, David Eigen, and Massoud Pedram. *Gradient Agreement as an Optimization Objective for Meta-Learning*. 2018. arXiv: 1810.08178 [cs.LG].

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. PMLR, 2017, pp. 1126–1135. URL: https://proceedings.mlr.press/v70/finn17a.html.

[10] Jakob Foerster, Gregory Farquhar, Maruan Al-Shedivat, Tim Rocktäschel, Eric P. Xing, and Shimon Whiteson. *DiCE: The Infinitely Differentiable Monte-Carlo Estimator*. 2018. arXiv: 1802.05098 [cs.LG].

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[12] Thomas L. Griffiths, Frederick Callaway, Michael B. Chang, Erin Grant, Paul M. Krueger, and Falk Lieder. “Doing more with less: meta-reasoning and meta-learning in humans and machines”. In: *Current Opinion in Behavioral Sciences* 29 (2019). Artificial Intelligence, pp. 24–30. DOI: https://doi.org/10.1016/j.cobeha.2019.01.005.

[13] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. “Deep Reinforcement Learning That Matters”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (2018). URL: https://ojs.aaai.org/index.php/AAAI/article/view/11694.

[14] Mike Huisman, Jan N. van Rijn, and Aske Plaat. “A survey of deep meta-learning”. In: *Artificial Intelligence Review* 54.6 (2021), pp. 4483–4541. DOI: 10.1007/s10462-021-10004-4.

[15] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. *Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control*. 2017. arXiv: 1708.04133 [cs.LG].

[16] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[17] Yuxi Li. *Deep Reinforcement Learning*. 2018. arXiv: 1810.06339 [cs.LG].

[18] Hao Liu, Richard Socher, and Caiming Xiong. “Taming MAML: Efficient unbiased meta-reinforcement learning”. In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. PMLR, 2019, pp. 4061–4071. URL: https://proceedings.mlr.press/v97/liu19g.html.

[19] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017. arXiv: 1608.03983 [cs.LG].

[20] Alex Nichol, Joshua Achiam, and John Schulman. *On First-Order Meta-Learning Algorithms*. 2018. arXiv: 1803.02999 [cs.LG].

[21] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. *Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML*. 2020. arXiv: 1909.09157 [cs.LG].

[22] Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. *Meta-Learning with Implicit Gradients*. 2019. arXiv: 1909.04630 [cs.LG].

[23] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. PMLR, 2015, pp. 1889–1897. URL: https://proceedings.mlr.press/v37/schulman15.html.

[24] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2018. arXiv: 1506.02438 [cs.LG].

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[26] Zebang Shen, Alejandro Ribeiro, Hamed Hassani, Hui Qian, and Chao Mi. "Hessian Aided Policy Gradient". In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. PMLR, 2019, pp. 5729–5738. URL: https://proceedings.mlr.press/v97/shen19d.html.

[27] Richard S. Sutton. "Learning to predict by the methods of temporal differences". In: *Machine Learning* 3.1 (1988), pp. 9–44. DOI: 10.1007/bf00115009.

[28] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. 2nd ed. Adaptive computation and machine learning. The MIT Press, 2018. ISBN: 978-0-2620-3924-6.

[29] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[30] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 1998. ISBN: 978-0-7923-8047-4.

[31] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. *Learning to reinforcement learn*. 2017. arXiv: 1611.05763 [cs.LG].

[32] Christopher John Cornish Hellaby Watkins. "Learning from Delayed Rewards". PhD thesis. Cambridge, UK: King's College, 1989. URL: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

[33] Lilian Weng. "Meta-Learning: Learning to Learn Fast". In: *lilianweng.github.io/lil-log* (2018). URL: http://lilianweng.github.io/lil-log/2018/11/29/meta-learning.html.

[34] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3-4 (1992), pp. 229–256. DOI: 10.1007/bf00992696.

[35] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. *Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning*. 2021. arXiv: 1910.10897 [cs.LG].

[36] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. "Dive into Deep Learning". In: *arXiv preprint* (2021). arXiv: 2106.11342 [cs.LG].

A Manual Tuning of Learning Rates

We tuned all learning rates manually utilizing the hyperopt [3] library. In the following, we describe the tuning setting. If not otherwise indicated, the parameters are applied as stated in Section 3.1.2. The resulting learning rates for the actor and critic network are listed in Table A.1.

In order to have a more feasible search space, we constrain each pair of learning rates lr_{actor} and lr_{citic} such that the actor learning rate is smaller or equal to the respective critic learning rate; more precisely $lr_{actor} = c \cdot lr_{critic}$, where $c \in [0.5, 1]$. The meta learning rate β and the evaluation learning rate for the critic are searched within $[e^{-10}, e^{-2}]$, whereas the inner learning rate α is searched over $[e^{-8}, 1]$. A larger α is preferred as otherwise the base-learner would be quite similar to the meta-learner when we update the latter, especially as we perform only $k = 1$ base-learner updates.
We first tune β based on the training of the *pretrained* baseline method. With the gained β, we then tune α by training with the MAML method. In both cases, we set the number of $iterations$ to 10 and the meta batch size M to 10 so as to reduce the computational effort. The objective to be maximized is the mean of the average reward over all iterations during training. The average reward of each iteration is computed by evaluating the current model with the tasks that are sampled for the next model update. For the evaluation learning rate, we evaluate a model trained

Table A.1: Manually tuned learning rates for the actor and critic network rounded to four decimal places.

	Actor	**Critic**
meta learning rate β	0.0028	0.0039
inner learning rate α	0.0990	0.1200
evaluation learning rate	0.0029	0.0036

with MAML (as it did not have a choice for its learning rate β) applying the previously found learning rates α and β. We aim to maximize the average final reward over the sampled evaluation tasks. Here, final reward refers to the average reward achieved in the last iteration of the evaluation. Due to memory constraints, we computed the objective over the first 20 evaluation tasks.

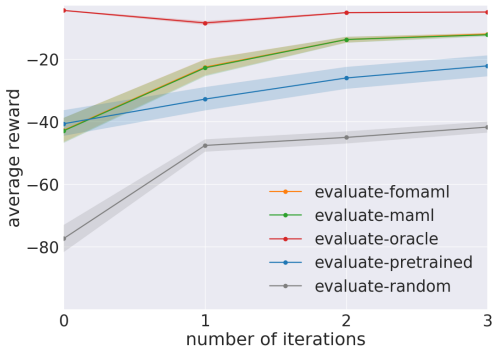
B Results of Supplementary Experiments

In Figure B.1, we present the results of our supplementary experiments, whose setup is described in Section 3.1.3. With these experiments, we searched for factors that potentially cause the performance of the MAML and *pretrained* method in our meta-learning setup to not increase as much as in the original work by Finn et al. [9].

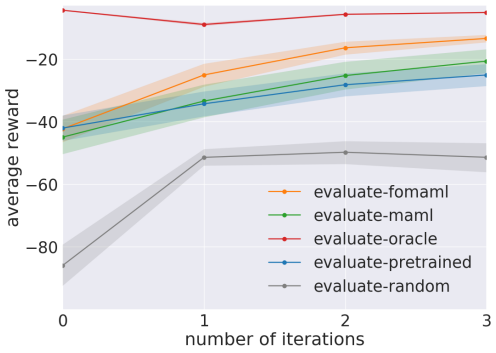
Replacing the simple VPG reinforcement learning algorithm in the base-learner with PPO, did not improve the performance of MAML and FOMAML as can be seen in Figure B.1d. Even after repeating the tuning steps for the inner learning rate α and the evaluation learning rate for this new setup, there is no change. Same applies for increasing the meta batch size M: The training is indeed more stable for all methods, however, there is no difference visible in the evaluation compared to our meta-learning setup. In our experiments, the only change affecting the evaluation is the seed for training. It is visibly noticeable that the average reward of the *random* baseline method and the MAML method suffer from this change. In fact, MAML is not significantly different from *pretrained* any more (see Table B.1 for t-test values). However, the performance of FOMAML is still well such that we can find a significant difference to the other methods after one step of gradient descent. Perhaps, MAML would recover if we tuned the learning rates for the new training seed again, nevertheless we leave this to future work.

Table B.1: Two-sided independent samples t-test [4] for comparing our evaluation results of the FOMAML, MAML and pretrained method with training seed 2.

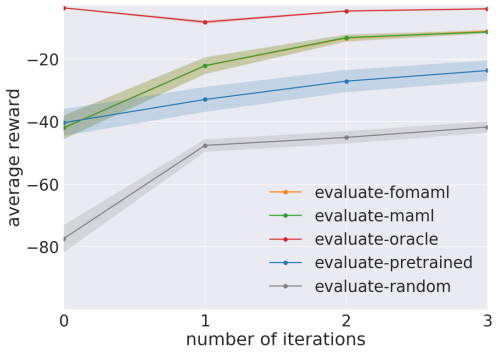
mean and standard deviation of the average return				
iteration	0	1	2	3
FOMAML	-42.37 ± 13.46	-25.16 ± 11.86	-16.48 ± 6.66	-13.49 ± 3.86
MAML	-45.01 ± 17.77	-33.50 ± 16.86	-25.36 ± 14.37	-20.74 ± 12.47
pretrained	-42.13 ± 13.49	-34.37 ± 13.15	-28.29 ± 12.11	-25.17 ± 11.36
comparing MAML and FOMAML				
iteration	0	1	2	3
	$t(39)$=0.75 p=0.46	$t(39)$=2.56 p=0.014	$t(39)$=3.55 p=0.0010	$t(39)$=3.51 p=0.0011
comparing MAML and pretrained				
iteration	0	1	2	3
	$t(39)$=0.82 p=0.42	$t(39)$=0.26 p=0.80	$t(39)$=0.99 p=0.33	$t(39)$=1.66 p=0.10



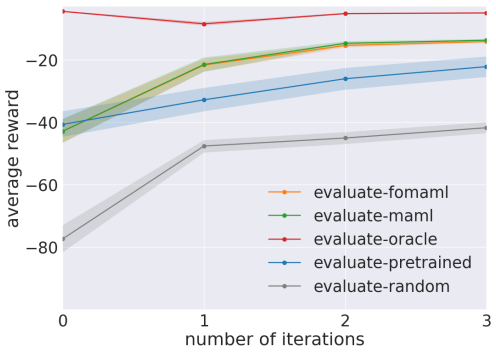
(a) Our meta-learning setup.



(b) Changed training seed (to 2).



(c) Increased meta batch size ($M = 40$).



(d) Applied PPO to the base-learner update.

Figure B.1: Results of the supplementary experiments in the 2D Navigation environment. Each plot shows the mean along with the 95% confidence interval.

C Calculation of Second-order Derivative

In the following, we mathematically derive why the primitive implementation of the second-order derivation leads to a wrong gradient calculation. We exemplary demonstrate this for the REINFORCE [34] update Equation (2.1) with $b(s_t) = 0$. A generalization to other update equations applying the Monte Carlo approximation is straight forward.

The equation below derives the correct second-order derivative using the expectation to represent the first-order derivative [10, 18, 26]:

$$
\begin{aligned}
\nabla_\theta \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{H} \mathcal{R}_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right] &= \nabla_\theta \frac{1}{K} \sum_{k=1}^{K} \pi_\theta(a_t^k|s_t^k)\left[\sum_{t=1}^{H} \mathcal{R}_t^k \nabla_\theta \log \pi_\theta(a_t^k|s_t^k)\right] \\
&= \tfrac{1}{K}\textstyle\sum_{k=1}^{K}\left[\sum_{t=1}^{H} \mathcal{R}_t^k \nabla_\theta \log \pi_\theta(a_t^k|s_t^k)\right] \nabla_\theta \pi_\theta(a_t^k|s_t^k)^\top \\
&\quad + \tfrac{1}{K}\textstyle\sum_{k=1}^{K}\left[\nabla_\theta \sum_{t=1}^{H} \mathcal{R}_t^k \nabla_\theta \log \pi_\theta(a_t^k|s_t^k)\right] \pi_\theta(a_t^k|s_t^k)^\top \\
&= \tfrac{1}{K}\textstyle\sum_{k=1}^{K}\left[\sum_{t=1}^{H} \mathcal{R}_t^k \nabla_\theta \log \pi_\theta(a_t^k|s_t^k)\right] \nabla_\theta \log \pi_\theta(a_t^k|s_t^k)^\top \pi_\theta(a_t^k|s_t^k)^\top \\
&\quad + \tfrac{1}{K}\textstyle\sum_{k=1}^{K}\left[\sum_{t=1}^{H} \mathcal{R}_t^k \nabla_\theta^2 \log \pi_\theta(a_t^k|s_t^k)\right] \pi_\theta(a_t^k|s_t^k)^\top \\
&= \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{H} \mathcal{R}_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \nabla_\theta \log \pi_\theta(a_t|s_t)^\top \\
&\quad + \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{H} \mathcal{R}_t \nabla_\theta^2 \log \pi_\theta(a_t|s_t)\right]
\end{aligned}
\tag{C.1a}
$$

When we apply Monte Carlo approximation to the resulting second-order derivative in the above equation, we obtain:

$$\begin{aligned} &\approx \frac{1}{K}\sum_{k=1}^{K}\left[\sum_{t=1}^{H}\mathcal{R}_t^k\nabla_\theta\log\pi_\theta(a_t^k|s_t^k)\right]\nabla_\theta\log\pi_\theta(a_t^k|s_t^k)^\top \\ &\qquad + \frac{1}{K}\sum_{k=1}^{K}\left[\sum_{t=1}^{H}\mathcal{R}_t^k\nabla_\theta^2\log\pi_\theta(a_t^k|s_t^k)\right] \end{aligned} \tag{C.1b}$$

If we instead replace the expectation already in the first-order derivative by a Monte Carlo approximation, the second-order derivative is calculated as follows:

$$\begin{aligned} \nabla_\theta\mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{H}\mathcal{R}_t\nabla_\theta\log\pi_\theta(a_t|s_t)\right] &\approx \nabla_\theta\frac{1}{K}\sum_{k=1}^{K}\left[\sum_{t=1}^{H}\mathcal{R}_t^k\nabla_\theta\log\pi_\theta(a_t^k|s_t^k)\right] \\ &= \frac{1}{K}\sum_{k=1}^{K}\left[\sum_{t=1}^{H}\mathcal{R}_t^k\nabla_\theta^2\log\pi_\theta(a_t^k|s_t^k)\right] \end{aligned} \tag{C.2}$$

Comparing the correct deviation in Equation (C.1b) with the one in Equation (C.2), we see that the term

$$\frac{1}{K}\sum_{k=1}^{K}\left[\sum_{t=1}^{H}\mathcal{R}_t^k\nabla_\theta\log\pi_\theta(a_t^k|s_t^k)\right]\nabla_\theta\log\pi_\theta(a_t^k|s_t^k)^\top$$

is missing in the latter one. Thus, the primitive implementation of the second-order derivative leads to a biased gradient estimate [18].