On the Robustness of Context- and Gradient-based Meta-Reinforcement Learning Algorithms

Patrik Okanovic
pokanovic@ethz.ch

Rafael Sterzinger
rsterzinger@ethz.ch

Fatjon Zogaj
fzogaj@ethz.ch

Abstract

Traditional reinforcement learning (RL) methods have shown recent breakthroughs in a variety of fields, where they are able to utilize large amounts of data and computational power. Meta-RL has emerged with the goal of learning a model that is able to quickly adapt to new tasks, leading to more efficient training and transfer-learning. Current research has mostly neglected the robustness of these algorithms when training and test data differ. We perform an extensive study on various models and environments to offer new insights for the problem of covariate shift in RL. Our results among others include experiments on various meta-batch sizes, prohibiting dense sampling and testing on out-of-distribution tasks. We show that when testing data is even slightly different and out-of-distribution, performance decreases substantially. This can however be alleviated by performing enough gradient steps.

1 Introduction

In order to tackle the problem of needing large amounts of data to learn a particular task, the widespread usage of meta-learning has emerged. In the domain of reinforcement learning (RL), meta-learning aims to capture the common dynamics of a group of similar tasks to later on utilize this knowledge to quickly adapt to new tasks. By doing so, algorithms are capable of reaching state-of-the-art performance during testing while using only a small number of training samples or gradient steps which makes them highly efficient. In general, there exist two different approaches to meta-RL: context-based and gradient-based methods. In context-based methods the idea is to aggregate and summarize experience through latent representations, which policies then are conditioned on. In contrast to this, gradient-based methods learn from aggregated experience using policy gradients, meta-learned loss functions, or hyperparameters.

In this work, we aim to provide an overview of recent context- and gradient-based meta-RL algorithms where we analyze, benchmark, and compare their performance on different predefined environments with varying complexity. Here, we mainly place our focus on the availability and variety of tasks for a given environment during training as data efficiency is one of the main goals of meta-learning. In order to illustrate this, we conduct an ablation study on the robustness of our selected algorithms when varying the amount of available tasks or withholding some of them during training completely. Forthgoing, robustness refers to test results stability in regard to changing inputs and parameters of the model, e.g. restricting sampling from an environment (input) or adjusting the meta-batch size (parameter). We provide new insights in regard to out-of-distribution performance, the extent to which these algorithms depend on dense sampling of tasks, and the robustness within each algorithm when provided with fewer tasks per training iteration.

Foundations of Reinforcement Learning HS2021, Department of Computer Science, ETH Zürich.

2 Related Work

For context-based meta-RL algorithms, we decided on RL^2 due to its popularity as one of the foundational papers in this realm [1]. The same holds for MAML in the domain of gradient-based methods [2]. A more recent work to this poses the algorithm by Rakelly et al. named PEARL [3]. Despite the fact that these three algorithms pursue two different approaches, they have one thing in common: all three of them assume the ability to collect environment interactions online during training/testing. As opposed to this, we also consider MACAW which is a recent algorithm purely operating in the offline meta-RL setting. Although it might seem difficult to compare this algorithm to online variants, we include it to emphasize the effects of having restricted amounts of data which is key in our analysis and ablation study.

While deciding on which algorithms we are going to explore in this paper, we noticed a lack of ablation studies on robustness in the sense of stability of tests results, when it comes to perturbations in the input (environment) or the parameters (model). Solely two out of the four proposed algorithms conducted an ablation study on robustness in miniscule form. For instance, Rakelly et al. investigated the performance of PEARL as a function of the number of meta-training samples [3]. On the other hand, Mitchell et al. compared their proposed algorithm, MACAW, when varying the amount of meta-training tasks at disposal [4]. Given this fact, a more broad and exhaustive comparison between the selected algorithms in regard to robustness seems necessary as scarcity of samples/tasks plays a key factor when applying these algorithms to real world scenarios. Furthermore, we hypothesise that another, in all four papers overlooked, but very important measurement is the out-of-distribution performance of these algorithms making the need of such a study ever more so crucial.

We think, that such an analysis is of great importance, given the fact that data/covariate shifts are a prominent research topic in other subdomains of machine learning and that out-of-distribution generalization is among the hardest problems in machine learning [5, 6, 7]. To enable generalization to unseen domains without without prior knowledge of any data, researchers have made significant progress in the past decade at developing a broad spectrum of methodologies [8]. Existing works can be categorized into methods based on domain-invariant representation learning [5] and meta-learning [9] to name a few.

3 Methods/Algorithms

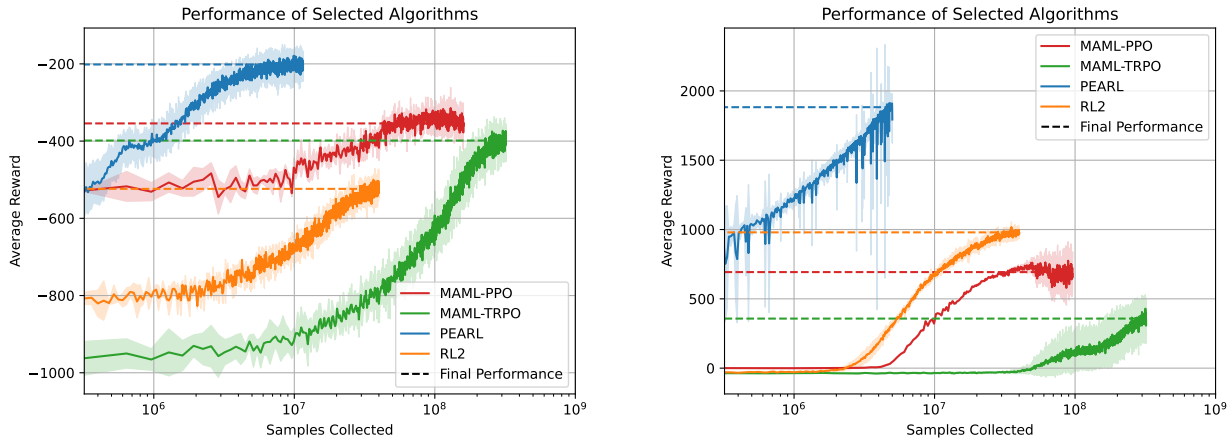
We provide a short overview of the different methods for readers already acquainted with the field of meta-RL. For a more thorough explanation we point the reader to the Appendix A where the algorithms and its procedures are described in detail.

RL^2: The RL^2 algorithm is encoded in the weights of an RNN, which are learned through a general-purpose "slow" RL algorithm. The RNN receives all information a typical RL algorithm would and retains its state across episodes. The activations of the RNN store the state of the “fast” RL algorithm on the current previously unseen MDP [1].

MAML: MAML is a model-agnostic meta-learning algorithm compatible with any model trained using gradient descent. It is applicable to a variety of different learning problems, including classification, regression, and RL [2]. It is comprised of two loops where, in the inner-loop, the policy is updated for each task which is then used to collect trajectories. Then, in the outer-loop, the original policy parameters are updated using learned policies from the inner-loop and the collected trajectories.

PEARL: PEARL performs online probabilistic filtering of latent task variables to infer how to solve new tasks using only small amounts of experience. This probabilistic interpretation makes posterior sampling possible and makes the exploration efficient. PEARL demonstrates how to integrate these task variables with off-policy RL algorithms to achieve meta-training and adaptation efficiency compared to other algorithms we examine [3].

MACAW: The offline-variant MACAW uses Meta-Actor Critic with Advantage Weighting. Inspired by MAML, it also consists of an inner- and outer-loop for training. Adaptation is performed for the value function using bootstrap-free updates via supervised regression onto Monte-Carlo returns. The policy is updated in a similar fashion with actions weighted by the estimated advantage. It imitates the general approach of pre-training on an existing dataset which allows it to fine-tune with only a small amount of data to a new task [4].



(a) Ant-Dir Environment

(b) Half-Cheetah-Dir Environment

Figure 1: Average reward for the Ant and Cheetah-Direction environment for MAML, RL^2, and PEARL. PEARL is able to achieve much higher end performance while at the same time requiring up to ~ 100 times less sample steps than the second-best to converge. [3]

4 Experiments

As pointed out already in Section 1, the focus of our experiments is laid on the availability and variety of tasks during training which has mostly been neglected in literature. The main idea behind this is to show the out-of-distribution performance and the resulting need for more robust models for practical applications which suffer from training data and covariate shift [10, 11, 12].

Building upon previous work related to this topic, we are going to extend the ablation study of Rakelly et al. [3] by not only considering the performance of algorithms as a function of the number of meta-training samples but also as a number of meta-training tasks. In other words, we are going to prohibit the algorithms from sampling randomly from a given task distribution and allow them to only sample from a predefined set of tasks. By varying the size of this predefined set, we will be able to illustrate the algorithms' ability to generalize well while disallowing dense sampling of the whole space of tasks. For this and most of the following experiments, we found the Ant-Direction environment from the MuJoCo [13] (also see Section 4.1) physics engine best suited as it allows us to easily uniformly sample a certain number of directions which we then keep fixed during training. Testing will then be performed after each training iteration on a newly drawn set of directions to illustrate generalizability.

A related experiment to this has already been conducted by Mitchell et al. [4]. However, they solely considered the setting of offline meta-RL. Given this fact, we want to see for MACAW if comparable performance conclusions can be drawn for less complex environments. These experiments analyze if a too small (< 5) or a too large (> 20) amount of tasks/offline buffers hurts performance. Here we are going to consider a Bernoulli-Bandit environment.

Furthermore, during some initial tests of our implementations, we found that the meta-batch size, i.e. the amount of tasks drawn during each meta-training iteration, can play an important role in regard to the performance during testing. For instance, if the meta-batch size was too small, MAML needed a few more gradient updates to obtain comparable results. We assume that this is due to the problem that if the meta-batch size is too small, the task distribution space is not covered well enough by the drawn samples. As an example of this, assume that we have only five 10-armed Bernoulli-Bandit tasks. If it now happens that in three out of those five, the best lever would be the seventh one, the model would get biased towards this arm after the meta-update, making deviations from that harder.

Based on this witnessed phenomenon, we designed two experiments. For the first one, we are going to analyse the robustness within our selected algorithms by varying the meta-batch size parameter and its effect on performance. For the second one, we are going to take a more drastic approach and remove some parts of the task distribution completely. In the Ant-Direction environment, this would mean, for instance, that only north-east direction tasks, i.e. from $0°$ to $90°$, are available during

training. By then testing on all the other directions, i.e. from 90° to 360°, we are able to measure the out-of-distribution performance of our selected algorithms in a precise manner which further would hint at their ability to handle covariate shifts.

Lastly, assuming that performance will be dampened, if our selected algorithms are only trained on a subspace of the initial task distribution, we consider a final experiment on MAML to see how many gradient steps are necessary to reach comparable performance to a model trained on the complete task distribution (if possible).

In summary, our designed experiments answer the following questions:

- Q1) How well do our selected online meta-reinforcement learning algorithms generalize when prohibiting dense sampling?
- Q2) How robust are the selected online meta-reinforcement learning algorithms if the meta-batch size is varied?
- Q3) How good do our selected algorithms perform in regard to out-of-distribution performance?
- Q4) How quickly can MAML obtain good performance on out-of-distribution tasks?
- Q5) How is the performance of the offline meta-reinforcement learning algorithm MACAW impacted when varying the amount of different tasks in less complex environments?

4.1 Environments

MuJoCo is a physics simulator for evaluating agents on continuous motor control tasks with contact dynamics [13]. Of this, we use two MuJoCo continuous environments, the Half-Cheetah-Direction and Ant-Direction. In those robotic locomotion tasks the goal is to teach a two-legged and a four-legged creature, respectively, to walk towards a certain direction/reach a specific goal. Furthermore, for our less complex environment, we use a discrete 10-armed Bernoulli-Bandit environment [1], which can be seen as a set of real distributions $B = \{R_1, ..., R_K\}$. Each distribution being associated with the rewards delivered by one of the K (in our case 10) levers.

4.2 Evaluation

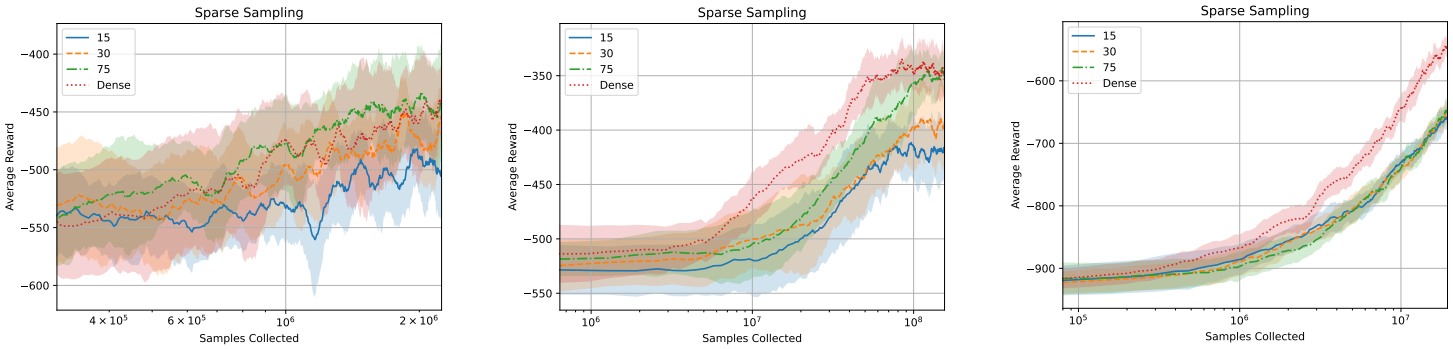
After experimenting with a variety of different implementations and libraries, we have conducted our experiments based primarily on the works of Rakelly et al. [3], Finn et al. [2], Rothfuss et al. [14], and Mitchell et al. [4].

The performance of our selected online algorithms in a general overview is depicted in Figure 1. Here, we plot the average reward for two environments, Ant-Direction and Half-Cheetah-Direction, in relation to the amount of environment steps that were in total taken during training. In both cases PEARL obtains the best performance and illustrates faster improvements during training. Furthermore, we see that there is a large discrepancy between the performance of the PPO and the TRPO variants of MAML. For this reason, we proceeded with our experiments only considering the PPO variant.

As pointed out by Rakelly et al., PEARL is a very sample efficient algorithm converging before the two MAML variants as well as RL^2 even start improving. Moreover, even when these algorithms converge after around $\sim 10^7 - 10^8$ steps, their obtained average reward is still below the performance of PEARL, showcasing its superiority. However, this comes at the cost of training duration, as PEARL takes five times longer to finish even though less samples are collected.

We conduct all our experiments utilizing the default configurations (see Appendix A) used by the authors of the corresponding algorithms in order to ensure best performance. This means that for our selected algorithms different amounts of environment steps are taken during training, resulting in different lengths of the samples collected axis. Given that our experiments are more concerned about behaviour/trends, absolute values play a less important role in the analysis.

Furthermore, based on the fact that our experiments are building upon multiple implementations and have different underlying environments/reward signals, absolute results between plots are not always directly comparable. This is the case for results reported for MACAW which has to be considered separately due to its offline nature. Additionally, for MACAW we report the amount of training iterations instead of samples collected. Another differentiation has to be made for the comparison



(a) PEARL (b) MAML-PPO (c) RL^2

Figure 2: An illustration of the performance of our selected online algorithms when trained with sparse sampling, i.e. using only a fixed number of tasks in the Ant-Direction environment. Results indicate that PEARL is not impacted by sparse sampling. For MAML a larger number of tasks seem to be beneficial whereas RL^2 is not able to learn in the sparse setting as good as in the regular one.

between the general MAML-PPO results and the experiment discussed in Section 5.4, as here the implementation and default configuration differ.

Regarding the design of the upcoming figures, we report a 95% confidence interval calculated over three different runs. Furthermore, to increase readability we perform smoothing by plotting moving averages and using log-scales. Furthermore, all of the experiments, if not stated otherwise, were conducted on the Ant-Direction environment.

Lastly, to give an impression on the computation power required, the final training and evaluation of all these models took in total two weeks of compute time on Euler [15] utilizing up to 1.300 cores and 30 GPUs. It is for this reason some experiments were only conducted on some of the models and environments as running an even more large-scale evaluation would be infeasible with the available time and computational resources.

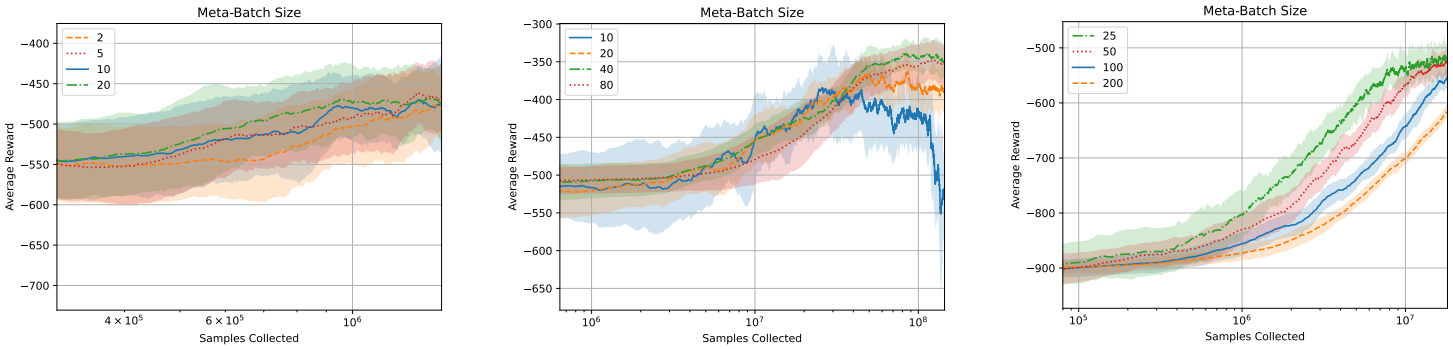
5 Results and Discussion

In the following section we present the results from our experiments in order to answer our posed questions from Section 4. For almost all of our experiments the Ant-Direction environment was used for evaluation except for 5.5 where we tested on a Bernoulli-Bandit environment. Figures and corresponding discussions aim to illustrate key differences of our selected meta-RL algorithms in regard to out-of-distribution performance, the extent to which these algorithms depend on dense sampling of tasks, and the robustness within each algorithm when provided with fewer tasks per training iteration. We emphasize that different implementations and environment configurations at times were used for different plots and as such absolute inter-plot comparisons cannot be made.

5.1 How well do our selected online meta-reinforcement learning algorithms generalize when prohibiting dense sampling?

In the following experiment we inspect how algorithms behave when restricted to sparse sampling during training, i.e. only allowing for an approximation of the underlying task distribution. As opposed to this, dense sampling can be thought of as covering the whole task distribution thoroughly [4]. Our experiments evaluate the performance of algorithms when trained on a fixed amount of initially sampled tasks instead of sampling new tasks in each iteration.

Looking at Figure 2, we observe that PEARL is more robust to sparse sampling since the performance does not differ significantly when fixing various amounts of tasks, given its high variance. As expected, the worst performance is obtained for all algorithms in the sparsest setting, having only 15 initial tasks at disposal. Another observation we made it that for MAML the reward increases with the number of training tasks. However, results of PEARL and MAML generally show large variance which indicate that while sampling dense does improve performance, after a certain threshold (30 -



(a) PEARL (b) MAML-PPO (c) RL^2

Figure 3: An illustration of the performance of our selected online algorithms when varying the meta-batch size in the Ant-Direction environment. We find that PEARL is the most unaffected by variations of the meta-batch size and that a smaller meta-batch size is benefiting RL^2 due to higher stochasticity.

75 fixed tasks) the improvements are not significant anymore. On the other hand, RL^2 does not show a difference when varying the number of tasks, improving only when conducting dense sampling.

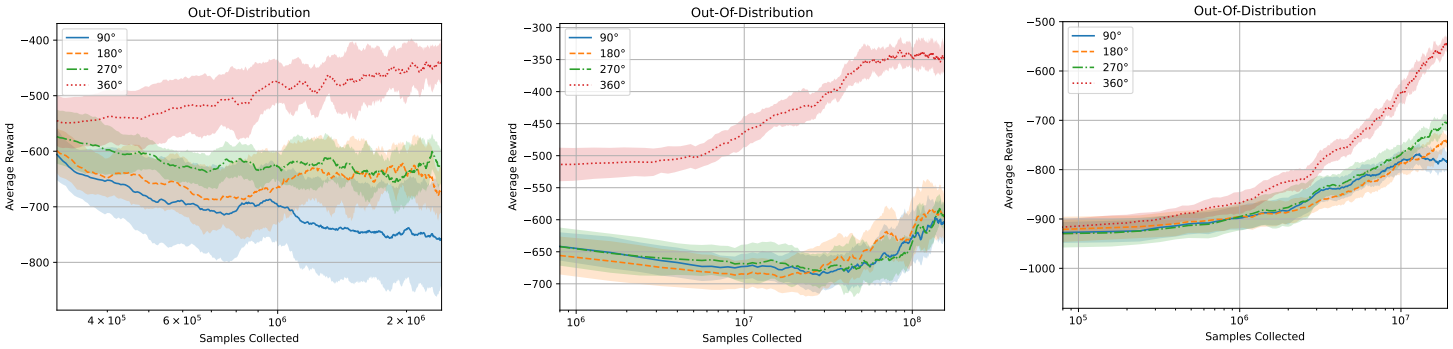
From this we conclude that prohibiting dense sampling does not pose a major problem for PEARL and MAML. For RL^2, however, scarcity of initial tasks compared to dense sampling seems to directly impact performance. Given this observation, we believe that this is mainly due to RL^2 being a context-based meta-RL approach which aims to summarize experience in its hidden state, hindering better generalization.

5.2 How robust are the selected online meta-reinforcement learning algorithms, if the meta-batch size is varied?

In this experiment, we inspect the robustness within our selected online algorithms when varying the meta-batch size, i.e. the number of tasks sampled per each training iteration. Starting from the default configuration for the meta-batch size (see Appendix A), we evaluate halving, quartering and doubling this initial value. Looking at Figure 3, we observe that PEARL is the most robust, i.e. its performance does not change significantly varying the meta-batch size. MAML shows a similar trend, except when quartering the meta-batch size where performance suddenly starts plummeting. We reason that this is caused by poor meta-parameter updates in the outer-loop which are biased due to the small size. On the other hand, for RL^2, we observed that higher average rewards are associated with a smaller meta-batch size. In fig. 3c, smaller meta-batch sizes have higher variance and perform better compared to larger batch sizes with less variance. Hence, in the case for RL^2, it seems that higher variance facilitates the exploration of the solution space. In regard to this phenomena, we draw an analogy to stochastic gradient descent where stochasticity in the optimization procedure can be beneficial, e.g. for escaping local minima and finding better trajectories.

5.3 How good do our selected algorithms perform in regard to out-of-distribution performance?

In the next experiment, we looked at the out-of-distribution performance of our selected algorithms, a performance measurement often overlooked in literature. The results of our selected algorithms are depicted in Figure 4 (PEARL, MAML, RL^2) and in Figure 5a (MACAW). When analyzing these figures, we find that there are some similarities between PEARL and MACAW. Both of them are unable to obtain large improvements through training if at all. Moreover, when trained only on one quarter of the full task distribution, performance seems to even decrease. For the other two algorithms, MAML with PPO and RL^2, this is not the case. Although the performance of all four selected algorithms is drastically impacted when evaluated in out-of-distribution settings, MAML and RL^2 are still able to improve. Overall, it seems that MAML and its offline variant, MACAW, are the most impacted by out-of-distribution training, showing the largest difference between 360°



(a) PEARL (b) MAML-PPO (c) RL^2

Figure 4: An illustration of the out-of-distribution performance of various online algorithms in the Ant-Direction environment. PEARL is unable to obtain better performance when trained on out-of-distribution. Opposed to this, MAML and RL^2 are still capable of achieving some improvements. However, all three are heavily impacted by not observing the whole task distribution.

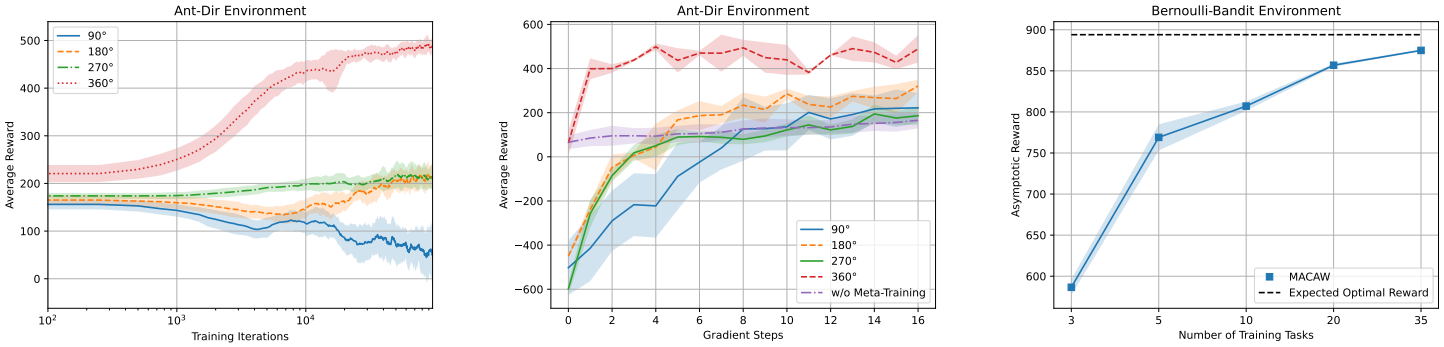
and 270°. Lastly, regarding the variance of the different algorithms, PEARL seems to vary the most within the different evaluations, a trend which was already made out in the other experiments.

5.4 How quickly can MAML obtain good performance on out-of-distribution tasks?

For our next experiment we build upon the findings of Section 5.3, that is all our selected algorithms are heavily dependent on the training distribution and are unable to perform well if training and testing data differ (see Figure 4 and 5a). Here we train our model for 500 iterations, then fix 40 out-of-distribution task in testing time for averaged results and perform multiple adaptation steps. This is comparable to fine tuning to see how quickly MAML will learn a good behaviour policy on unseen tasks drawn from out-of-distribution. We observe in Figure 5b that after around 13 gradient steps all MAML instances outperform the algorithm without meta-training/any training at all. This can be explained by the fact that the model still seems to learn dynamics of the environment even if the training data is out-of-distribution. For instance, even though the model in the Ant-Direction environment might not be able to experience all possible directions, it will still obtain a basic understanding of locomotion. With enough training updates it is able to use/restructure that information to infer on the out-of-distribution test task. Furthermore, we observe that training on half of the data distribution is able to achieve good result quicker than the algorithm trained on three quarters of data distribution. A possible explanation for this could be that it is easier for the model to simply mirror what it has learned than figure out how to extensively use its attained information.

5.5 How is the performance of the offline meta-reinforcement learning algorithm MACAW impacted when varying the amount of different tasks in less complex environments?

This experiment is concerned with the question on whether similar conclusions, as drawn by Mitchell et al. [4], can be made for MACAW when evaluating its performance in a less complex environment such as the Bernoulli-Bandit environment with varying amounts of initial training tasks. This environment differs a lot in complexity when compared to the Cheetah-Velocity environment which MACAW was originally evaluated on, with the main difference being that the action and state space are discrete instead of continuous. The results of this experiment are depicted in Figure 5c, illustrating the average asymptotic reward of MACAW when trained on a certain amount of fixed training tasks. In more detail, the training tasks refer to different, randomly drawn 10-armed Bernoulli-Bandit tasks. After training MACAW for 5.000 gradient updates, its performance was then evaluated for 1.000 lever pulls in 20 test tasks. In this Figure, an interesting trend can be made out. As the amount of training tasks increases, the better its performance will be, ultimately converging to its maximum possible obtainable reward. Furthermore, the variance in performance overall gets reduced as well. These findings are in stark contrast to the observations reported by Mitchell et al., where they found that in the Cheetah-Velocity environment there exists a sweet-spot and that too many tasks, i.e. 35, will actually hurt performance.



(a) (b) (c)

Figure 5: **(a)** An illustration of MACAW's out-of-distribution performance in the Ant-Direction environment. MACAW is quite impacted by out-of-distribution training and is only able to obtain marginally better performance than in the beginning with 270° and 180°. For 90°, meta-training even seems to be detrimental. **(b)** An illustration of the performance of MAML after a certain amount of gradient steps showcasing how quickly it can adapt to out-of-distribution tasks in the Ant-Direction environment. MAML takes around 13 gradient steps for all of its out-of-distribution instances to outperform an untrained network. In the case, however, where MAML was only trained on half of the the underlying task distribution, it still seems that meta-training can be beneficial in the long term. **(c)** The performance of MACAW when varying the amount of different tasks in a less complex Bernoulli-Bandit environment. As opposed to the findings by Mitchell et al. [4] for a complex environment, we find that an increasing amount of training tasks is beneficial and reduces variance.

6 Conclusion and Future Work

In this work we have analyzed the robustness of various meta-RL algorithms (PEARL, MAML, RL^2, MACAW) through multiple extensive experiments. This is an important metric for practical application which has mostly been neglected in literature so far.

To see if the models are able to adapt to related unseen data, we evaluate performance when the distribution of the testing data differs from the training data. All models show a significant performance decrease, which is improved only for MAML and RL^2 when training for longer. MACAW here only shows slight improvements at the end for two settings (180°, 270°). When performing multiple gradient updates, we show that MAML can utilize out-of-distribution information and get better scores compared to without any meta-training, even though it performs worse in the beginning. Nevertheless it is not able to achieve the same score as when training and testing data come from the same distribution.

We vary the meta-batch size to analyze if the amounts of tasks sampled per iteration (and as such training variety) has an effect on performance. For PEARL performance stays consistent with a rather large variance for all meta-batch sizes. In the same vein, MAML only shows a larger deviation towards the end when the performance for the smallest meta-batch size of 10 drops significantly. For RL^2 an increasing meta-batch size seems to directly correlate with variance, but negatively correlates with performance which shows similarities to stochastic gradient descent.

Building upon this we also consider the more drastic setting where training tasks are initially sampled and then are selected throughout training, imitating sparse sampling. PEARL shows no effect from sparse sampling while for MAML a larger amount of sampled tasks performs as good as the dense sampling setting. RL^2 is not able to reach its dense sampling score for any amount, in fact the amount does not make a difference.

With more time and computational resources it would be interesting to run our experiments on more model/environment combinations. Evaluating these with potentially more gradient steps could additionally show how results change when trying to squeeze out performance. Other future work lies in the unification of algorithms and environments so that more direct comparisons between can be made.

References

[1] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning. November 2016. URL http://arxiv.org/abs/1611.02779. arXiv: 1611.02779.

[2] 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*, pages 1126–1135. PMLR, July 2017. URL https://proceedings.mlr.press/v70/finn17a.html. ISSN: 2640-3498.

[3] Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables. page 10, 2019.

[4] Eric Mitchell, Rafael Rafailov, Xue Bin Peng, Sergey Levine, and Chelsea Finn. Offline Meta-Reinforcement Learning with Advantage Weighting. page 12, 2021.

[5] Krikamol Muandet, David Balduzzi, and Bernhard Schölkopf. Domain generalization via invariant feature representation, 2013.

[6] Gilles Blanchard, Gyemin Lee, and Clayton Scott. Generalizing from several related classification tasks to a new unlabeled sample. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper/2011/file/b571ecea16a9824023ee1af16897a582-Paper.pdf.

[7] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization, 2020.

[8] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. Domain generalization in vision: A survey, 2021.

[9] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M. Hospedales. Learning to generalize: Meta-learning for domain generalization, 2017.

[10] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua V Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *arXiv preprint arXiv:1906.02530*, 2019.

[11] Joaquin Quiñonero-Candela, Masashi Sugiyama, Neil D Lawrence, and Anton Schwaighofer. *Dataset shift in machine learning*. Mit Press, 2009.

[12] Masashi Sugiyama, Taiji Suzuki, Shinichi Nakajima, Hisashi Kashima, Paul von Bünau, and Motoaki Kawanabe. Direct importance estimation for covariate shift adaptation. *Annals of the Institute of Statistical Mathematics*, 60(4):699–746, 2008.

[13] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109. URL https://doi.org/10.1109/IROS.2012.6386109.

[14] Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. Promp: Proximal meta-policy search. *CoRR*, abs/1810.06784, 2018. URL http://arxiv.org/abs/1810.06784.

[15] Euler - ScientificComputing. URL https://scicomp.ethz.ch/wiki/Euler.

[16] 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. January 2017. URL http://arxiv.org/abs/1611.05763. arXiv: 1611.05763.

[17] Hao Liu, Richard Socher, and Caiming Xiong. Taming MAML: Efficient unbiased meta-reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 4061–4071. PMLR, May 2019. URL https://proceedings.mlr.press/v97/liu19g.html. ISSN: 2640-3498.

[18] Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning. February 2020. URL http://arxiv.org/abs/1910.08348. arXiv: 1910.08348.

[19] Ron Dorfman, Idan Shenfeld, and Aviv Tamar. Offline Meta Learning of Exploration. February 2021. URL http://arxiv.org/abs/2008.02598. arXiv: 2008.02598.

[20] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on PPO and TRPO. *CoRR*, abs/2005.12729, 2020. URL https://arxiv.org/abs/2005.12729.

[21] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization. *arXiv:1502.05477 [cs]*, April 2017. URL http://arxiv.org/abs/1502.05477. arXiv: 1502.05477.

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, August 2017. URL http://arxiv.org/abs/1707.06347. arXiv: 1707.06347.

[23] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

A Appendix

Methods/Algorithms

RL^2

In RL^2 [1] the problem of requiring a lot of data is alleviated through modelling of 1) a recurrent neural network (RNN) to act as the "fast" RL algorithm and 2) a "slow" general RL algorithm like TRPO [21] or PPO [22]. However, PPO has been shown to perform preferably in practice, although performance increases may primarily be due to code-level optimizations as suggested by Engstrom et al. [20].

The "fast" RL algorithm is encoded in the weights of the RNN which are learned by the "slow" RL one, hence the name RL^2. This approach of learning the RL algorithm automatically instead of hand-crafting it has laid the foundations for many meta-RL algorithms and has back then simultaneously been researched by Wang et al. [16].

This general procedure is visualized in Figure 6 where a new trial denotes the initialization of a new Markov Decision Process (MDP) and a new episode the beginning of a new trajectory from a starting point s_0. The input for the policy consists of the newly acquired observation s_{t+1}, the just taken action a_t, the reward r_t and a termination flag d_t which gets encoded $\phi(s_{t+1}, a_t, r_t, d_t)$ and fed into the RNN. The output of the policy is the next action a_{t+1} while at the same time updating the hidden state h_{t+2} by conditioning on h_{t+1}. The goal is to maximize the expected total discounted reward per trial, which in combination with the MDPs changing across trials, leads to the agent incorporating all prior information and gradually changing its strategy.

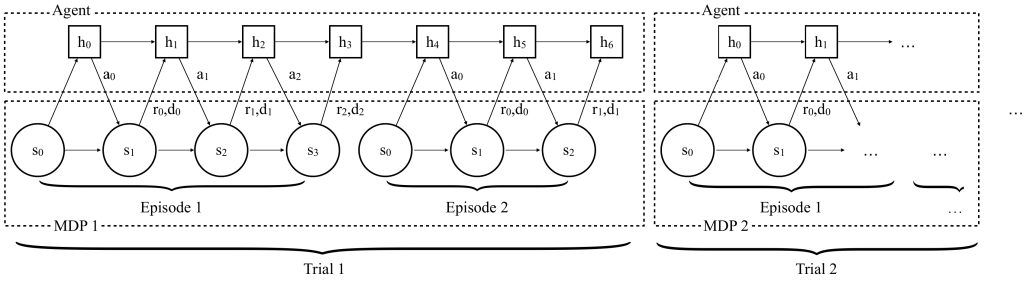


Figure 6: In RL^2 an agent acts within an environment learning the weights of its RNN to maximize the per trial reward where each trial constitutes of a separate MDP [1]

.

For a more detailed description of the algorithm and its training procedure, we advise the reader to take a look at the work by Duan et al. [1].

MAML

As the name already indicates, Model-Agnostic Meta-Learning (MAML) [2], is compatible with any model trained with gradient descent and applicable to a variety of different learning problems, including classification, regression and RL. During meta-learning, the model parametrized by θ, denoted further on as f_θ, is trained to be able to adapt to a large or infinite number of tasks in a short period of time. In the framework of RL, the goal of few-shot meta-learning is to enable an agent to quickly learn a policy for a new task by gathering only a small amount of experience in the test setting. Generally speaking, MAML can thus be thought of as training a model to be easily adaptable when provided with a new task.

For each task $\mathcal{T}_i$, drawn from some underlying task distribution, we have an initial state distribution $q_i(\mathbf{x}_1)$ and a transition distribution $q_i(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)$. Furthermore, the loss $\mathcal{L}_{\mathcal{T}_i}$ corresponds to a possibly negative reward function R. The model that we are trying to learn is f_θ, a policy that finds a mapping from states $\mathbf{x}_t$ to actions $\mathbf{a}_t$ for each time step $t \in \{1, ..., H\}$. In order to learn such a

mapping, the following optimization procedure is performed:

$$\mathcal{L}_{\mathcal{T}_i}(f_\theta) = -\mathbb{E}_{\mathbf{x}_t, \mathbf{a}_t \sim f_\theta, q_{\mathcal{T}_i}} \left[\sum_{t=1}^{H} R_i(\mathbf{x}_t, \mathbf{a}_t) \right]$$

For this loss, we then calculate the gradient in regard to each task resulting in a specific $f_{\theta_i'}$ from which we then sample new trajectories. After one iteration over all training tasks, the gradient w.r.t our original θ is calculated based on the individual loss terms of $f_{\theta_i'}$. These gradients summed up pose our update to the original parametrization. A more conclusive description of MAML can be found in Algorithm 1 or in the paper by Finn et al. [2].

Algorithm 1 MAML Meta-Training

Require: $p(\mathcal{T})$ distribution over tasks
Require: α, β step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K trajectories $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using f_θ in $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample trajectories $\mathcal{D}_i' = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using $f_{\theta_i'}$ in $\mathcal{T}_i$
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$
11: **end while**

PEARL

Previously discussed algorithms pose two major problems. First, due to their dependence on on-policy experience, their sampling efficiency is affected. Second, even though these algorithms allow for meta-training, given a new task, they are unable to reason about uncertainty when adapting to it. These challenges are addressed by Probabilistic Embeddings for Actor-Critic Meta-RL (PEARL) [3], an off-policy algorithm that aims to disentangle task inference and control.

The goal of meta-training in PEARL is to learn a context variable Z from a recent history of experience for the new task, leveraging data from a variety of training tasks. In order to adapt its behaviour to the task, the policy is conditioned on the state and the latent representation $\pi_\theta(\mathbf{a}|\mathbf{s}, \mathbf{z})$. This is achieved by training an inference network $q_\phi(\mathbf{z}|\mathbf{c})$ with parameters ϕ that estimates the posterior $p(\mathbf{z}|\mathbf{c})$. Here, $\mathbf{c}$ is referred to as the context. Assuming the objective to be a log-likelihood, the resulting variational lower bound is denoted by:

$$\mathbb{E}_{\mathcal{T}}[\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{c}^{\mathcal{T}})}[\mathcal{R}(\mathcal{T}, \mathbf{z}) + \beta D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{c}^{\mathcal{T}}))||p(\mathbf{z})]],$$

where prior over Z, $p(\mathbf{z})$, is modelled by a unit Gaussian and $\mathcal{R}(\mathcal{T}, \mathbf{z})$ stands for the objective. The Kullback–Leibler divergence term constrains $\mathbf{z}$ to contain only information that is necessary to adapt to new tasks.

The inference network $q(\mathbf{z}|\mathbf{c}_{1:N})$ is designed to be a product of independent factors, i.e.

$$q(\mathbf{z}|\mathbf{c}_{1:N}) \propto \Pi_{n=1}^{N} \Psi_\phi(\mathbf{z}|\mathbf{c}_n).$$

For tractability reasons Gaussian factors are used, $\Psi_\phi(\mathbf{z}|\mathbf{c}_n) = \mathcal{N}(f_\phi^\mu(\mathbf{c}_n), f_\phi^\sigma(\mathbf{c}_n))$, resulting in a Gaussian posterior.

This function f_ϕ is represented as a neural network that learns to predict the mean μ and the variance σ of $\mathbf{c}_n$. With this architecture the encoding of a fully observed MDP is so-called permutation invariant. The method of modeling the latent context as probabilistic allows for posterior sampling enabling efficient exploration at testing.

Algorithm 2 denotes in detail the meta-training procedure of PEARL. For readers not familiar with the soft-actor-critic algorithm and its corresponding loss terms used in PEARL, we kindly point to the work by Rakelly et al. [3].

Algorithm 2 PEARL Meta-Training

Require: Batch of training tasks $\{\mathcal{T}_i\}_{i=1...T}$ from $p(\mathcal{T})$, learning rates $\alpha_1, \alpha_2, \alpha_3$

Initialize replay buffers $\mathcal{B}^i$ for each training task
while not done **do**
  for each $\mathcal{T}_i$ **do**
    Initialize context $\mathbf{c}^i = \{\}$
    for $k = 1, ..., K$ **do**
      Sample $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{c}^i)$
      Gather data from $\pi_\theta(\mathbf{a}|\mathbf{s}, \mathbf{z})$ and add to $\mathcal{B}^i$
      Update $\mathbf{c}^i = \{(\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j)\}_{j:1...N} \sim \mathcal{B}^i$
    end for
  end for
  for step in training steps **do**
    for each $\mathcal{T}_i$ **do**
      Sample context $\mathbf{c}^i \sim \mathcal{S}_c(\mathcal{B}^i)$ and RL batch $b^i \sim \mathcal{B}^i$
      Sample $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{c}^i)$
      $\mathcal{L}^i_{actor} = \mathcal{L}_{actor}(b^i, \mathbf{z})$
      $\mathcal{L}^i_{critic} = \mathcal{L}_{critic}(b^i, \mathbf{z})$
      $\mathcal{L}^i_{KL} = \beta D_{KL}(q(\mathbf{z}|\mathbf{c}^i)||r(\mathbf{z}))$
    end for
    $\phi \leftarrow \phi - \alpha_1 \nabla_\phi \sum_i (\mathcal{L}^i_{critic} + \mathcal{L}^i_{KL})$
    $\theta_\pi \leftarrow \theta_\pi - \alpha_2 \nabla_\theta \sum_i \mathcal{L}^i_{actor}$
    $\theta_Q \leftarrow \theta_Q - \alpha_3 \nabla_\theta \sum_i \mathcal{L}^i_{critic}$
  end for
end while

MACAW

Meta-Actor Critic with Advantage Weighting (MACAW) [4] is an offline meta-RL algorithm and architecture that possesses three key properties: sample efficiency, offline meta-training, and consistency at meta-test time. MACAW is the first algorithm to successfully combine gradient-based meta-learning and off-policy value-based RL [4]. By learning initializations ϕ and θ for a value function V_ϕ and policy π_θ, MACAW is able to quickly adapt to tasks first seen at meta-test time by means of gradient descent. The policy objectives as well as the value function are learned by simple weighted regression losses in the inner- and the outer-loop, similarly to MAML. In the inner-loop, adaptation is first performed for the value function using bootstrap-free updates via supervised regression onto Monte-Carlo returns. The loss for this update procedure is denoted as follows:

$$\mathcal{L}_V(\phi, D) = \mathbb{E}_{\mathbf{s},\mathbf{a}\sim D}\left[(V_\phi(\mathbf{s}) - \mathcal{R}_D(\mathbf{s}, \mathbf{a}))^2\right],$$

where $\mathcal{R}_D(\mathbf{s}, \mathbf{a})$ referring to the Monte-Carlo return.

Followed by this, is the policy update which performs supervised regression as well. Here however, actions weighted by the estimated advantage are regressed instead of Monte-Carlo returns. The loss for this is given by the following term

$$\begin{aligned}
\mathcal{L}_\pi =& \mathcal{L}^{\text{AWR}} + \lambda \mathcal{L}^{\text{ADV}} \\
\mathcal{L}^{\text{AWR}}(\theta, \phi, D) =& \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim D}\left[-\log \pi_\theta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{T}(\mathcal{R}_D(\mathbf{s}, \mathbf{a}) - V_\phi(\mathbf{s}))\right)\right] \\
\mathcal{L}^{\text{ADV}}(\theta, \phi'_i, D) =& \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim D}\left[(A_\theta(\mathbf{s}, \mathbf{a}) - (\mathcal{R}_D(\mathbf{s}, \mathbf{a}) - V_{\phi'_i}(\mathbf{s})))^2\right]
\end{aligned}$$

For the policy loss, these two objectives are combined as using solely $\mathcal{L}^{\text{AWR}}$ lacks expressiveness.

The training procedure done in the outer-loop is conducted similarly as in MAML. This time however meta-parameters θ and ϕ are optimized utilizing the objectives $\mathcal{L}_V$ and $\mathcal{L}^{\text{AWR}}$.

For a more detailed view on the performed updates during the inner- and outer-loop see Algorithm 3 or correspondingly the work by Mitchell et al. [4].

Algorithm 3 MACAW Meta-Training

Require: Tasks $\{\mathcal{T}_i\}$, offline buffers $\{D_i\}$, learning rates $\alpha_1, \alpha_2, \eta_1, \eta_2$
Randomly initialize meta-parameters θ, ϕ
while not done **do**
  for task $\mathcal{T}_i \in \{\mathcal{T}_i\}$ **do**
    Sample disjoint batches $D_i^{\text{tr}}, D_i^{\text{ts}} \sim D_i$
    $\phi_i' \leftarrow \phi - \eta_1 \nabla_\phi \mathcal{L}_V(\phi, D_i^{\text{tr}})$
    $\theta_i' \leftarrow \theta - \alpha_1 \nabla_\theta \mathcal{L}_\pi(\theta, \phi_i', D_i^{\text{tr}})$
  end for
  $\phi \leftarrow \phi - \eta_2 \sum_i [\nabla_\phi \mathcal{L}_V(\phi_i', D_i^{\text{ts}})]$
  $\theta \leftarrow \theta - \alpha_2 \sum_i [\nabla_\theta \mathcal{L}^{\text{AWR}}(\theta_i', \phi_i', D_i^{\text{ts}})]$
end while

Default Parameters

MAML-PPO	
Parameter	**Value**
meta_batch_size	40
num_inner_grad_steps	1
num_ppo_steps	5
rollouts_per_meta_task	20
α	0.1
β	0.001
rollouts_per_meta_task	20
max_path_length	200
γ	0.99
clip_eps	0.5
init_inner_kl_penalty	0.001

MACAW	
Parameter	**Value**
α_1, η_1	0.001
α_2, η_2	0.0001
meta_batch_size	256
batch_size	256
inner_batch_size	256
eval_batch_size	256
buffer_skip	7
inner_buffer_skip	7
instances	3
task_batch_size	5
maml_steps	1
eval_maml_steps	1
γ	0.99

RL^2	
Parameter	**Value**
meta_batch_size	100
rollouts_per_meta_task	2
max_path_length	200
γ	0.99
learning_rate	0.001
cell_type	LSTM
max_epochs	5

PEARL	
Parameter	**Value**
meta_batch_size	10
num_steps_per_eval	600
num_initial_steps	2000
num_train_steps_per_itr	4000
num_iterations	500
$\alpha_1, \alpha_2, \alpha_3$	0.0003
batch_size	256
context_lr	0.0003
γ	0.99
embedding_batch_size	256
embedding_mini_batch_size	256
max_path_length	200
num_exp_traj_eval	2
num_extra_rl_steps_posterior	600
num_steps_prior	400
num_tasks_sample	5
soft_target_tau	0.005