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

1 Introduction

In order to tackle the problem of needing large amounts of data to learn a particular task, a commonly used method is the idea of meta-learning. In the domain of reinforcement learning, meta-learning aims to capture the common dynamics of a group of similar tasks originating from some environment to later on utilize this knowledge to quickly adapt to 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 in meta-reinforcement learning: context-based and gradient-based methods. In context-based approaches 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-reinforcement learning 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 of 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 these algorithms when varying the amount of available tasks or withholding some of them during training completely. Robustness from here on out, 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 hypothesise that this will provide new insights in regard to out-of-sample 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.

2 Related Work

The following enumeration depicts the selected meta-reinforcement learning algorithms explored in this work:

- RL^2 [1]
- MAML [2]
- PEARL [3]
- MACAW [4]

Regarding context-based meta-reinforcement learning algorithms, we decided on RL^2 due to its popularity as one of the foundational papers in this realm. The same holds for MAML in the domain of gradient-based methods. A more recent work to this poses the algorithm by Rakelly et al. named

PEARL. 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 these, we also consider MACAW which is a recent algorithm purely operating in the offline meta-reinforcement learning setting. Although it might seem difficult to compare this algorithm to online variants, we choose to include this algorithm for the reason of emphasising the availability 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 some 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-sample performance of these algorithms making the need of such a study ever more so crucial.

3 Experiments

As pointed out already in Section 1, the focus of our experiments is laid on the availability and variety of tasks during training. The main idea behind this focus is due to the mentioned lack of such studies as described in Section 2.

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 experiment, we found the Ant-Direction environment from the MuJoCo [5] 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 similar experiment to this has already been conducted by Mitchell et al. [4]. However, they solely considered the setting of offline meta-reinforcement learning. Given this fact, we want to see for MACAW if comparable performance conclusions can be drawn for less complex environments, i.e. if a too small (< 5) or a too large (> 20) amount of tasks/offline buffers will hurt performance similarly. 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, plays 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 good results. We assume that this is due to the problem that if the amount 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 a ten-arm Bernoulli-Bandit environment with only five tasks. If it now happens that in three of those, 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. For instance, in the Ant-Direction environment, this could mean that only north-east direction tasks, i.e. from 0 to 90°, are available during training. By then testing on all the other directions, we are able to measure the out-of-sample performance of our selected algorithms in a precise manner.

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 serve the purpose to answer the following questions:

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

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

(c) How robust are the selected online meta-reinforcement learning algorithms within, if the meta-batch size is varied?

(d) How good do our selected algorithms perform in regard to out-of-sample performance?

(e) How quickly can MAML obtain good performance on out-of-sample tasks?

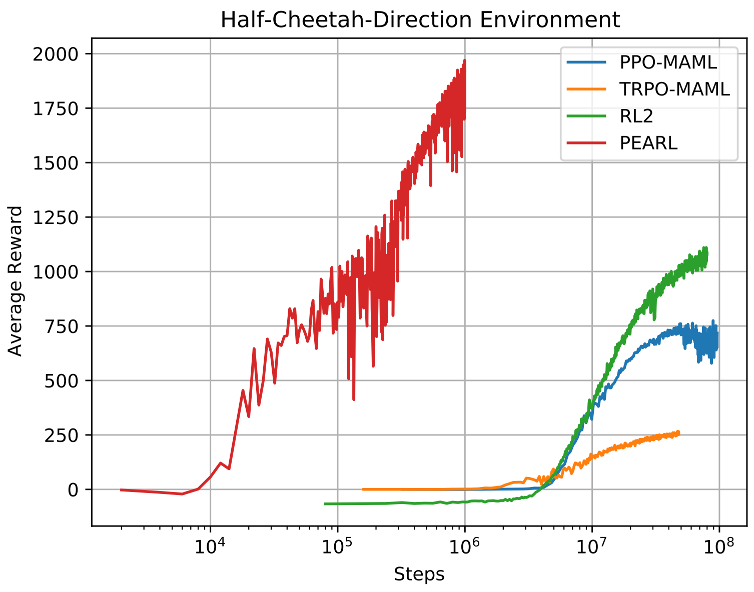

Figure 1: Average reward for the Cheetah-Direction environment for MAML, RL^2, and PEARL. PEARL is able to achieve much higher end performance while at the same time requiring $\sim$ 100 times less sample steps than the second-best RL^2 to converge.

4 Results and Discussion

In the following section we present some initial results in order to analyze the properties of our selected meta-reinforcement learning algorithms. These were conducted in order gain insights about sample-efficiency. Our results preliminary include experiments on the MuJoCo physics engine and its respective environments [5].

After experimenting with a variety of different implementations and libraries, most of them being out of date or containing errors, we have conducted our experiments based primarily on the works of [3], [4] and [6].

4.1 Evaluation on MuJoCo

For our first experiment, we have visualized the performance of our algorithms on the Half-Cheetah-Direction environment in Figure 1. Here, we plot the average reward over a whole episode with max

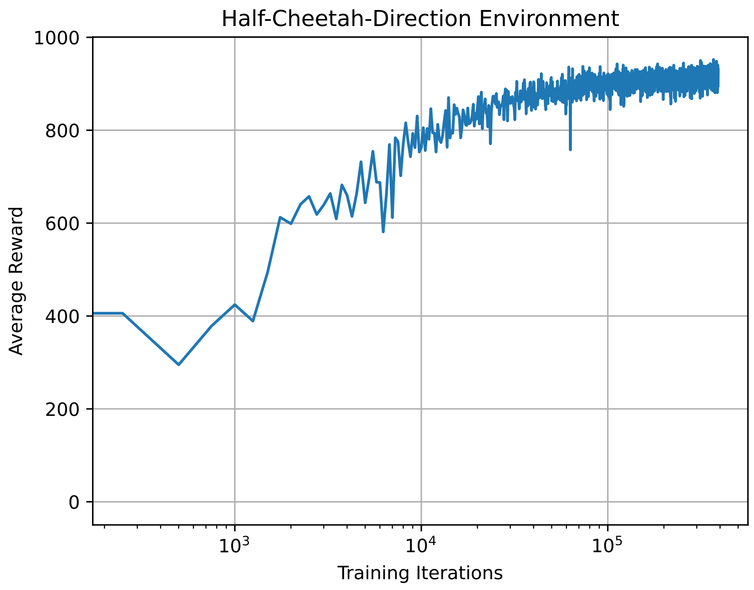

Figure 2: Average reward for the Cheetah-Direction environment for MACAW with a offline buffer size of 5 million.

trajectory lengths of 200 in relation to the amount of steps that were in total taken during training. As the algorithms differ in their strategies, they were trained with varying amounts of steps. For instance, for the PPO-MAML algorithm, we have *n_tasks* (2) * *meta_batch_size* (20) * *rollouts_per_meta_task* (40) * *max_path_len* (200) * *n_itr* (300) $= 96,000,000 \sim 10^8$ steps while the offline approach of MACAW utilizes significantly less steps due to its offline buffer usage. This is also the reason why the results of the algorithms have different starting points as e.g. PEARL is updated and evaluated after much fewer samples.

There are multiple interesting insights that can be made out. The two MAML variants as well as RL^2 are very sample inefficient, needing around $10^7 - 10^8$ samples to converge and even start improving. The off-policy approach of PEARL on the other hand not only needs much less samples to converge (10^6) but also performs much better earlier on.

As mentioned in Appendix A, we see that there is a large discrepancy between the performance of the PPO and the TRPO version of MAML, achieving 750 for the former and 250 for the latter. RL^2 initially starts out the worst with a negative reward, but is able to achieve the second best reward of 1,100 after close to 10^8 steps. We note that, as opposed to the initial implementation with TRPO [1], our variation uses PPO and as such performs better than the results mentioned in the original paper. This is only beaten by PEARL which achieves a more unstable top reward of 1,900, which is significantly higher and achieved with 1% of the amount of sample steps of RL^2.

MACAW has to be considered separately as we first create a buffer of $5 * 10^6$ samples. Looking at Figure 2, we see that after 10^5 training iterations the algorithm converges with a score of around 900. This makes it better than both MAML variants and worse than PEARl and RL^2. While MACAW has due to the initial buffer very good sample-efficiency, similar to PEARL, its sophisticated training procedure means it takes a lot longer to finish.

We have further looked at a less complex Bandit environment for MACAW with a buffer size of 60,000. Here we see that the model only needs around 250 training iterations to achieve near optimal performance.

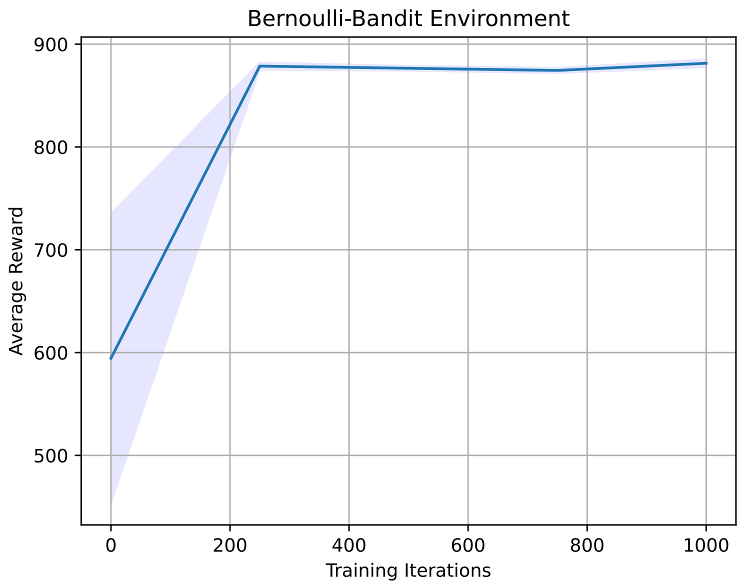

Figure 3: Comparing the performance of MACAW in a Bernoulli-Bandit environment evaluated for a 1,000 pulls at each step with 12 tasks to train on.

5 Conclusion/Future Work

For now we illustrated a proof of concept and that we were able to obtain results and report them for the selected algorithms. Note that most of these results will be adapted in the final report. Based on this, our future work mainly consists of extending our implementations to be suitable for our designed experiments. This includes adding further environments (Ant-Direction), modifying the algorithms such that the availability and variety of tasks can be restricted, conducting the experiments, and reporting the results analytically and visually. For a more in-detail explanation of the experiments refer to Section 3. Finally, as we already exceeded the amount of pages, we did not spend more time on explaining the algorithms in a precise manner and thus moved them to the Appendix A.

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] 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`.

[6] 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`.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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`.

[12] 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.

[13] 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.

A Appendix

Methods/Algorithms

RL^2

In RL^2 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 [12] or PPO [13]. However, PPO has been shown to perform preferably in practice, although performance increases may primarily be due to code-level optimizations as suggestd by Engstrom et al. [11].

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 handcrafting it has laid the foundations for many meta-reinforcement learning algorithms and has back then simultaneously been researched by Wang et al. [7].

This general procedure is visualized in Figure 4 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, reward r_t and 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} which the policy is conditioned on for the next output. 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.

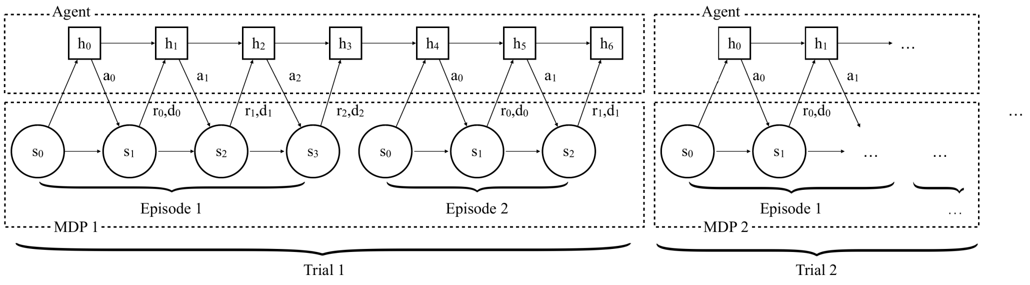

Figure 4: In RL^2 an agent acts within an environment learning the weights of its RNN to maximize the per trial reward where each trial constitutes of a separate MDP. [1]

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 reinforcement learning. During meta-learning, the model, denoted further on as f, is trained to be able to adapt to a large or infinite number of tasks. In the framework of reinforcement learning, the goal of few-shot meta-learning is to enable an agent to quickly learn a policy for a new task using a small amount of experience in the test setting. For example, an agent might learn to quickly figure out how to navigate mazes in general, so that when exposed to a new maze it reaches the exit with only a few examples. MAML can thus be thought of as training a model to be easily fine-tuneable.

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 $q_i(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)$, while the loss $\mathcal{L}_{\mathcal{T}_i}$ corresponds to the reward function R. The model that we are trying to learn is f_θ, a policy that maps from states $\mathbf{x}_t$ to a distribution over actions $\mathbf{a}_t$ at each timestep $t \in \{1, ..., H\}$. Optimization is performed on the following objective;

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

For this loss we then calculate the gradient in regard to each task resulting in a specific θ_i' from which we then sample new trajectories. After one iteration of all tasks we calculate the gradient to our original θ summing up the loss for each $f_{\theta_i'}$. Details about the algorithm can be found in Algorithm 1.

PEARL

Previously discussed algorithms rely heavily on on-policy experience, limiting their sampling efficiency. What is more, they lack mechanisms to reason about task uncertainty when adapting to new tasks. These challenges are addressed by Probabilistic Embeddings for Actor-critic meta-reinforcement learning (PEARL) [3], an off-policy algorithm that tries 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 $\pi_\theta(\mathbf{a}|\mathbf{s}, \mathbf{z})$. In order to do that, the latent context variable Z encodes the underlying information about the task. The inference network $q_\phi(\mathbf{z}|\mathbf{c})$ estimates the posterior $p(\mathbf{z}|\mathbf{c})$. Assuming the objective to be a log-likelihood, the resulting variational lower bound is given by:

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

where $p(\mathbf{z})$ is a unit Gaussian prior over Z and $\mathcal{R}(\mathcal{T}, \mathbf{z})$ could be a variety of objectives. The Kullback–Leibler divergence term constrains **z** to contain only information that is necessary to adapt to new tasks.

The inference network $q(\mathbf{z}|\mathbf{c}_{1:N})$ is modelled as 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. We learn the function f_ϕ, represented as a neural network, which predicts the mean μ and the variance σ of $\mathbf{c}_n$. With this architecture the encoding of a fully observed MDP is permutation invariant. Modeling the latent context as probabilistic allows us to make use of posterior sampling for efficient exploration at testing. Details about the algorithm can be found in Algorithm 2.

Designing off-policy meta-reinforcement learning algorithms is non-trivial partly because modern meta-learning is predicated on the assumption that the distribution of data used for adaptation will match across training and testing. This poses an interesting problem which we are going to investigate further during our our robustness and out-of-sample performance analysis.

MACAW

Meta-Actor Critic with Advantage Weighting (MACAW) [4] is an offline meta-reinforcement learning 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]. It learns an initialization ϕ and θ for a value function V_ϕ and policy π_θ, that can rapidly adapt to new tasks seen at meta-test time via gradient descent. Policy objectives and value function both correspond to simple weighted regression losses in the inner-loop and the outer-loop. This leads to a stable, consistent inner-loop adaptation process and outer-loop meta-training signal. In the inner-loop of the MACAW's training bootstrap-free updates are used for the value function that simply performs supervised regression onto Monte-Carlo returns, where the loss function equals:

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

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

The policy update uses the following objective:

$$\mathcal{L}_\pi = \mathcal{L}^{AWR} + \lambda\mathcal{L}^{ADV}$$

$$\mathcal{L}^{AWR}(\theta, \phi, D) = \mathbb{E}_{\mathbf{s},\mathbf{a}\sim D}\left[-\log\pi_\theta(\mathbf{a}|\mathbf{s})\exp\left(\frac{1}{T}(\mathcal{R}_D(\mathbf{s}, \mathbf{a}) - V_\phi(\mathbf{s}))\right)\right]$$

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

In contrast to the inner.loop, the initial policy parameters are optimized utilizing solely an advantage-weighted regression objective, i.e. $\mathcal{L}^{\text{AWR}}$. Regarding the value function, the the same loss as in the inner-loop is used, however, now averaged over the previous obtained gradients. In order to further improve the algorithm a weight transform layer is proposed. This layer maps a latent code into the layer's weight matrix and bias, which are then used to compute the layer's output just as in a typical fully-connected layer. That way the expressiveness of MAML's gradient increases. For a more detailed view on the performed updates during meta-training see Algorithm 3.

Algorithms

Algorithm 1 MAML for Reinforcement Learning

```
Require: $p(\mathcal{T})$ distribution over tasks
Require: $\alpha, \beta$ step size hyperparameters
 1: randomly initialize $\theta$
 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 $f_\theta$ in $\mathcal{T}_i$
 6:         Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Appendix A
 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 $\mathcal{L}_{\mathcal{T}_i}$ in Appendix A
11: end while
```

Algorithm 2 PEARL Meta-Training

```
Require: Batch of training tasks $\{\mathcal{T}_i\}_{i=1...T}$ from $p(\mathcal{T})$
Require: Learning rates $\alpha_1, \alpha_2, \alpha_3$
 1: Initialize replay buffers $\mathcal{B}^i$ for each training task
 2: while not done do
 3:     Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 4:     for each $\mathcal{T}_i$ do
 5:         Initialize context $\mathbf{c}^i = \{\}$
 6:         for $k = 1, ..., K$ do
 7:             Sample $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{c}^i)$
 8:             Gather data from $\pi_\theta(\mathbf{a}|\mathbf{s}, \mathbf{z})$ and add to$\mathcal{B}^i$
 9:             Update $\mathbf{c}^i = \{(\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}_j', r_j)\}_{j:1...N} \sim \mathcal{B}^i$
10:         end for
11:     end for
12:     for step in training steps do
13:         for each $\mathcal{T}_i$ do
14:             Sample context $\mathbf{c}^i \sim \mathcal{S}_c(\mathcal{B}^i)$ and RL batch $b^i \sim \mathcal{B}^i$
15:             Sample $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{c}^i)$
16:             $\mathcal{L}^i_{actor} = \mathcal{L}_{actor}(b^i, \mathbf{z})$
17:             $\mathcal{L}^i_{critic} = \mathcal{L}_{critic}(b^i, \mathbf{z})$
18:             $\mathcal{L}^i_{KL} = \beta D_{KL}(q(\mathbf{z}|\mathbf{c}^i)||r(\mathbf{z}))$
19:         end for
20:         $\phi \leftarrow \phi - \alpha_1 \nabla_\phi \sum_i (\mathcal{L}^i_{critic} + \mathcal{L}^i_{KL})$
21:         $\theta_\pi \leftarrow \theta_\pi - \alpha_2 \nabla_\theta \sum_i \mathcal{L}^i_{actor}$
22:         $\theta_Q \leftarrow \theta_Q - \alpha_3 \nabla_\theta \sum_i \mathcal{L}^i_{critic}$
23:     end for
24: end while
```

Algorithm 3 MACAW Meta-Training

```
Require: Tasks $\{\mathcal{T}_i\}$, offline buffers $\{D_i\}$
Require: learning rates $\alpha_1, \alpha_2, \eta_1, \eta_2$
 1: Randomly initialize meta-parameters $\theta, \phi$
 2: while not done do
 3:     for task $\mathcal{T}_i \in \{\mathcal{T}_i\}$ do
 4:         Sample disjoint batches $D_i^{\text{tr}}, D_i^{\text{ts}} \sim D_i$
 5:         $\phi'_i \leftarrow \phi - \eta_1 \nabla_\phi \mathcal{L}_V(\phi, D_i^{\text{tr}})$
 6:         $\theta'_i \leftarrow \theta - \alpha_1 \nabla_\theta \mathcal{L}_\pi(\theta, \phi'_i, D_i^{\text{tr}})$
 7:     end for
 8:     $\phi \leftarrow \phi - \eta_2 \sum_i [\nabla_\phi \mathcal{L}_V(\phi'_i, D_i^{\text{ts}})]$
 9:     $\theta \leftarrow \theta - \alpha_2 \sum_i [\nabla_\theta \mathcal{L}^{\text{AWR}}(\theta'_i, \phi'_i, D_i^{\text{ts}})]$
10: end while
```