Research in Data Science Project Report

Increasing Sample-Efficiency in Reward Learning with F-PACOH

Student: Lukas Fluri

Supervisors: David Lindner, Jonas Rothfuss, Prof. Dr. Andreas Krause

flurilu@student.ethz.ch

{david.lindner, jonas.rothfuss, andreas.krause}@inf.ethz.ch

December 27, 2022

1 Introduction

For many problems in reinforcement learning it can be very hard to specify a reward function which correctly represents the user's goal. [1] Reward learning tries to remedy this problem by letting an agent learn the reward function instead of requiring a user to provide its exact definition. The exact way how the reward function is learned can be different from task to task. Many different methods have been proposed, such as comparisons [2], demonstrations [3], learning from the environment [4] etc. (see [5] for a nice overview). However, such methods require a lot of information in order to be able to infer the correct reward function. And since this information usually has to be generated from a human, such methods can become quite labor-intensive. Meta learning and especially its application to few-shot learning tries to alleviate this problem by making use of prior information gained from training on similar tasks [6]. By exploiting the information of previous tasks, the promise of meta-learning is that this drastically reduces the number of data points required to adapt a model to a new, related task. Applied to reward learning, this could help to reduce the amount of manual work required to let an agent learn a new reward function. However, if the number of provided prior tasks is small, existing methods tend to over-fit to the previously learned tasks [7]. As a result the model tends to perform poorly on a new, unseen task. This reduces the benefits of applying meta-learning to the problem of reward learning since specifying a large number of prior tasks requires again a lot of manual human labor.

A recently proposed method called PACOH [8] tries to solve these issues by providing a learning-theoretic approach to meta-learning. Their experimental results show that PACOH is able to improve the predictive accuracy of its base learner with as little as 5 meta-tasks. A recent enhancement called F-PACOH [9] additionally provides well-calibrated uncertainty estimates which are particularly important for reinforcement-learning (and hence also reward learning) problems. In this project we apply the F-PACOH framework to several meta reward-learning problem settings and compare the amount of prior tasks needed to improve performance on a new, unseen task. Our code can be found on Github (see here [10])

The rest of this report is structured as follows: Section 2 first provides some basic background and describes the general methodology used to perform the experiments, section 3 states and analyzes the results, section 4 talks about solved and open issues, as well as future work, and section 5 summarizes the main findings of the report.

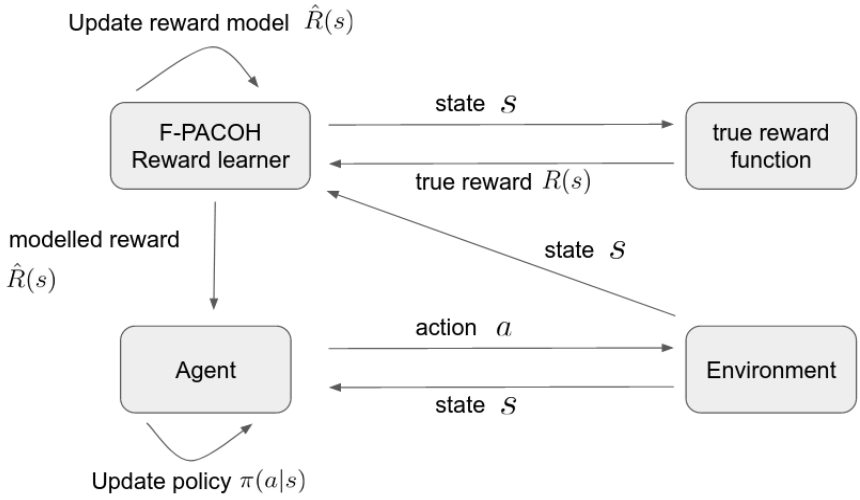


Figure 1: An overview of the experiment setup

2 Background & Methodology

2.1 Background

Reinforcement learning The classic reinforcement learning problem models the scenario of an agent acting inside some environment. It can be described by a 5-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where $\mathcal{S}$ is the set of states of the environment, $\mathcal{A}$ is the set of actions an agent can take in this environment, and $\mathcal{P} : S \times A \to \Delta(\mathcal{S})$ is the transition function which maps states and actions to a probability distribution over the set of states. For example, the transition $s, a \to \Delta(\mathcal{S})$ can be understood as the probability of an agent ending up in the different states of the environment, given that it started in state s and took action a. The reward function $\mathcal{R}$ is defined over transitions $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to R$. The goal of the agent is to find a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ which maximizes the expected discounted return:

$$r(\pi) \;=\; E\left[\sum_{i=0}^{\infty} \gamma^i \cdot \mathcal{R}(s_i, a_i, s_{i+1}) \mid a_i \sim \pi(s_i),\; s_{i+1} \sim P(s_i, a_i)\right] \tag{1}$$

Reward learning Reward learning is an extension of reinforcement learning. The basic idea is that the agent either has no direct access or only very limited access to the true reward function. In order to still be able to learn a policy π which maximizes the expected return $r(\pi)$, it needs to model the reward function itself. Reward learning can be defined by the 6-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \hat{\mathcal{R}}, \gamma \rangle$, where $\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}$, and γ are defined as before and $\hat{\mathcal{R}}$ is the agent's learned reward function, which tries to approximate the true reward function $\mathcal{R}$. The goal of the reward learning problem is again to find a policy π which maximizes the expected discounted return $r(\pi)$ (see expression 1).

Meta-learning Meta-learning is a framework which can be used in situations where one needs to learn how to solve a task t from little data while having access to a distribution of "similar" tasks $p(\mathcal{T})$ where $t \sim \mathcal{T}$. The idea is that by learning model parameters θ which minimize the expected loss $E_{t \sim p(\mathcal{T})}\left[\mathcal{L}(\mathcal{D}_t, \theta)\right]$, it is possible to adapt these parameters quickly and with little extra data to a previously unseen task $t \sim p(\mathcal{T})$.

Algorithm 1 Experiment setup

Initialize MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
Initialize reward model $\hat{\mathcal{R}}$
if do meta-training **then**
    Meta-train $\hat{\mathcal{R}}$
end if
$num\_queries \leftarrow 0$
while $num\_queries < num\_total\_queries$ **do**
    Train rl agent a with reward model $\hat{\mathcal{R}}$
    $V \leftarrow$ state-action pairs visited during training of a
    $U \leftarrow$ state-action pairs from V where $\hat{\mathcal{R}}$ is most uncertain
    $r \leftarrow \mathcal{R}(U)$ ▷ Get the true rewards
    Update reward model $\hat{\mathcal{R}}$ with (U, r)
end while

As a simple example, imagine a robot arm which learns how to reach a certain position in its action space. Since we will probably want the arm to learn how to reach several different positions in the future, it makes sense to directly train the robot arm to optimize the expected reward on a randomly sampled reaching task, instead of learning to solve an individual task only.

If the available set of tasks $t \sim p(\mathcal{T})$ is small, it might happen that the meta-learner overfits to the meta-training tasks $\{t_i | t_i \sim p(\mathcal{T})\}_{i=1}^{n}$ and will perform poorly when trying to learn a new task from $p(\mathcal{T})$. PACOH [8] and its enhancement F-PACOH [9] alleviate this problem by providing a Bayes-optimal learning method with PAC guarantees which allows a user to reason about the generalization ability of their meta-learning model.

2.2 Methodology

Experiment overview In order to compare the advantages of using F-PACOH for reward learning we perform reward learning experiments where we compare classic reward learners with meta-learned reward learners based on F-PACOH. The whole experiment setting is depicted in figure 1. In each experiment the reward learner starts with little knowledge about the true reward function $\mathcal{R}$. We then repeatedly perform the following steps:

1. Train a reinforcement learning agent on the environment. The rl agent uses the approximated reward function $\hat{\mathcal{R}}$ of the reward learner as reward function.
2. Let the reward learner choose a few state-action pairs (s, a) which were visited by the rl agent during a previous training step. Preferably, choose the state-action pairs about whose reward $\hat{\mathcal{R}}((s, a))$ the reward model is most uncertain about.
3. Query the true reward function about the selected state-action pairs (s, a) and receive the true reward $\mathcal{R}((s, a))$. Update the reward model.

A complete pseudocode of the experiment can be found in algorithm 1. Depending on the particular choice of environment, the performance of the experiment might vary quite a bit. In order to remedy this variance, we run each experiment for 50 different environments and plot the mean performance, as well as the standard deviation.

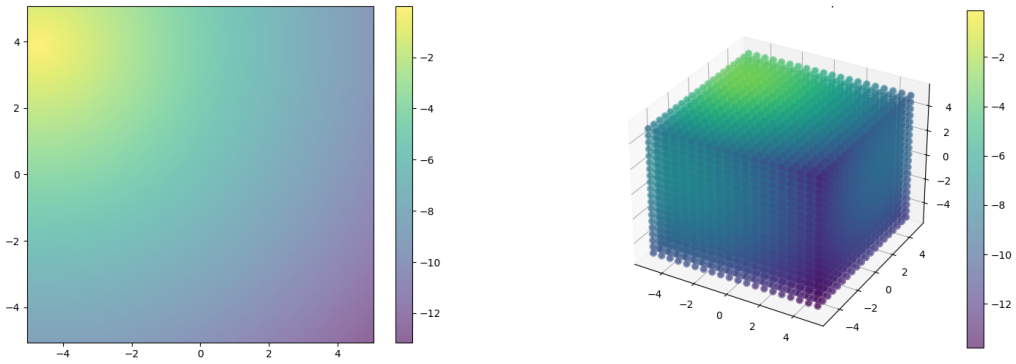


Figure 2: Examples of the two different environment classes used for the experiments. The plots depict the 2D- and 3D hypercubes centered around $\mathbf{0}$. The colormap indicates the reward an agent receives at the different positions.

Environments For our experiments we use simple navigation task environments (see figure 2). The state-space is a 2D- or 3D- hypercube. Each environment contains a randomly sampled target position t. The state of the environment is the current position x of the agent. Initially, the agent starts in the center of the hypercube (i.e at position $(0, 0)$ or $(0, 0, 0)$) and the goal of the task is for the agent to reach the target position t. As a reward function the negative Euclidean distance of the agent position and the target position, defined as $\mathcal{R}(x) = -||x - t||_2$, was chosen. Each episode consists of 200 steps in which the agent can walk up to 0.1 units in each direction.

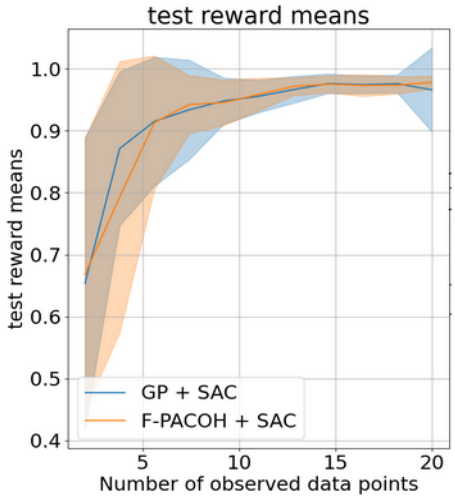
Reward learner We use Gaussian processes as baseline for the standard reward learner. For the meta-learned reward learner, we use F-PACOH with Gaussian processes as base learners. The meta-learner is trained with 20 tasks containing 20 data samples each.

Reinforcement learning agent For our experiments we use an SAC algorithm due to its sample efficiency. The agent is trained for 100k time steps. As reward function, we use the upper confidence bound of the reward model. This is because the reward learner can only query the true reward function for states which the reinforcement learning agent visited and we want the reward learner to be able to make queries for states with high uncertainty. So, instead of defining the reward function $R(s) = \hat{\mathcal{R}}(s)$ for some state s, we also add two times the standard deviation: $R(s) = \hat{\mathcal{R}}(s) + 2 \cdot std(\hat{\mathcal{R}}(s))$.

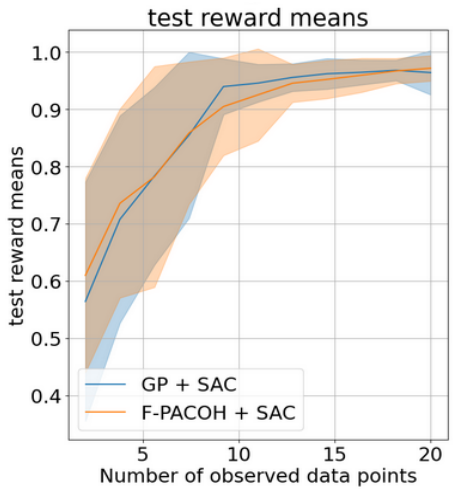
Depending on the environment, the minimum and maximum possible return of the reinforcement learning agent can differ. In order to compare the performance of the agent in different environments, the return must therefore be re-scaled. The smallest and largest possible return are achieved by the policies which move the agent to the target position (largest reward) or the furthest away from the target position (smallest reward). Using the returns from these two policies, the return can be scaled to lie in the interval $[0, 1]$.

3 Results

Baselines To test the advantages of meta-learning in reward learning, we compare it with standard reward learning. We use Gaussian processes [11] as standard reward learners. Choosing GPs



(a) Scaled returns of the SAC agent using different reward learners in the 2D environment



(b) Scaled returns of the SAC agent using different reward learners in the 3D environment

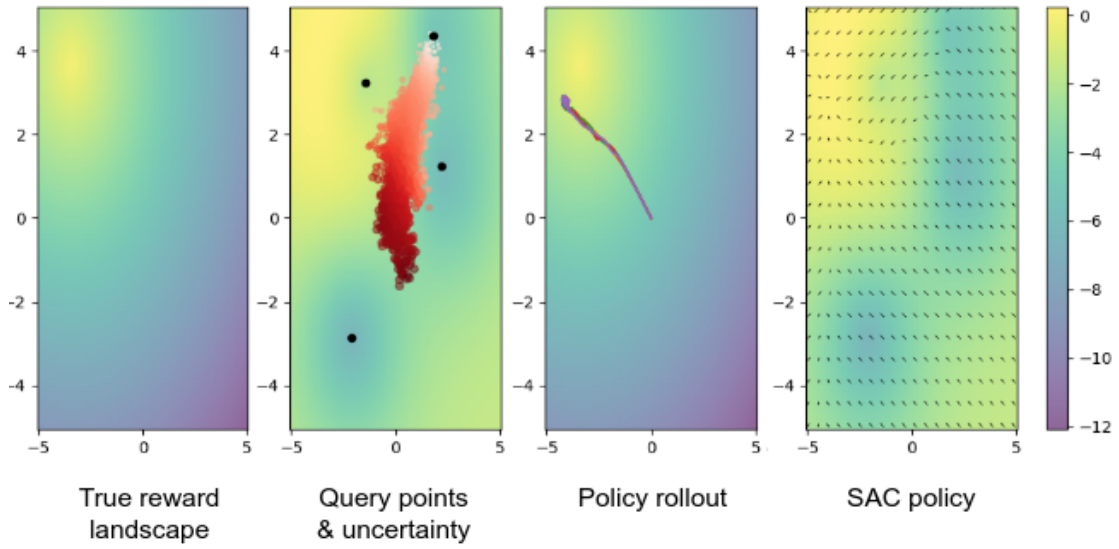
Figure 3: The scaled returns of the reinforcement learning agent as a function of the number of observed reward signals. Each experiment was run with 50 different environments and seeds. The plots display the mean and standard deviation of these runs

is suitable because F-PACOH also uses GPs as base learners. This allows for a more direct and fair comparison.

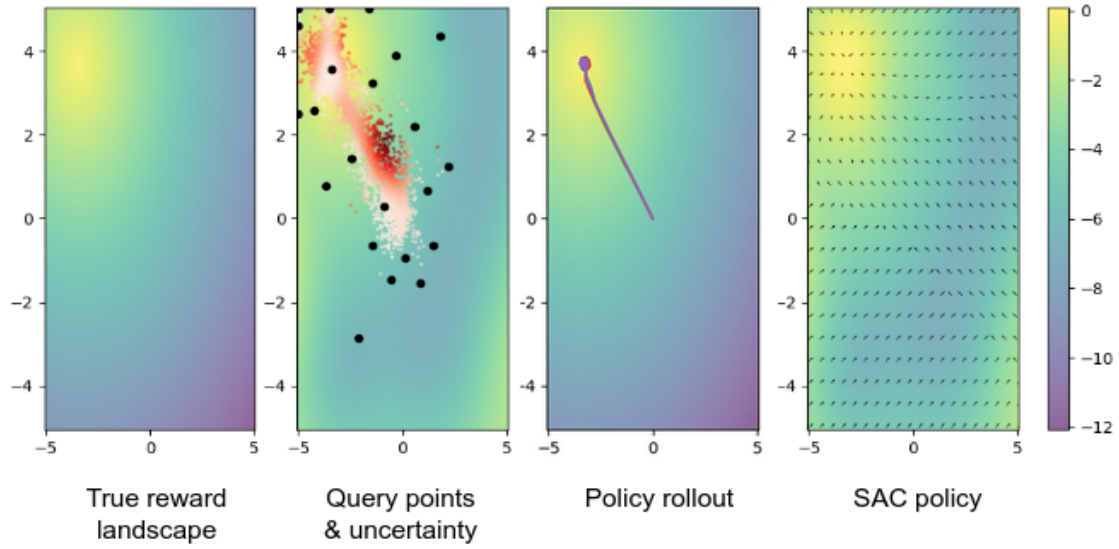
Main results Figure 3 shows the main results of our experiment. Both, the simple GP reward learner and the meta-learned reward learner, are able to adapt to a new, previously unseen task. They are both able to learn the new reward landscape to a degree good enough, that the reinforcement learning agent is able to learn a correct policy which moves to the target position. For these specific environment types, the choice of reward learner does not seem to make a difference regarding performance.

Analysis of main result Figure 3 To find the reason for why the meta-learned reward model does not lead to better performance it helps to look at an individual run of our experiments. Figure 4 shows two snapshots of a single experiment run with a Gaussian process reward learner and an SAC policy. The first row (figure 4a) shows the state after the GP made 4 queries to the true reward function, whereas the second row (figure 4b) shows the state after the GP made 20 queries to the true reward function. The second and third column show an interesting behavior. While the uncertainty of the GP reward model is still very high after 4 queries to the true reward function (figure 4a, column 2), the mean function of the GP has already a very high accuracy (4b, compare column 3 to column 1).

The reason for this fast convergence to the true reward function is the hyperparameter tuning of the Gaussian process. We selected the hyperparameters which let the GP best learn the true reward function over a large set of randomly sampled environments. Because our reward functions are so simple, this lead the hyperparameter search to select a large lengthscale parameter for the RBF kernel, such that a few points are already sufficient for the GP to model the entire reward landscape. Figure 8 on page 14 shows the same snapshot for the F-PACOH reward learner. We can see a similar behavior as for the Gaussian process. However, in this case the model is able to

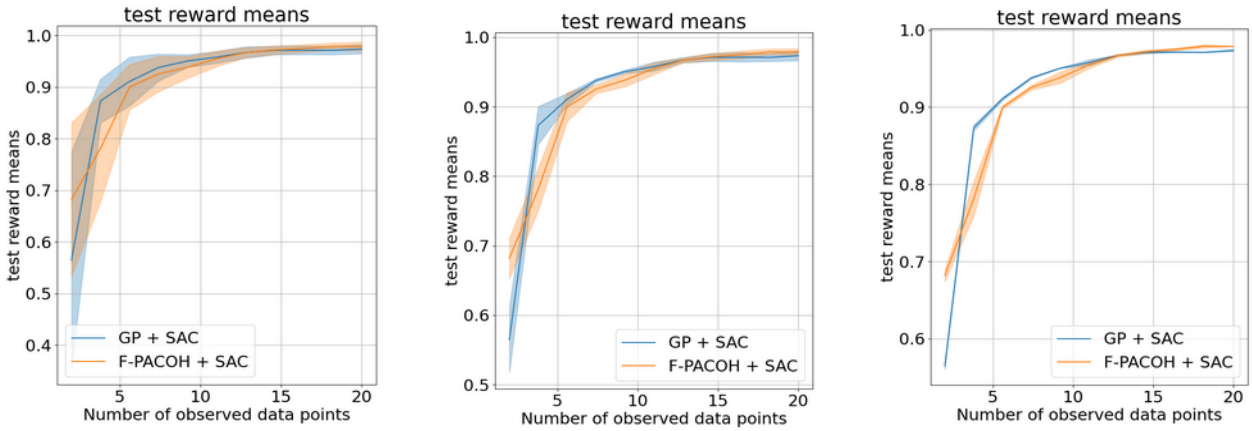


(a) A snapshot of the GP reward learner performance and SAC performance after 4 queries to the true reward function.



(b) A snapshot of the GP reward learner performance and SAC performance after 20 queries to the true reward function.

Figure 4: A visualization of the GP reward-learner performance and the SAC agent performance. Figure 4a shows the state after the reward model queried the true reward function 4 times, and figure 4b shows the state after the reward model queried the true reward function 20 times. The first column displays the true reward function. The second column shows the UCB of the learned reward landscape of the Gaussian process. The black points are the positions which the GP queried to the true reward function. The red points are the 1000 points with the highest UCB value, which have already been visited by the SAC agent and can therefore be used for querying. The more red the points are, the higher their UCB. The third column displays the learned reward landscape of the GP reward learner, together with 5 rollouts of a learned SAC policy. Finally, the last column shows the UCB of the learned reward function and the entire learned SAC policy in form of a vector field.



(a) Impact of environment noise (b) Impact of RL noise (c) Impact of reward model noise

Figure 5: Plots depicting the impact of the 3 sources of randomness on the total standard deviation. The plots were made with experiment data of the 2D environment.

quickly learn the reward landscape due to the meta-learning. We note that meta-learning is more likely to generalize to more complex environments than simply tuning the hyperparameters of the GP. This is mainly due to the larger expressivity of F-PACOH which lets one learn non-linear embeddings via neural networks. Therefore, we expect the advantages of meta-learning to become perceivable for more complex environments.

Analysis of standard deviation The main experiment was run for 50 different random seeds. The whole reward learning pipeline contains three different sources of randomness: **1)** The environment (different seeds lead to different environments being sampled), **2)** the reinforcement learning agent (the SAC algorithm is inherently stochastic), and **3)** the reward learner (e.g. during meta-learning of F-PACOH). In order to analyze the contribution of each individual source of randomness to the whole variation, we fix the randomness of one of the three sources, and let the other two sources vary. The randomness can be fixed by setting individual seeds for the three modules responsible for the randomness. We repeat this process several times for each source and then average the other sources of randomness out. The results for the 2D environment can be seen in figure 5. The results for the 3D environment are similar and can be seen in figure 11 on page 16. These figures show clearly that the main source of randomness comes from the different environments, whereas the reinforcement learning agent and the reward model contribute little towards the total variation.

 # Samples required to learn the reward landscape Figure 3 indicates that it takes the two agents (GP + SAC and F-PACOH + SAC) longer to learn the reward landscape of the 3D environment than it takes them to learn the 2D environment. This is expected, since the 3D environment is larger due to the extra dimension. During the learning phase the learned reward landscapes can become quite non-convex. This is because we use the upper confidence bound of the reward landscape to encourage exploration. As a side effect, this automatically leads to local minima around the points with high certainty (i.e the points which the reward model used to query to the true reward function). We notice that for the 3D reward landscape these non-convexities are stronger than in the 2D case. As a consequence, the reinforcement learning agent has a harder time to converge to the true optimum. We visualize one such case of non-convex learned reward

landscapes in figures 9 and 10 on pages 15 and 16.

4 Discussion

4.1 Issues

Working on combining meta-learning with reward learning presented us with several challenges. In the following we briefly present the two biggest difficulties we faced.

Numerical instabilities We use Gaussian processes as base learners for F-PACOH. During meta-learning F-PACOH learns an input embedding and the parameters of the Gaussian process in a data-driven fashion using the meta-training data. We faced the issue that since we used very little meta-training data, the learned parameters, especially the kernel matrix of the GP were numerically unstable. As a consequence, we either suffered from a dramatic increase in runtime, or the reward learner was never able to learn the reward landscape. We tried to apply several methods to remedy this problem, with limited success. For a more thorough analysis of this problem we refer the interested reader to section B in the appendix.

Complex dependencies between the reward learner and the reinforcement learning agent In our experiment the reward learner is dependent on the reinforcement learning agent, because it only queries the true reward function about positions which the RL agent has visited in a previous iteration. Likewise, the RL agent requires the reward learner to do a good job in learning the reward landscape, as otherwise it won't be able to train a good policy. This means that one cannot simply perform a hyperparameter search for the RL hyperparameters which yield the best performance, because one also needs to somehow incentivize the agent to thoroughly explore its environment. We tried several different methods to solve this problem. In the end we found the combination of two methods to work best. First, we don't train the RL agent directly on the learned reward landscape of the reward model, but on the upper confidence bound of the learned reward landscape. This automatically incentivizes the agent to explore regions which it rarely visited so far. Secondly, we make use of directed action noise, which adds directed noise to the actions of the agent, leading to a more elaborate exploration.

4.2 Limitations and future work

We are not able to demonstrate the superiority of meta reward learning over standard reward learning with our experiments, despite there being reasons to expect the contrary. The main cause of this is most likely the environment type we used. The target position environments are very simple. The single local and global optimum makes them easy to approximate with a simple multi-dimensional normal distribution. The advantage of meta-learned reward models is probably easier seen for more complex environment types. A natural next step would therefore be to extend the experiments to more complex, non-convex reward landscapes.

Another important next step is to alleviate the issue of numerical instability. There exist several methods to fix unstable or broken kernel matrices [12, 13] which might be helpful to tackle this issue.

5 Conclusion

In this report we studied the application of meta-learning to the problem of reward learning. We developed an experiment framework which allowed us to compare the performance of meta-learned reward models with standard Gaussian process based reward models. Our experiments showed that both types of reward learners yielded good results for simple environments. However, we were not able to show that meta-learned reward models are more performant than standard reward models. This might be because our environments are too simple and that the advantages of meta-learned reward models appear in more complex environments where simple reward models struggle.

6 Acknowledgement

I am very grateful to my two supervisors, David Lindner and Jonas Rothfuss for supporting me throughout this project. They provided me with invaluable advice and support without which I would not have been able to conduct this research project.

References

[1] Dan Hendrycks, Nicholas Carlini, John Schulman, and Jacob Steinhardt. Unsolved problems in ml safety. *arXiv preprint arXiv:2109.13916*, 2021.

[2] Christian Wirth, Riad Akrour, Gerhard Neumann, Johannes Fürnkranz, et al. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18 (136):1–46, 2017.

[3] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

[4] Rohin Shah, Dmitrii Krasheninnikov, Jordan Alexander, Pieter Abbeel, and Anca Dragan. Preferences implicit in the state of the world. *arXiv preprint arXiv:1902.04198*, 2019.

[5] Hong Jun Jeon, Smitha Milli, and Anca Dragan. Reward-rational (implicit) choice: A unifying formalism for reward learning. *Advances in Neural Information Processing Systems*, 33:4415–4426, 2020.

[6] Joaquin Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

[7] Yunxiao Qin, Weiguo Zhang, Chenxu Zhao, Zezheng Wang, Xiangyu Zhu, Jingping Shi, Guojun Qi, and Zhen Lei. Prior-knowledge and attention based meta-learning for few-shot learning. *Knowledge-Based Systems*, 213:106609, 2021.

[8] Jonas Rothfuss, Vincent Fortuin, Martin Josifoski, and Andreas Krause. Pacoh: Bayes-optimal meta-learning with pac-guarantees. In *International Conference on Machine Learning*, pages 9116–9126. PMLR, 2021.

[9] Jonas Rothfuss, Dominique Heyn, Andreas Krause, et al. Meta-learning reliable priors in the function space. *Advances in Neural Information Processing Systems*, 34, 2021.

[10] Jonas Rothfuss Lukas Fluri, David Lindner. Meta active reward learning. https://github.com/david-lindner/meta-active-reward-learning, 2022.

[11] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[12] Nicholas J Higham. Computing a nearest symmetric positive semidefinite matrix. *Linear algebra and its applications*, 103:103–118, 1988.

[13] Nicholas J Higham. Computing the nearest correlation matrix—a problem from finance. *IMA journal of Numerical Analysis*, 22(3):329–343, 2002.

Parameter	2D Environment	3D Environment
Buffer size	110k	110k
learning rate	0.001	0.001
tau	0.01	0.005
gamma	0.99	0.99
train frequency	64	64
gradient steps	64	64
learning start	1000	1000
network architecture	2×256	2×256
action noise type	Ornstein-Uhlenbeck	Ornstein-Uhlenbeck
action noise mean	0	0
action noise std	0.05	0.1
Total training timesteps	100k	100k

Table 1: The hyperparameters of the SAC agent

Parameter	2D Environment	3D Environment
kernel variance	0.5	0.8
kernel lengthscale	0.4	1.5
likelihood std	0.01	0.0001

Table 2: The hyperparameters of the GP reward learner

A Experiment parameters

All parameters used for the different experiments can be found in tables 1, 2, and 3. All parameters which are not explicitly stated were left on their default value.

B Numerical instabilities

As mentioned in section 4.1, when training F-PACOH we faced the issue that the kernel matrix of the learned GP base learner was numerically unstable. This lead to either performance issues or extremely long experiment runtime. The main difficulty is that one does not have direct influence on the covariance matrix generation. The situation is depicted in figure 6.

Parameter	2D Environment	3D Environment
weight decay	0.001	0.0001
# training steps	10k	10k
learning rate	0.001	0.001
learning rate decay	0.9999	1
prior lengthscale	0.5	0.7
prior outputscale	0.5	0.5
# KL approximation samples	20	20
Prior factor	0.05	0.1

Table 3: The hyperparameters of the F-PACOH reward learner

As a user we are able to select hyperparameters for F-PACOH, as well as the meta-training data. During meta-learning, we train neural networks for the kernel map, as well as the mean function of the Gaussian process. Unfortunately, the learned mappings seem to have undesirable properties, since the resulting covariance matrices can be numerically unstable. More precisely, the covariance matrix might not be symmetric (rarely) or not positive definite (often). Unfortunately, one can't change these mappings directly, because they get learned via neural networks.

Given the potentially broken covariance matrices, there are two options of how to handle them. Either one leaves them as they are and lets GPytorch, the library we use for implementing our Gaussian processes, handle the instabilities internally. This is possible, but results in an extreme increase of runtime, especially for making predictions with the learned Gaussian processes. See figure 7 for an example how the runtime of the experiment deteriorates. In each experiment we train 10 different SAC policies. Between two consecutive training steps we let the reward learner query the true reward function twice. One can see that especially in the beginning, after having made very few queries the SAC training time increases drastically.

The other option is to try to manually fix the covariance matrix. We tried to fix broken covariance matrices K via the following transformations:

$$\begin{aligned} \text{non-symmetric:} \quad & K \;\to\; \frac{K + K^T}{2} \\ \text{not positive definite:} \quad & K \;\to\; K + \epsilon \cdot \mathcal{I} \end{aligned}$$

for some small ϵ. While this fixed the covariance matrices, it sometimes lead to bad performance because we had to add quite a bit of jitter to the diagonal which falsified the original covariance matrix heavily.

C Further plots

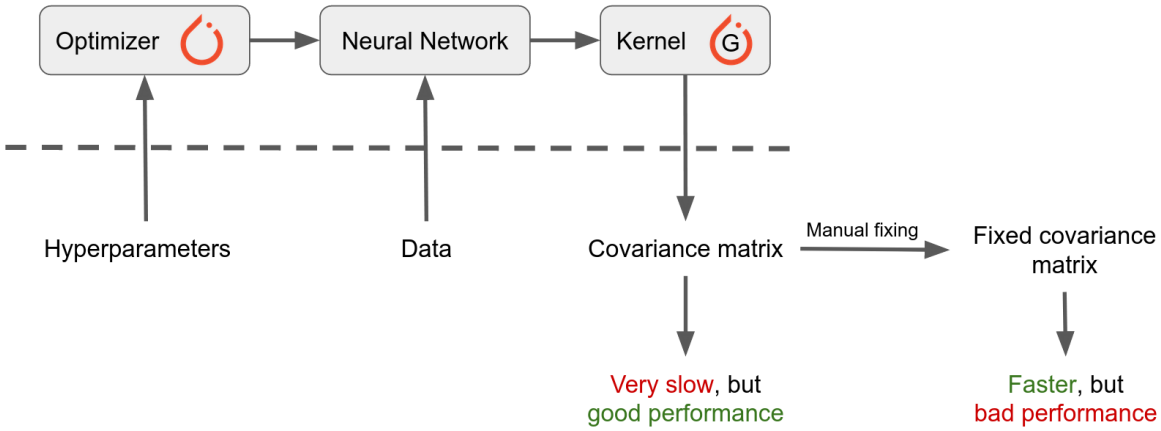


Figure 6: A diagram depicting the different sources of the numerical instabilities. Elements below the dashed line can be directly influenced by a user of the pipeline, whereas elements above the line are only indirectly accessible.

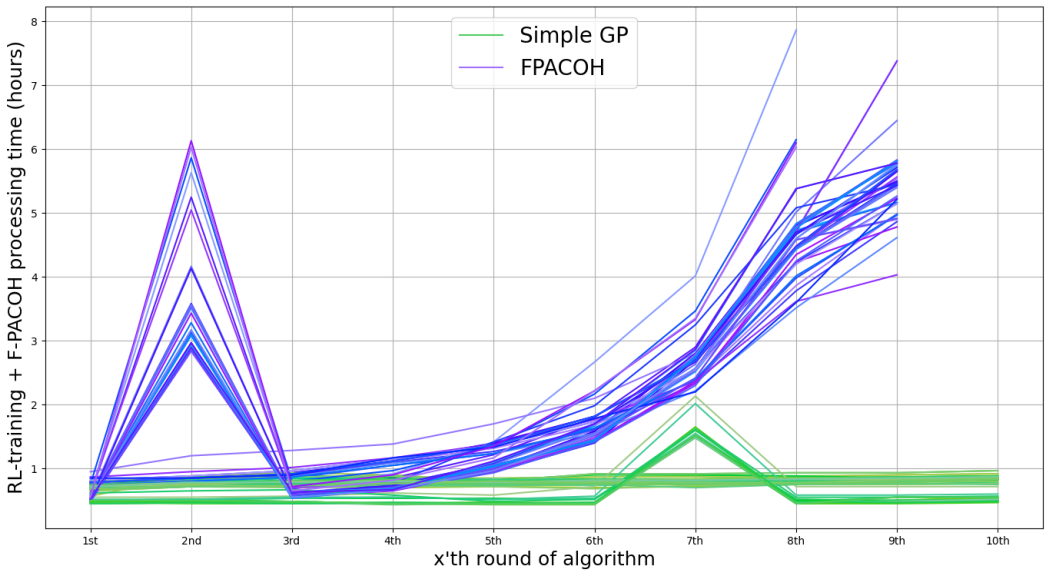
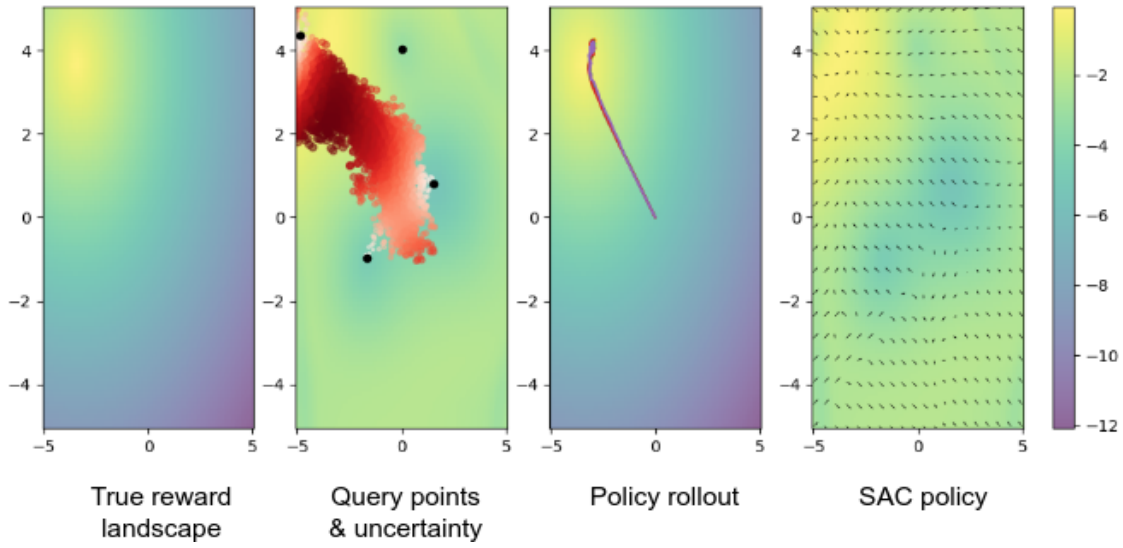
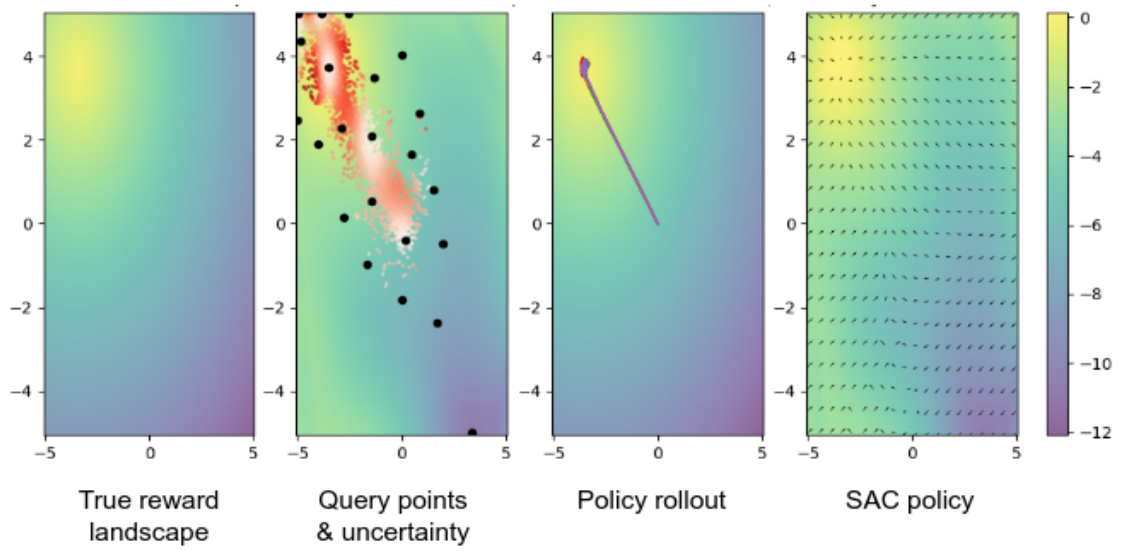


Figure 7: An example of an experiment run, where the runtime of the F-PACOH runs deteriorated due to numerical instabilities. Each experiment consists of 10 rounds where in each round the reward model performs two queries to the true reward function, and then an SAC policy is trained on the updated reward model. The plot displays the runtime of each individual round.



(a) A snapshot of the F-PACOH reward learner performance and SAC performance after 4 queries to the true reward function.



(b) A snapshot of the F-PACOH reward learner performance and SAC performance after 20 queries to the true reward function.

Figure 8: A visualization of the F-PACOH reward-learner performance and the SAC agent performance. Figure 8a shows the state after the reward model queried the true reward function 4 times, and figure 8b shows the state after the reward model queried the true reward function 20 times. The first column displays the true reward function. The second column shows the UCB of the learned reward landscape of the F-PACOH reward model. The black points are the positions which the reward model queried to the true reward function. The red points are the 1000 points with the highest UCB value, which have already been visited by the SAC agent and can therefore be used for querying. The more red the points are, the higher their UCB. The third column displays the learned reward landscape of the F-PACOH reward learner, together with 5 rollouts of a learned SAC policy. Finally, the last column shows the UCB of the learned reward function and the entire learned SAC policy in form of a vector field.

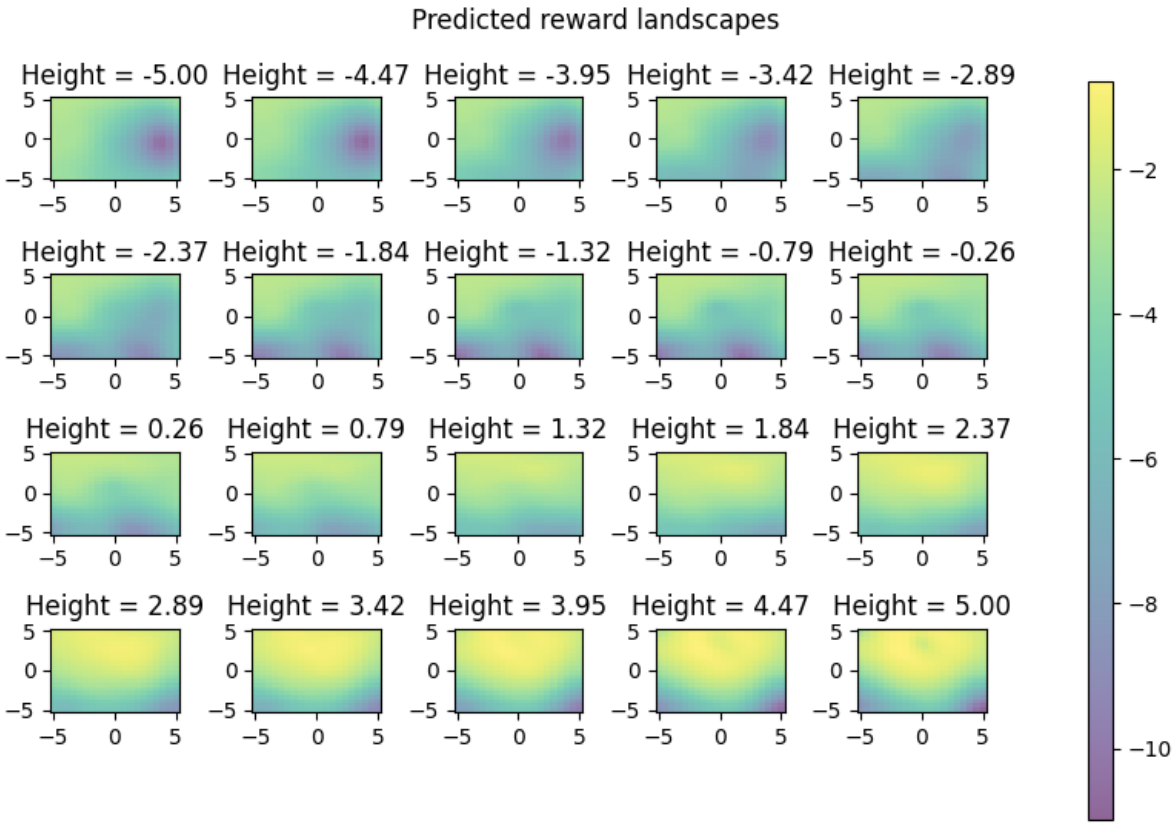


Figure 9: The learned 3D reward landscape of the F-PACOH reward learner after having queried the true reward function for 6 points. The 3D environment is sliced into 25 2D slices. In several of the slices, the high non-convexity of the reward landscape is visible

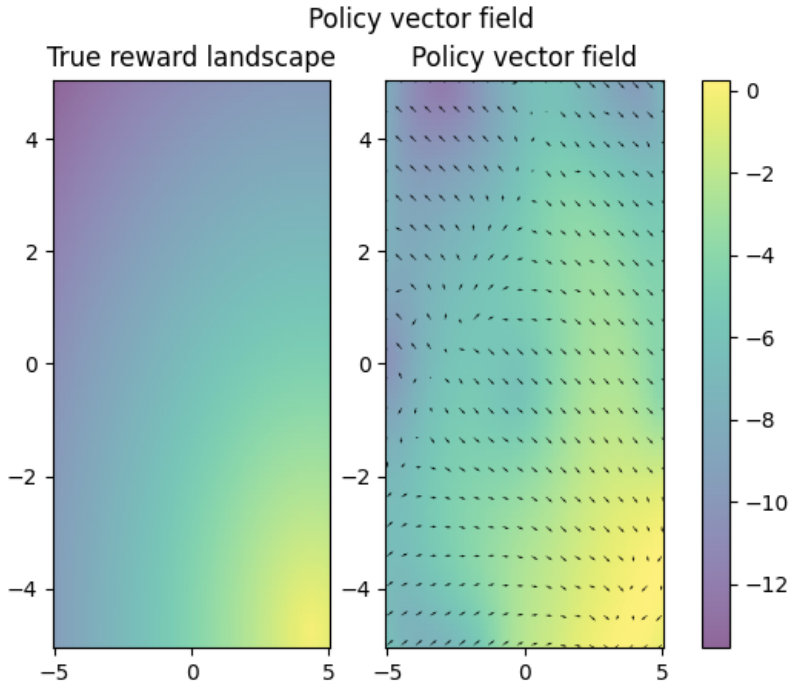
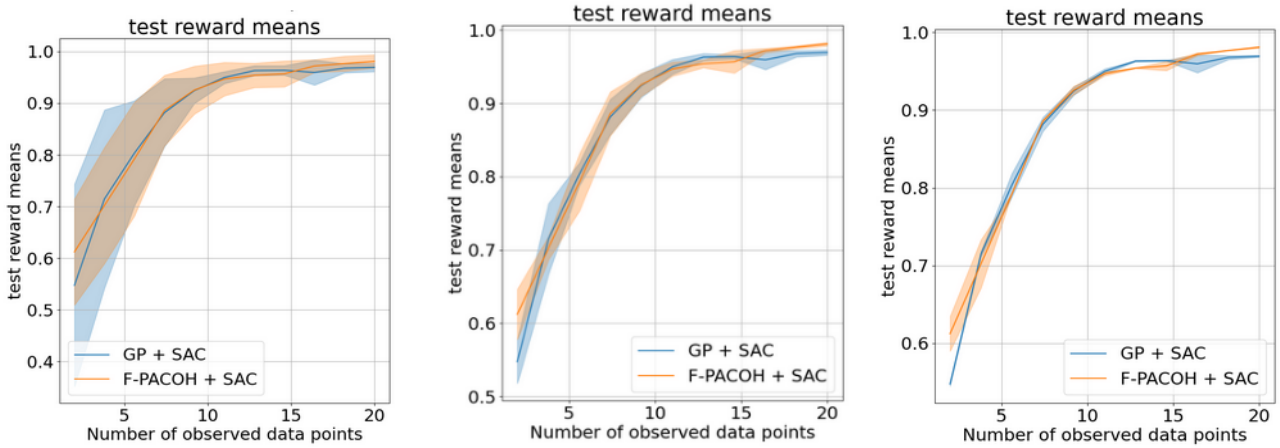


Figure 10: The learned 2D reward landscape of the F-PACOH reward learner after having queried the true reward function for 6 points. The learned reward landscape has almost converged to the true reward landscape.



(a) Impact of environment noise (b) Impact of RL noise (c) Impact of reward model noise

Figure 11: Plots depicting the impact of the 3 sources of randomness on the total standard deviation. The plots were made with experiment data of the 3D environment.