[Replication] A Meta-MDP Approach to Exploration for Lifelong Reinforcement Learning

David Cabatingan[*]
Department of Computer Science
Brown University

Kendrick Cole[*]
Department of Computer Science
Brown University

Natalie Delworth[*]
Data Science Initiative
Brown University

Petar Peshev[*]
Department of Computer Science
Brown University

Abstract

In *A Meta-MDP Approach to Exploration for Lifelong Reinforcement Learning* by Garcia and Thomas [4], the authors propose an improved method of optimizing exploration in reinforcement learning agents by creating an 'advisor agent' that solves its own MDP over distributions of tasks using meta-learning approaches. We describe our efforts in replicating the results for the REINFORCE algorithm using an advisor policy in the CartPole environment. We conclude that the strategy is highly sensitive to variations in learnable environments, hyperparameters, and training details, as we are unable to produce results that mirror the conclusions of the paper.

1 Introduction

Humans have the ability to have a problem or task described to them and then leverage their prior knowledge about similar tasks or problems to allow them to perform the new task well. Reinforcement learning methods typically lack this ability, as they are trained from a *tabula rasa* for each task. They lack the priors that humans are able to use.

When given a task to learn, it is common for a reinforcement learning agent to have an exploration vs. exploitation trade-off as it learns the behaviors of the environment it is in while also trying to maximize reward. That is, it will both have to take actions it believes grants it high reward while also taking actions it does not have enough information about in order to explore its environment. This trade-off can be explicit (as with Q-Learning [9]) or implicit (as with RMax [2]), but nevertheless exists in all domains.

One question that this brings up is: when an agent is exploring, how should it most effectively do so? In this paper, we will attempt to reproduce the results of Garcia and Thomas [4] (henceforth, the authors) in their paper *A Meta-MDP Approach to Exploration for Lifelong Reinforcement Learning*.

They formalize the notion of what it means to learn an optimal exploration policy, and then give a meta-learning approach where an agent is told how to explore by an advisor agent which learns over many lifetimes of exploiter agents. We focus on the reproducibility of their empirical results on learning optimal exploration policies in the CartPole problem domain [3].

[*]Equal contribution by all authors, ordered by last name.

Preprint. Under review.

2 Background

2.1 Meta-Learning

The problem of meta-learning is a problem of *learning about learning*. As discussed by Vilalta and Drissi [8] and Bertinetto et al. [1], meta-learning is a process where there is a base-learner and meta-learner, and the meta-learner's objective is to maximize the performance of the base-learner. The specifiec model used by the authors of the paper is one where the meta-learner, the advisor, is used to dictate the exploration policy of the base-learner, the exploiter.

2.2 ϵ-greedy Exploration

ϵ-greedy Learning is a standard exploration model used where an agent explores with probability $\epsilon \in [0, 1]$, and otherwise takes the reward maximizing action. It is currently used in many contexts (SARSA [7], Q-Learning [9], and DQN [6]). This is the model that the authors use to base their meta-learner off of, where the meta-learner replaces the ϵ-greedy exploration and dictates how the base-learner should behave in order to best explore.

2.3 Meta-MDPs

The authors describe a framework for meta-learning that allows for the incorporation of known results about solving Markov Decision Processes (MDPs). They formulate the problem of learning how to advise a sequence of agents attempting to solve MDPs over variations on an environment as being itself an MDP, therefore constructing an MDP in an MDP, a meta-MDP.

3 Analysis

In their paper, Garcia and Thomas [4] present a new approach to meta-learning that involves using the meta-learner to only influence the exploration of the base-learner, thus creating a meta-learner (advisor) that will advise the base-learner on how to explore its environment, but otherwise let the base-learner learn the environment on its own. For the purposes of reproducing their results, we will focus on their results obtained using REINFORCE [10]. The implementation of that approach is captured in the following pseudocode:

Algorithm 1: Garcia and Thomas [4]'s Agent and Advisor REINFORCE implementation

```
Initialize advisor policy $\mu$ at random;
for $i_{meta} \in [0, I_{meta}]$ do
    Sample a task $t_{i_{meta}}$ from task distribution;
    $\epsilon := 0.8$;
    for $i \in [0, I]$ do
        Initialize exploiter policy $\pi$ at random;
        for $t \in [0, T]$ do
            Sample action $a_t$ from $\mu$ with probability $\epsilon$, else sample from $\pi$;
            Take action $a_t$ and get reward $r_t$ at state $s_t$;
        end
        $\epsilon := 0.995 * \epsilon$;
        Update policy $\pi$ with $(s_t, a_t, r_t)$ for each $t \in [0, T]$;
    end
    if $i_{meta}\%5 == 4$ then
        Update policy $\mu$ with $(s_t, a_t, r_t)$ for each $t \in [0, 5IT]$;
    end
end
```

with the values $I_{meta} = 500, I = 1000, T = 1000, \gamma = 0.99$

4 Replication

Our focus was to replicate a particular empirical result from the paper. We chose to focus on the results described in Section 6.1 of the original paper. To this end, we implemented two architectures that solve the CartPole environment provided by OpenAI, including the meta-MDP framework. In this section, we describe the process of implementation; in Section 5 we describe our results. In translating the pseudocode for Algorithm 1 (as provided in the paper) to a real model, it was necessary to make decisions that were not described in the paper. We have included some of the details in our above pseudocode for our implementation. Particularly, we had problems with replication owing to training details, architectures, and possible issues with the CartPole environment.

4.1 Architecture

An important component of any deep learning model is network architecture. The REINFORCE algorithm is largely agnostic to the particular model used to calculate policy, simply giving an update rule for its parameters. We chose to initially implement the REINFORCE advisor/exploiter framework using a neural network with a hidden layer of size 128 for both advisor and exploiter as used in the PyTorch REINFORCE example implementation for CartPole [2]. We later experimented with differing values for the hyperparameters (architecture, γ for discounted reward, learning rates). Altering these hyperparameters led to changes in the performance of the exploiters which we cover in the next section. An issue we encountered with this method of solving the environments is that often, the agents learned too quickly to allow the effectiveness of the advisor to be shown at all.

After correspondence with the authors of the original paper to clarify implementation details, we learned that their implementation was based on a linear function, with features represented via the Fourier basis as described by Konidaris et al. [5] which was not mentioned in the original paper. In light of this, we also experimented with using a linear function on the features after transforming them into the Fourier basis.

4.2 Training Details

We found a significant challenge in tuning the model to work well. We experimented with large ranges of hyperparameters and varying network architectures as described above. We reached out to the authors to discuss what details we may have missed when we were unable to find success in training an advisor.

Through correspondence with the authors, we learned that they were updating the advisors parameters after multiple meta-iterations, which was not explicitly stated in the pseudocode of the algorithm. The purpose of this modification is to reduce the variance in gradient estimates, allowing for more stable updates. This is critical to consistent training, since the amount of noise introduced by action selection and variance in environment parameters is significant. Beyond the high variance typically present in the domain, the advisor generally receives rewards based on actions which it recommended, but were not carried out, since the agent only takes the recommended action when exploring.

4.3 The CartPole Environment

The CartPole environment is a standard environment from control theory. With the default physics parameters of the OpenAI Gym implementation of CartPole, there are a variety of algorithms that are known to achieve optimal performance on the environment within several hundred training episodes. But less is known about the effect that modifications to the environment have on the ability of agents to solve the problem.

For the purposes of replication, we modified the environment as described in the original paper, changing the length and mass of the pole, mass of the cart, and magnitude of the force applied by the agent. However, the boundary between sets of environment parameters that are solvable and ones that are not is unclear. We experimented with a variety of ranges for each parameter, and settled on values that gave environments where agents were likely to have at least some level of success in learning over time.

[2] `github.com/pytorch/examples/blob/master/reinforcement_learning/reinforce.py`.

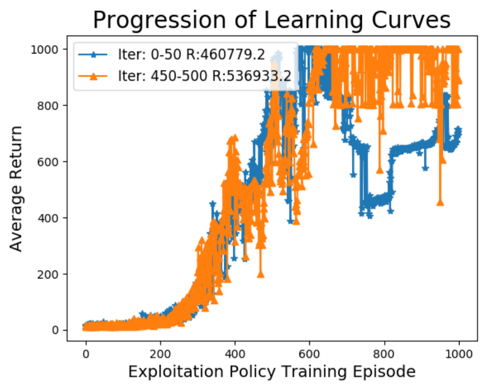

Figure 1: Average training curves during the first 50 and last 50 meta-iterations with neural network. Figure 3(b) from original paper.

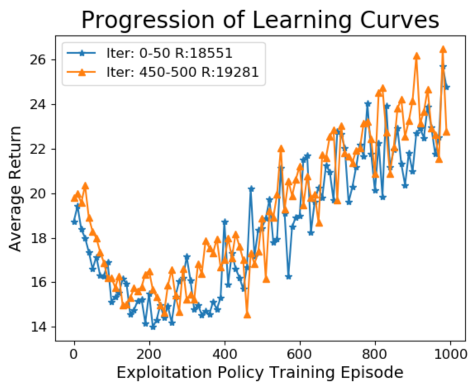

Figure 2: Average training curves during the first 50 and last 50 meta-iterations with Fourier basis. Figure 3(b) from original paper.

We also corresponded with the authors to clarify how the sampling of different tasks was implemented, which is not specified in the paper, and learned that they experimented to find the bounds on which CartPole tasks are learnable, and then sampled uniformly from the range of learnable tasks. However, we did not obtain any exact bounds on environment parameters.

5 Results

5.1 Training Times

We trained the models on a Google Cloud Platform[3] virtual machine. We found that the memory requirement was significant, choosing to use 40 GB of memory to run several training loops simultaneously. Given the small size of the models used to solve the environment, we chose to forego using a GPU to train, finding that using a GPU actually slowed down training. This is likely because the model was small, so the gain in computational speed was not worth the overhead of the GPU. Instead, we trained on a VM with 14 vCPUs. Training times depended on the size of the agent models and their performance - better-performing agents average more steps per episode, taking up to 10 times longer on average. In our implementation, we found the fastest training times were about 1 minute per meta-iteration (1000 iterations of the base-learner), and the slowest training times were 10 minutes per meta-iteration.

5.2 Empirical Results

We encountered difficulty in implementing an advisor agent that was able to achieve the high performance described in the original paper across both the neural network and Fourier basis implementations of the model and a large range of hyperparameters. Our advisor model was able to achieve only marginal gains over a random exploration policy.

While working with our hyperparameter tuning, we focused on replicating the behavior found in Figure 3(b) from the original paper, before running a random agent for comparison to create our version of Figure 3(a) from the original paper. Figure 3(b) from the original paper shows two plots on the chart: the return for each iteration of the base-learner averaged over the first 50 meta-iterations and the return for each iteration of the base-learner averaged over the last 50 meta-iterations. We used this chart to evaluate if training the advisor was making a difference in the performance of the base-learner.

When tuning our REINFORCE implementation using the neural network, the best performance we were able to achieve can be seen in Figure 1. Notice that we were able to tune the base-learner agent to be able to achieve an average reward of up to 536 per iteration while the implementation in the original paper only achieved an average reward of about 30, yet there was still no significant change

[3]`https://cloud.google.com`

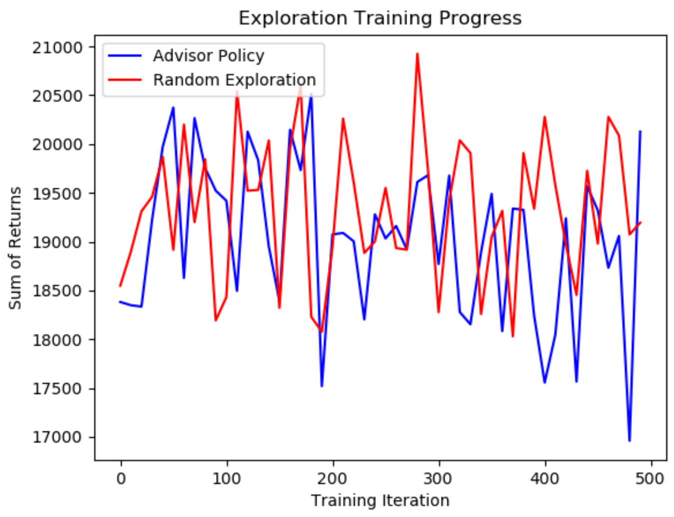

Figure 3: Performance during training, comparing advisor policy to random exploration policy. Figure 3(a) from original paper.

from the beginning of meta-training to the end of meta-training. Thus we can conclude we were unable to tune the advisor to be useful.

Figure 2 shows the same data plotted for our best-tuned Fourier basis implementation. Notice that the average return over the iterations now much more closely matches the plots from the paper, but there is still no substantial gain in base-learner performance over the training of the advisor. From this, it is clear that the performance gains given by the advisor over time were yet again not as significant in our experiments as in the ones given by the paper. The most clear indicator of this is the shape of the average learning curve over the last 50 iterations. In our chart, it is nearly identical to the curve of the first 50 iterations, albeit with a constant offset. On the other hand, the shape of the last 50 iterations in the original paper is significantly different from that of the first 50, representing a shift in *how* the base-learners progress in their learning. Given this, we are not able to conclude that the results of the paper have been fully replicated.

However, we believe that an advisor which achieves more significant gains in training exploiters is not out of reach. From the process of replication, it is clear that small modifications to a variety of parameters of the training process can have a significant impact on results, and this approach of training the advisor for learning an exploration policy is very sensitive to hyperparameters.

Figure 3 shows the sum of returns across the entire lifetime of the advisor agent, and for reference compares it to the sum of returns for a random agent. This is our recreation of their original Figure 3(a), and we can see that learned advisor performance is very similar to that of using a random advisor.

6 Conclusion

Through all of the testing, we found it very difficult to adapt our parameters and model to one that would allow the advisor policy to properly learn any useful general information regarding how to explore the CartPole environment. The biggest suspicion we have for this is that, in the neural network case, the CartPole environment is easy enough to learn that even with the noise of an advisor agent, the exploiter policy still managed to learn an effective policy to play the environment. The rewards of the Fourier basis experiments resembled those of the paper, but the hyperparameters still seemed to be sensitive enough to not result in sufficient learning on the side of the advisor. For future efforts to verify the results of the original paper, we would advise more challenging environments (including the continuous ones explored in the paper) and potentially more tuning of hyperparameters that will allow for effective learning of the advisor's policy.

Acknowledgments

We thank Francisco Garcia for providing invaluable information and advice throughout the course of this project. We thank Jules Becker for his help as a shepherd TA.

References

[1] L. Bertinetto, J. F. Henriques, P. H. S. Torr, and A. Vedaldi. Meta-learning with differentiable closed-form solvers. *CoRR*, abs/1805.08136, 2018. URL `http://arxiv.org/abs/1805.08136`.

[2] R. I. Brafman and M. Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3:213–231, Mar. 2003. ISSN 1532-4435. doi: 10.1162/153244303765208377. URL `https://doi.org/10.1162/153244303765208377`.

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[4] F. M. Garcia and P. S. Thomas. A meta-mdp approach to exploration for lifelong reinforcement learning. *CoRR*, abs/1902.00843, 2019. URL `http://arxiv.org/abs/1902.00843`.

[5] G. Konidaris, S. Osentoski, and P. Thomas. Value function approximation in reinforcement learning using the Fourier basis. pages 380–385, August 2011. URL `http://lis.csail.mit.edu/pubs/konidaris-aaai11a.pdf`.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, 02 2015. doi: 10.1038/nature14236.

[7] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

[8] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18, 09 2001. doi: 10.1023/A:1019956318069.

[9] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL `https://doi.org/10.1007/BF00992698`.

[10] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3-4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL `https://doi.org/10.1007/BF00992696`.