Some Considerations on Learning to Explore via Meta-Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

We consider the problem of exploration in meta reinforcement learning. Two new meta reinforcement learning algorithms are suggested: E-MAML and E-RL^2. Results are presented on a novel environment we call 'Krazy World' and a set of maze environments. We show E-MAML and E-RL^2 deliver better performance on tasks where exploration is important.

1 Introduction

Supervised learning algorithms typically have their accuracy measured against some held-out test set which did not appear at training time. A supervised learning model generalizes well if it maintains its accuracy under data that has non-trivial dissimilarity from the training data. This approach to evaluating algorithms based on their generalization ability contrasts the approach in reinforcement learning (RL), wherein there is usually no distinction between training and testing environments. Instead, an agent is trained on one environment, and results are reported on this same environment. Most RL algorithms thus favor mastery over generalization.

Meta RL is the suggestion that RL should take generalization seriously. In Meta RL, agents are evaluated on their ability to quickly master new environments at test time. Thus, a meta RL agent must not learn how to master the environments it is given, but rather it must *learn how to learn* so that it can quickly train at test time. Recent advances in meta RL have introduced algorithms that are capable of generating large policy improvements at test time with minimal sample complexity requirements Finn et al. (2017); Duan et al. (2016); Wang et al. (2016).

One key question for meta RL that has hitherto been neglected in the literature is that of exploration. A crucial step in learning to solve many environments is an initial period of exploration and system identification. Furthermore, we know that real-life agents become better at this exploratory phase with practice. Consider, for example, an individual playing an entirely new video game. They will first need to identify the objective of the game and its mechanics. Further, we would expect that individuals who have played many video games would have a significantly easier time learning new games. Similarly, we would expect a good meta RL agent to become more efficient at this exploratory period. Unfortunately, we have found existing algorithms deficient in this area. We hypothesize that this failure can at least partially be attributed to the tendency of existing algorithms to do greedy local optimizations at each step of meta-training, as discussed further below.

To begin addressing the problem of exploration in meta RL, we introduce two new algorithms: E-MAML and E-RL^2. It should come as no surprise that these algorithms are similar to their respective namesakes MAML and RL^2. The algorithms are derived by reformulating the underlying meta-learning objective to account for the impact of initial sampling on future (post-meta-updated) returns. We show that our algorithm achieves better results than MAML and RL^2 on two environments: a Krazy World environment we developed to benchmark meta RL, and a set of maze environments. Code for these benchmark environments, as well as code for new and faster implementations of E-MAML, MAML, E-RL^2, and RL^2 will be released upon this paper's publication.

2 Related Work

Recently, a flurry of new work in Deep Reinforcement Learning has provided the foundations for tackling RL problems that were previously thought intractable. This work includes: 1) Mnih et al.

(2015; 2016), which allow for discrete control in complex environments directly from raw images. 2) Schulman et al. (2015); Mnih et al. (2016); Schulman et al. (2017); Lillicrap et al. (2015), which have allowed for high-dimensional continuous control in complex environments from raw state information.

Although these algorithms offer impressive capabilities, they still often falter when faced with environments where exploration presents a significant challenge. This is not surprising, as there exists a large body of RL work addressing the exploration problem (Kearns & Singh, 2002; Brafman & Tennenholtz, 2002; Kolter & Ng, 2009). In practice, these methods are often not used due to difficulties with high-dimensional observations, difficulty in implementation on arbitrary domains, and lack of promising results. This resulted in most deep RL work utilizing epsilon greedy exploration (Mnih et al., 2015), or perhaps a simple scheme like Boltzmann exploration (Carmel & Markovitch, 1999). As a result of these shortcomings, a variety of new approaches to exploration in deep RL have recently been suggested (Tang et al., 2016; Houthooft et al., 2016; Stadie et al., 2015; Osband et al., 2016; Bellemare et al., 2016; Ostrovski et al., 2017). Further, there has been extensive research on the idea of artificial curiosity. For example Ngo et al. (2012); Graziano et al. (2011); Schmidhuber (1991; 2015); Storck et al. (1995); Sun et al. (2011) all deal with teaching agents to generate a curiosity signal that aids in exploration. Although this line of work does not explicitly deal with exploration in meta learning, it remains a large source of inspiration for this work. In spite of the numerous cited efforts in the above paragraph, the problem of exploration in RL remains difficult.

This paper will consider the problem of exploration in meta RL. However, it is important to note that many of the problems in meta RL can alternatively be addressed with the field of hierarchical reinforcement learning. In hierarchical RL, a major focus is on learning primitives that can be reused and strung together. These primitives will frequently enable better exploration, since they'll often relate to better coverage over state visitation frequencies. Recent work in this direction includes (Vezhnevets et al., 2017; Bacon & Precup, 2015; Tessler et al., 2016; Rusu et al., 2016). The primary reason we consider meta learning over hierarchical RL is that we find hierarchical RL tends to focus more on defining specific architectures that should lead to hierarchical behavior, whereas meta learning instead attempts to directly optimize for these behaviors.

As for meta RL itself, the landscape is much less mature than deep RL and hierarchical RL. As a consequence, the available methods are more limited. To the best of our knowledge, there does not exist any literature addressing the topic of exploration in meta RL. This paper will seek to begin that discussion by considering E-MAML and E-RL^2, which build upon the existing meta learning algorithms MAML (Finn et al., 2017) and RL^2 (Duan et al., 2016).

In MAML, tasks are sampled and a policy gradient update is computed for each task with respect to a fixed initial set of parameters. Subsequently, a meta update is performed where a gradient step is taken that moves the initial parameter in a direction that would have maximally benefited the average return over all of the sub-updates. Results are presented on an environment wherein a simulated ant must determine its goal direction, which changes each time a new task is sampled. In RL^2, an RNN is made to ingest multiple rollouts from many different MDPs and then perform a policy gradient update through the entire temporal span of the RNN. The hope is that the RNN will learn a faster RL algorithm in its memory weights.

3 Reinforcement Learning Notation

Let $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma, T)$ represent a discrete-time finite-horizon discounted Markov decision process (MDP). The elements of M have the following definitions: $\mathcal{S}$ is a state set, $\mathcal{A}$ an action set, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ a transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ a reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}_+$ an initial state distribution, $\gamma \in [0, 1]$ a discount factor, and T the horizon. We will sometimes speak of M having a loss function $\mathcal{L}$ rather than reward function r. All we mean here is that $\mathcal{L}(s) = -r(s)$

In a classical reinforcement learning setting, we optimize to obtain a set of parameters θ which maximize the expected discounted return under the policy $\pi_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$. That is, we optimize to obtain θ that maximizes $\eta(\pi_\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} \gamma^t r(s_t)]$, where $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi_\theta(a_t|s_t)$, and $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$.

4 Problem Formulation and Algorithms

4.1 The Meta Reinforcement Learning Objective

In meta reinforcement learning, we consider a family of MDPs $\mathcal{M} = \{M_i\}_{i=1}^N$ which comprise a distribution of tasks. The goal of meta RL is to find a policy π_θ and paired update method U such that, given $M_i \sim \mathcal{M}$, we have $\pi_{U(\theta)}$ solves M_i quickly. The word quickly is important here, so let us elaborate. By quickly, we mean orders of magnitude more sample efficient than simply solving M_i with policy gradient or value iteration methods. In fact, an ideal meta RL algorithm would solve these tasks by collecting 1-10 trajectories from the environment for tasks where policy gradients require 100,000 or more trajectories. The thinking here goes that if an algorithm can solve a problem with so few samples, then it might be 'learning to learn.' That is, the agent is not learning how to master a given task but rather how to quickly master new tasks. We can write this objective cleanly as

$$\min_\theta \sum_{M_i} \mathbb{E}_{(\pi_{U(\theta)})} \left[\mathcal{L}_{M_i}\right] \tag{1}$$

Note that this objective is similar to the one that appears in MAML Finn et al. (2017), which we will discuss further below.

4.2 E-MAML

We can expand the expectation from (1) into the integral

$$\int_{\tau \sim \pi_{U(\theta)}} R(\tau)\pi_{U(\theta)}(\tau) d\tau \tag{2}$$

The objective (1) can be optimized by taking a derivative of this integral with respect to θ and carrying out a standard REINFORCE style analysis to obtain a tractable expression for the gradient Williams (1992).

One issue with this direct optimization is that it will not account for the impact of the original sampling distribution π_θ on the future rewards $R(\tau), \tau \sim \pi_{U(\theta)}$. We would like to account for this, because it would allow us to weight the initial samples $\bar{\tau} \sim \pi_\theta$ by the expected future returns we expect after the sampling update $R(\tau)$. We argue that optimizing for this impact will make the resulting policy more exploratory. This is because when the dependence is not accounted for, π_θ will collect samples that lead to the largest local improvement after $U(\theta)$. However, when the dependence is accounted for, π_θ will be encouraged to also collect samples that lead to good meta-updates. Including this dependency can be done by writing the modified expectation as

$$\int_{\tau \sim \pi_{U(\theta)}} \int_{\bar{\tau} \sim \pi_\theta} R(\tau)\pi_{U(\theta)}(\tau)\pi_\theta(\bar{\tau}) d\tau d\bar{\tau} \tag{3}$$

Note that one could certainly argue that this is the correct form of (2), and that the failure to include the dependency above makes (2) incomplete. We choose to view both as valid approaches, with (3) simply being the more exploratory version of the objective for reasons discussed above.

In any case, we find ourselves wishing to find a tractable expression for the gradient of (3). This can be done quite smoothly by applying the product rule under the integral sign and going through the REINFORCE style derivation twice to arrive at a two term expression, one of which encourages exploitation and one of which encourages exploration.

$$\begin{aligned}
&\frac{\partial}{\partial \theta}\int_{\tau\sim\pi_{U(\theta)}}\int_{\bar{\tau}\sim\pi_\theta} R(\tau)\pi_{U(\theta)}(\tau)\pi_\theta(\bar{\tau})d\tau d\bar{\tau}\\
&=\int_{\tau\sim\pi_{U(\theta)}}\int_{\bar{\tau}\sim\pi_\theta} R(\tau)\left[\pi_\theta(\bar{\tau})\frac{\partial}{\partial\theta}\pi_{U(\theta)}(\tau)+\pi_{U(\theta)}(\tau)\frac{\partial}{\partial\theta}\pi_\theta(\bar{\tau})\right]d\tau d\bar{\tau}\\
&\approx\frac{1}{T}\sum_{i=1}^{T}R(\tau^i)\frac{\partial}{\partial\theta}\log\pi_{U(\theta)}(\tau^i)+\frac{1}{T}\sum_{i=1}^{T}R(\tau^i)\frac{\partial}{\partial\theta}\log\pi_\theta(\bar{\tau}^i)\quad \tau^i\sim\pi_{U(\theta)}\quad \bar{\tau}^i\sim\pi_\theta
\end{aligned}$$

Note that, if we only consider the term on the left, we arrive at the original MAML algorithm Finn et al. (2017). This term encourages the agent to take update steps U that achieve good final rewards. It is exploitative. The second term encourages the agent to take actions such that the eventual meta-update yields good rewards (crucially, it does not try and exploit the reward signal under its own trajectory $\bar{\tau}$). This term allows the policy to be more exploratory, as it will attempt to deliver the maximal amount of information useful for the future rewards $R(\tau)$ without worrying about its own rewards $R(\bar{\tau})$. Since this algorithm augments MAML by adding in an exploratory term, we call it E-MAML.

4.3 E-RL^2

RL^2 optimizes (1) by feeding multiple rollouts from multiple different MDPs into an RNN. The hope is that the RNN hidden state update $h_t = C(x_t, h_{t-1})$, will learn to act as the update function U. Then, performing a policy gradient update on the RNN will correspond to optimizing the meta objective (1).

We can write the RL^2 update rule more explicitly in the following way. Suppose L represents an RNN. Let Env$_k(a)$ be a function that takes an action, uses it to interact with the MDP representing task k, and returns the next observation o, reward r, and a termination flag d. Then we have

$$\begin{aligned}
x_t &= [o_{t-1}, a_{t-1}, r_{t-1}, d_{t-1}]\\
L(h_t, x_t) &= [a_t, h_{t+1}]\\
\text{Env}_k(a_t) &= [o_t, r_t, d_t]
\end{aligned}$$

To train this RNN, we sample N MDPs from $\mathcal{M}$ and obtain k rollouts for each MDP by running the MDP through the RNN as above. We then compute a policy gradient update to move the RNN parameters in a direction which maximizes the returns over the k trials performed for each MDP.

To make this algorithm more exploratory, we can take inspiration from the insights shone by E-MAML. For each MDP M_i, we will sample p exploratory rollouts and $k - p$ non-exploratory rollouts. During an exploratory rollout, the forward pass through the RNN will receive all information. However, during the backwards pass, the future discounted returns for the policy gradient computation will zero out the contributions from exploratory episodes. The thinking is that this will encourage the RNN to treat its initial rollouts as exploratory, taking actions which may not lead to immediate rewards but rather to the RNN hidden weights performing better system identification, which will lead to higher rewards in later episodes.

5 Experiments

5.1 Krazy World Environment

To test the ability of meta RL algorithms to explore, we introduce a new environment known as Krazy World. This environment has the following features:

1. **8 different tile types**: Goal squares provide +1 reward when retrieved. The agent reaching the goal does not cause the episode to terminate, and there can be multiple goals. Ice squares will be skipped over in the direction the agent is transversing. Death squares will kill the agent and end the episode. Wall squares act as a wall, impeding the agent's movement. Lock squares can only be passed once the agent has collected the corresponding key

from a key square. Teleporter squares transport the agent to a different teleporter square on the map. Energy squares provide the agent with additional energy. If the agent runs out of energy, it can no longer move. The agent proceeds normally across normal squares.

2. **Ability to randomly swap color palette**: The color palette for the grid can be randomly permuted, changing the color that corresponds to each of the different tile types. The agent will thus have to identify the new system to achieve a high score. Note that in representations of the gird wherein basis vectors are used rather than images to describe the state space, each basis vector will correspond to a tile type, and permuting the colors will correspond to permuting the types of tiles these basis vectors represent. We prefer to use the basis vector representation in our experiments, as it is more sample efficient.
3. **Ability to randomly swap dynamics**: The game's dynamics can be altered. The most naive alteration simply permutes the player's inputs and corresponding actions (issuing the command for down moves the player up etc). More complex dynamics alterations allow the agent to move multiple steps at a time in arbitrary directions, making the movement more akin to that of chess pieces.
4. **Local or Global Observations**: The agent's observation space can be set to some fixed number of squares around the agent, the squares in front of the agent, or the entire grid. Observations can be given as images or as a grid of basis vectors. For the case of basis vectors, each element of the grid is embedded as a basis vector that corresponds to that tile type. These embeddings are then concatenated together to form the observation proper. We will use local observations.

A successful meta learning agent will first need to explore the environment, identifying the different tile types, color palette, and dynamics. A meta learning algorithm that cannot learn to incorporate exploration into its quick updating strategy will ultimately have insufficient data to navigate the game. We see this empirically in our experiments below.

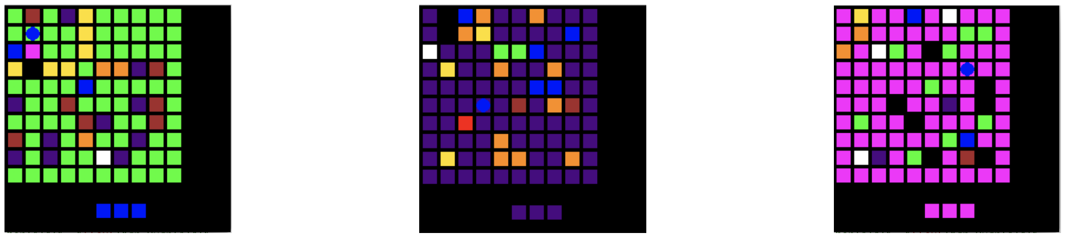


Figure 1: Three example worlds drawn from the task distribution. The agent must first perform an exploratory stage of system identification before exploiting. For example, in the leftmost grid the agent must first identify that the orange squares give +1 reward and the blue squares replenish its limited supply of energy. Further, it will need to identify that the gold squares block progress and the black square can only be passed by picking up the pink key. The agent may also want to identify that the brown squares will kill it and that it will slide over the purple squares. The other center and right worlds show these assignments will change and need to be re-identified every time a new task is drawn.

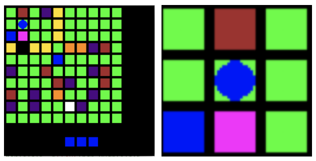


Figure 2: A comparison of local and global observations for the Krazy World environment. In the local mode, the agent only views a 3×3 grid centered about itself. In global mode, the agent views the entire environment.

5.2 Mazes

A collection of maze environments. The agent is placed at a random square within the maze and must learn to navigate the twists and turns to reach the goal square. A good exploratory agent will spend some time learning the maze's layout in a way that minimizes repetition of future mistakes. The mazes are not rendered, and consequently this task is done with state space only. The mazes are 20×20 squares large.

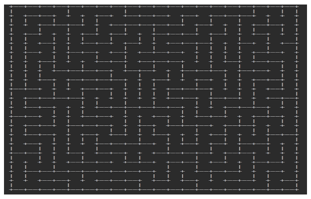


Figure 3: One example maze environment rendered in human readable format. The agent attempts to find a goal within the maze.

5.3 Results

In this section we present the following experimental results.

1. Learning curves on Krazy World and mazes.
2. The gap between the agent's initial performance on new environments and its performance after updating. A good meta learning agent will have a large gap after updating. A standard RL agent will have virtually no gap after one update.
3. Two heuristics which are designed to measure how well all of the algorithms are exploring.
4. Additionally, the following results can be found in the appendix:
   (a) Curves that show variance on returns of all four meta learning algorithms over 16 different experimental seeds.
   (b) More learning, variance, and first-update-performance-gap curves for Krazy World. These curves use different seeds to sample the testing environments. To convince ourselves that our results work without over-fitting to the test distribution, no hyper-parameter tuning was done on these seeds.
   (c) An analysis of the number of gradient steps taken vs. the final returns for MAML and E-MAML. Since we see that returns are not improved dramatically with additional gradient steps, we present results on one gradient step in this section.

Figure 4 presents learning curves on Krazy World for MAML, E-MAML, RL^2, and E-RL^2. Results are averaged over 16 randomly sampled hyper parameter choices for all algorithms (See the appendix for a detailed listing of hyper-parameters. As noted in this section, the algorithmic seed is considered a hyper-parameter in these experiments). The Y axis is the reward obtained after training at test time on a set of 64 held-out test environments. For MAML and E-MAML, this means performing one gradient step at test time. For RL^2 and E-RL^2, this means running three exploratory rollouts to allow the RNN memory weights time to update and then reporting the loss on the fourth and fifth rollouts. The X axis is the total number of environmental time-steps the algorithm has used for training. Every time an action is handed to an environment during training and that environment advances forward, this count increments by one.

Learning curves for mazes are presented in Figure 5. Here, the story is different than Krazy World. RL^2 and E-RL^2 both perform better than MAML and E-MAML. We suspect the reason for this is that RNNs are able to leverage memory, which is more important in mazes than in Krazy World. It is interesting to note that E-RL^2 was still more successful than RL^2, though both algorithms do converge to similar policies. It is interesting to note that this environment carries a penalty for hitting the wall, which MAML and E-MAML discover immediately. However, it takes E-RL^2 and RL^2 much longer to discover this penalty, resulting in worse performance at the beginning of training.

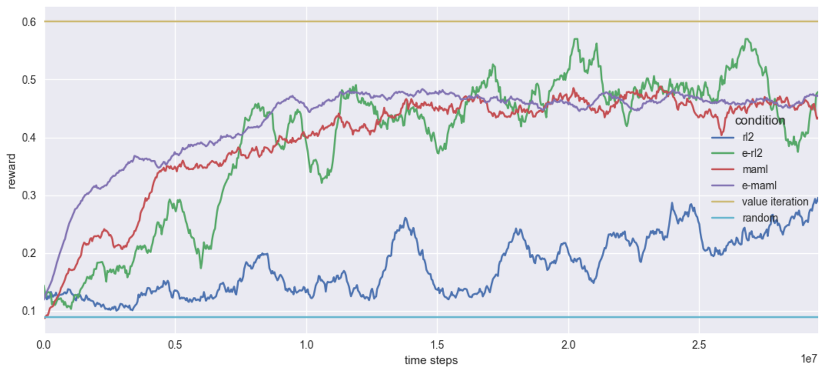


Figure 4: Meta learning curve on Krazy World. We see that E-RL^2 is at times closest to the optimal solution (obtained via value iteration), but has higher variance. E-MAML converges faster than MAML, although both algorithms do manage to converge. Meanwhile, RL^2 cannot converge in the given time. However, it is possible this curve would converge if given more time.

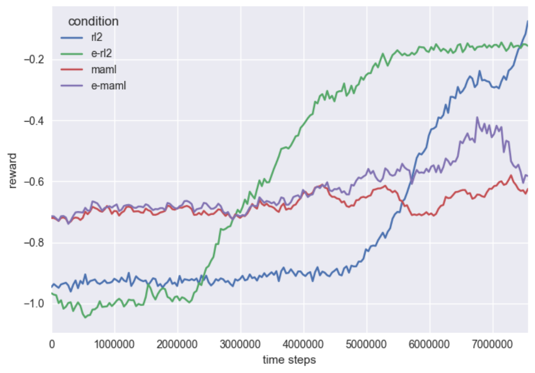


Figure 5: Meta learning curve on mazes. The environment gives the agent a penalty for hitting the wall. Interestingly, E-RL^2 and RL^2 take much longer to learn how to avoid the wall. However, they also deliver better final performance. E-RL^2 learns faster than RL^2, but offers comperable final performance. Since MAML and E-MAML utilize MLPs and have no memory, we do not expect good performance on this task.

When examining meta learning algorithms, one important metric is the update size after one learning episode. A good meta learning algorithm should have a large gap between its initial policy, which is largely exploratory, and its updated policy, which should often solve the problem entirely. For MAML, we look at the gap between the initial policy and the policy after one policy gradient step (see the appendix for information on further gradient steps). For RL^2, we look at the results after three exploratory rollouts, which give the RNN hidden state h sufficient time to update. Note that this number of exploratory rollouts was used during training as well. As baselines, we also plot the average gap in performance for two rollouts of a random policy, and the average gap in a PPO policy that is trained on only the test-time rollouts.

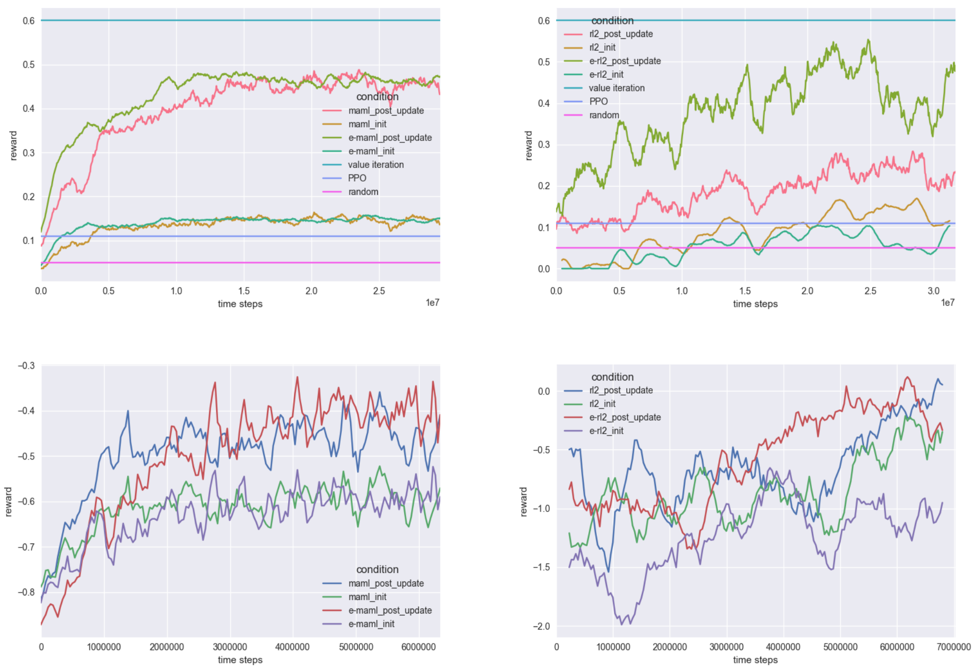
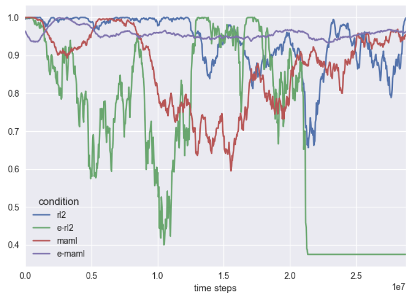


Figure 6: Gap in initial policy and post update policy for MAML/EMAML on Krazy World (top left), RL^2/E-RL^2 on Krazy World(top right), MAML/EMAML on mazes (bottom left) and RL^2/E-RL^2 on mazes (bottom right). We see that generally the MAML family is more stable than the RL^2 family for this task. For Krazy World, the gap between the initial policy and the updated policy is large, which indicates that at the very least the meta learning is able to do system identification correctly. For mazes, the gap is often less pronounced.

Finally, in Figure 7 we consider two heuristics for measuring the exploratory ability of the meta-learners. First, we consider the fraction of tile types visited by the agent at test time. The intuition is that good exploratory agents will visit and identify all different tile types so it can optimally solve the game. Second, we consider the number of times an agent visits a death tile at test time. Agents that are efficient at exploring should visit this tile type exactly once and then avoid it. We often find that more naive agents will run into these tiles repeatedly, causing their death many times over and a sense of pity in any onlookers. Overall, we see that the exploratory algorithms achieve better performance on these heuristics. However, we hope that in the future with further refinement to exploration in meta-learning, the results can be made more clear and convincing.

6 Conclusion

We considered the problem of exploration in meta reinforcement learning. Two new algorithms were derived and their properties were analyzed. In particular, we showed that these algorithms tend to learn more quickly and explore more efficiently than existing algorithms. It is likely that future work in this area will focus on meta-learning a curiosity signal which is robust and transfers across tasks. Perhaps this will enable meta agents which learn to explore rather than being forced to explore by mathematical trickery in their objectives.



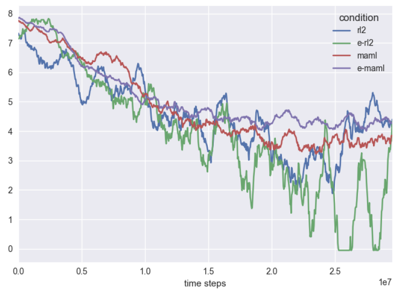


Figure 7: Two heuristics to measure exploration: Fraction of tile types visited during test time (left) and number of times killed at a death square during test time (right). We see that E-MAML is consistently the most diligent algorithm at checking every tile type during test time. Beyond that, things are fuzzy with RL^2 and MAML both checking most tile types at test time and E-RL^2 being exceptionally sporadic in this regard. Note that for this experiment, agents were given 8 rollouts at test time. Hence, the maximal number of expected deaths is 8. We see that most agents start around this number, and decrease to around 4 by the time they have converged.

REFERENCES

P. Bacon and D. Precup. The option-critic architecture. *In NIPS Deep RL Workshop*, 2015.

M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count based exploration and intrinsic motivation. *NIPS*, 2016.

R. I. Brafman and M. Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 2002.

D. Carmel and S. Markovitch. Exploration strategies for model-based learning in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 1999.

Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. Rl2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic metalearning for fast adaptation of deep networks. *ICML*, 2017.

Vincent Graziano, Tobias Glasmachers, Tom Schaul, Leo Pape, Giuseppe Cuccu, Jurgen Leitner, and Jürgen Schmidhuber. Artificial Curiosity for Autonomous Space Exploration. *Acta Futura*, 1(1): 41–51, 2011. doi: 10.2420/AF04.2011.41. URL `http://dx.doi.org/10.2420/AF04.2011.41`.

R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. Vime: Variational information maximizing exploration. *NIPS*, 2016.

M. J. Kearns and S. P. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 2002.

J. Z. Kolter and A. Y. Ng. Near-bayesian exploration in polynomial time. *ICML*, 2009.

T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv:1602.01783*, 2016.

Hung Ngo, Matthew Luciw, Alexander Forster, and Juergen Schmidhuber. Learning skills from play: Artificial curiosity on a Katana robot arm. *Proceedings of the International Joint Conference on Neural Networks*, pp. 10–15, 2012. ISSN 2161-4393. doi: 10.1109/IJCNN.2012.6252824.

I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. *NIPS*, 2016.

G. Ostrovski, M. G. Bellemare, A. v. d. Oord, and R. Munos. Count-based exploration with neural density models. *arXiv:1703.01310*, 2017.

A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *CoRR, vol. abs/1606.04671*, 2016.

Juergen Schmidhuber. On Learning to Think: Algorithmic Information Theory for Novel Combinations of Reinforcement Learning Controllers and Recurrent Neural World Models. pp. 1–36, 2015. ISSN 10450823. doi: 1511.09249v1. URL `http://arxiv.org/abs/1511.09249`.

Jürgen Schmidhuber. A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers. *Meyer, J.A. and Wilson, S.W. (eds) : From Animals to animats*, pp. 222–227, 1991.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *Arxiv preprint 1502.05477*, 2015.

Bradly C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv:1507.00814*, 2015.

Jan Storck, Sepp Hochreiter, and Jürgen Schmidhuber. Reinforcement driven information acquisition in non-deterministic environments. *Proceedings of the International ...*, 2:159–164, 1995.

Yi Sun, Faustino Gomez, and Jürgen Schmidhuber. Planning to be surprised: Optimal Bayesian exploration in dynamic environments. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6830 LNAI:41–51, 2011. ISSN 03029743. doi: 10.1007/978-3-642-22887-2_5.

H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. exploration: A study of count-based exploration for deep reinforcement learning. *arXiv:1611.04717*, 2016.

S. Tessler, C. Givony, T. Zahavy, D.J. Mankowitz, and S. Mannor. A deep hierarchical approach to lifelong learning in minecraft. *arXiv:1604.07255*, 2016.

A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. *arXiv:1611.05763*, 2016.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning 8, 3-4 (1992), 229256*, 1992.

7 Appendix A: Additional curves

7.1 Variance curves for MAML, E-MAML, RL2, E-RL2

Figure 8 and Figure 9 show the variance of learning curves over the 16 sampled hyper-parameters on the 64 selected test environments. These graphs are designed to show both the sensitivity of the algorithms to choices of hyper-parameters, as well as the variance of the algorithms themselves.

We see that E-RL^2 and RL^2 start with the lowest variance. Unfortunately, their variance never decreases throughout training, suggesting the algorithms have not yet fully converged in this time-step regime. When we manually inspected the behavior of instances in E-RL^2 that yielded particularly high variance, we found that most of the bad behaviors devolved to taking seemingly random actions. To us, this suggests that the RL^2 family of algorithms may be over-fitting to the training set.

For MAML and E-MAML, the variance for E-MAML starts higher initially but then decreases and eventually converges much faster than MAML. Since E-MAML provides an unbiased estimate of the gradient and MAML does not, this behavior is in line with expectations.

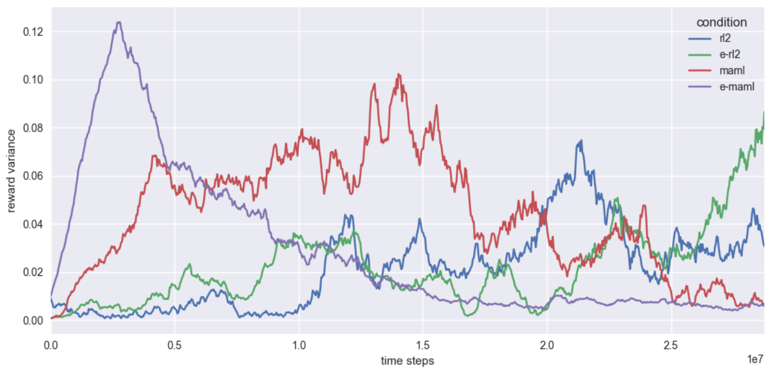


Figure 8: Variance of meta learning curves on Krazy World. E-MAML offers the best performance here, converging in around half the time it takes MAML. Meanwhile, the variance of E-RL^2 is the highest towards the end, which suggests over-fitting.

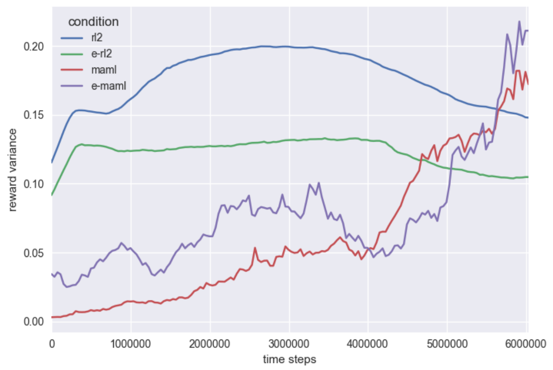


Figure 9: Variance of meta learning curves on mazes.

7.2 Learning, Variance, and First-update-performance-gap Curves for Three additional Test Environment Seeds

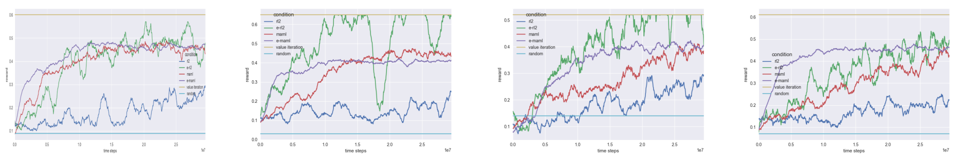


Figure 10: Learning curves for 4 different seeds of test environments. To convince ourselves that our results work without over-fitting to the test distribution, no hyper-parameter tuning was done on these seeds.

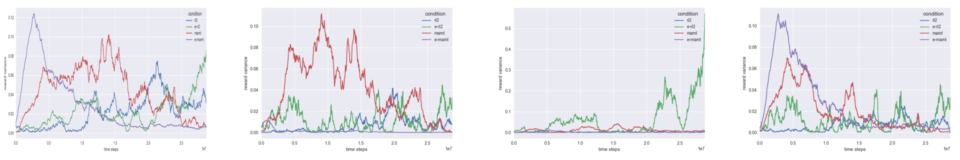


Figure 11: Variance curves for 4 different seeds of test environments.

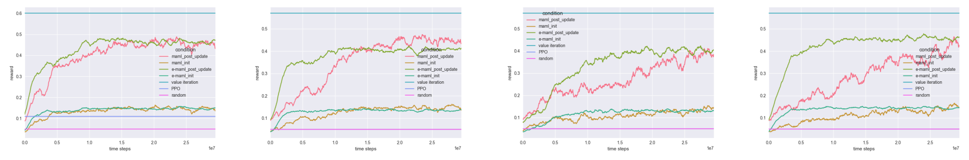


Figure 12: First-update-performance-gap curves for 4 different seeds of test environments. Results for MAML and E-MAML

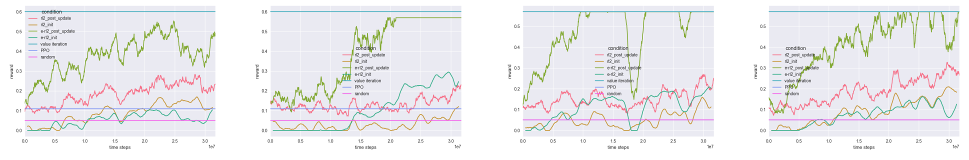


Figure 13: First-update-performance-gap curves for 4 different seeds of test environments. Results for E-RL^2 and RL^2.

7.3 MAML Rewards vs number of gradient steps

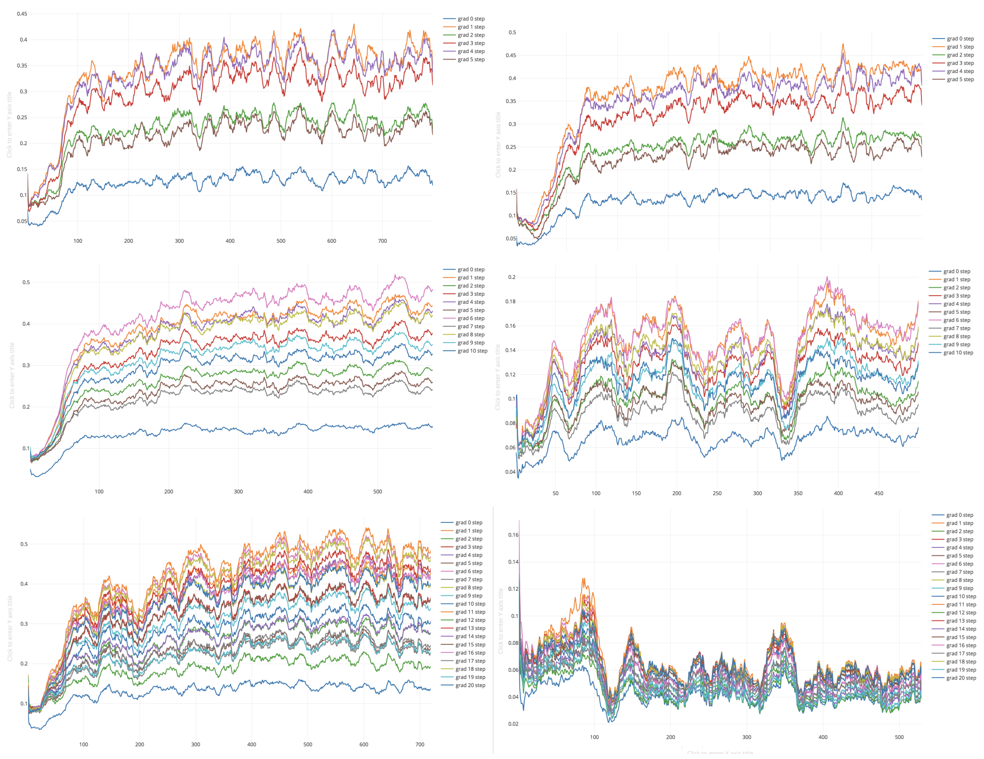


Figure 14: MAML on the right and E-MAML on the left. A look at the number of gradient steps at test time vs reward on the Krazy World environment. We see that beyond one gradient step, the returns are usually non-existent for both MAML and E-MAML. Hence, we only show results for one gradient step in the experimental results section.