META-LEARNING Q VALUES IN MOUNTAINCAR

Fengjun Yang
Department of Aeronautics and Astronautics
Stanford University
Stanford, CA, 94305, USA
fyang3@stanford.edu

ABSTRACT

In this work, we examine the possibility of producing an agent that can quickly adaptive to a new task by learning a prior distribution of action value functions. This work builds on the meta-learning framework ALPaCA and investigates the use of Thompson sampling to encourage more efficient exploration in the online setting. We also frame the popular MountainCar environment into a meta-learning environment by randomizing the gravity of the environment and thrust of the car. We show that both the mean and the variance prediction given by the meta-trained ALPaCA model capture useful information. However, empirically, exploring using Thompson sampling in this case does not outperform simple ε-greedyexploration strategy. We acknowledge that the reason for this is still unclear, but offer some hypotheses.

1 INTRODUCTION

In recent years, the advance of machine learning algorithms have been conspicuous. Just a few years ago, people were still marvelled at the fact that algorithmic agents can play some Atari games on a similar level as humans do (Mnih et al. (2015)). Less than one year later, reinforcement learning agents already mastered the art of Go and defeated the best human player (Silver et al. (2016)), only to be defeated by another agent that learned Go from scratch by just self-play (Silver et al. (2017)). These advancements demonstrated great potential for building algorithmic agents that can specialize in a specific task. However, despite their super-human level expertise in their specialty areas, these agents lack the ability to transfer their learned skills to new unseen tasks.

This has prompted research in building agents that can learn skills for more general tasks, or adapt quickly to new tasks. One framework for this problem is learning to learn, or meta-learning. In the meta-learning setting, there exist a family of tasks, which follow a distribution unknown to the agent. The algorithm takes as input several tasks that are sampled from the distribution, and outputs a learner that can quickly adapt to new, unseen tasks under the same distribution. One of the more popular algorithm, MAML (Finn et al. (2017)) learns a representation of the general family of tasks during meta-training time, and frames the new task adaptation as the fine tuning of the learned neural network. The authors demonstrated great performance empirically both for supervised and reinforcement learning tasks.

However, meta-reinforcement-learning (meta-RL) poses a unique challenge that deserves to be specifically addressed. In meta-RL, the output of the meta-learning algorithm is an agent that can be deployed in a new environment and quickly find the optimal policy in the new environment. The agent not only has to do quickly learn a new policy, but also has to gather the data necessary to perform the learning by autonomous interacting with the environment. This suggests that we need to meta-train the agent to not only be good at learning, but also at exploring. The problem of efficient exploration in meta-RL has been studied in several works that build upon the MAML framework(Stadie et al. (2018); Gupta et al. (2018); Rothfuss et al. (2018)). However, using meta-learned action values to guide exploration has been less explored.

We propose a method that uses the meta-training phase to approximate a distribution of q functions that can be used as the prior for the output agent during online reinforcement learning. We assume that we have a number of Q functions that are known a priori to us and can be used for meta-training.

We build our method heavily on the ALPaCA (Harrison et al. (2018)) framework. We represent the distribution of Q functions with a neural network with a Bayesian last layer, which is equivalent to performing Bayesian linear model in a feature space. By maintaining this distribution over the last layer weights, we gain uncertainty information in our estimates. During online training, we leverage the learned uncertainty information to guide exploration, using the technique of Thompson sampling. However, instead of the idea of sampling MDP's as proposed in (Strens (2000); Osband et al. (2013)), we only sample the weight of the last layer. In this sense, our method for online exploration is very similar to that of (Osband et al. (2016); Azizzadenesheli et al. (2018)). We then tested our method in a modified version of a classical control problem called mountain car. However, in our specific experiment setup, the result suggests that uncertainty-guided exploration is not as helpful as anticipated and is outperformed by simple ε-greedy exploration.

In the rest of this report, we first present a survey of related works in section 2, with the exception of the ALPaCA framework, which will be introduced in detail in section three. In section four, we specify our method and sketch out our algorithm. We then present and discuss the experiment setup and results in section five. Finally, in section six, we report on current progress, and suggest potential extensions to this method.

2 RELATED WORKS

2.1 META-LEARNING

Generally speaking, in a meta-learning task, the meta-learner tries to learn a general skill for a distribution of tasks by solving a number of tasks sampled from the distribution of interest. This work is heavily built on the ALPaCA (Harrison et al. (2018)) algorithm, which uses a neural network with Bayesian last layer. During the meta-training (offline) phase, ALPaCA learns the weights of the neural network, as well as a prior distribution over the weights of the last layer. In test-time (online), the weights of the neural network is fixed and the prediction is performed as a Bayesian linear regression in the feature space represented by the neural network. This method allows us to obtain information about how certain we are about a prediction, which can be desirable in many cases. Recently, the MAML (Finn et al. (2017)) algorithm has achieved impressive results by framing the test-time training as parameter fine tuning and explicitly model fast adaptation in the meta-training loss function. Although MAML only gives point estimates for a given test point, there have been methods that recasts MAML as a hierarchical Bayes approach to get Bayesian prediction (Grant et al. (2018); Finn et al. (2018)). We note, however, that the ALPaCA approach is different in that the weights of the neural network is fixed online, and we only maintain the posterior over the last layer weights. This can be seen as assuming a delta-prior over the neural network weights, as opposed to the other methods that maintain a posterior over all the weights and thereby adopt a fully Bayesian approach. This decision about ALPaCA is made in the interest of computation complexity (Harrison et al. (2018)), which can be important in certain scenarios in reinforcement learning where the agent has to quickly react to the environment.

2.2 EFFICIENT EXPLORATION IN META-RL

Although efficient exploration in meta-RL is a relatively new field, several results have been public, especially for those that extend the MAML framework. (Stadie et al. (2018)) seeks explicitly model the update step and its effect on sampling efficiency into the MAML loss function and proposes an algorithm names E-MAML. (Gupta et al. (2018)) tries encourage exploration by the injection of noise. The level of the noise to inject is controlled by a latent variable that is learned during meta-training time. (Rothfuss et al. (2018)) argues that previous works in meta-RL suffer from poor credit assignments. It gives a theoretical analysis of the credit assignment problem in the meta-RL scenario and also proposes an algorithm that applies proximal policy optimization(PPO)(Schulman et al. (2017)) to the search of the meta-policy to improve the efficiency of both meta-training and test-time task identification. These works based on the MAML framework fall under policy gradient methods, and are different from the q-learning approaches that we take in this work. While these work demonstrate impressive empirical results, we hope to improve the sample efficiency by meta-learning the q-functions instead of the policy directly, and use the learned uncertainty information to guide better exploration behaviors during test-time.

2.3 THOMPSON SAMPLING AND BAYESIAN DEEP Q NETWORK

Thompson sampling(Thompson (1933)), or posterior sampling, was studied extensively as a heuristics in the multi-arm bandit problem, before it was extended to MDP scenarios by (Strens (2000)). In an episodic MDP setting, posterior sampling reinforcement learning (PSRL) requires one to maintain a belief over the MDP of interest. At the start of each episode, one samples an MDP from one's belief, and then roll out the optimal policy for the sampled MDP. The experience can then be used to update one's belief of the MDP. Theoretical analysis of this approach (Osband et al. (2013)) have showed that they have strong regret bounds. Empirical results also suggest that they can perform better than both naive exploration strategies (Azizzadenesheli et al. (2018)) or optimism-based methods (Osband et al. (2013)). Intuition and discussion on why posterior sampling leads to more efficient exploration than ε-greedy method can also be found in (Azizzadenesheli et al. (2018); Osband et al. (2016)). However, sampling an MDP and planning for the best policy can be intractable when the size of the MDP grows larger. To address this problem, the sampling of MDP can be replaced with sampling the value function (Osband et al. (2016); Azizzadenesheli et al. (2018)). Specifically, our method is similar to that of (Azizzadenesheli et al. (2018)), where we use the neural network with Bayesian last layer to represent the distribution of value functions and sample the last layer weights during each sampling.

3 THE ALPACA ALGORITHM

On a high level, ALPaCA (Harrison et al. (2018)) learns an efficient online regressor through meta-training on a batch of similar tasks. The meta-training occurs offline where the algorithm samples tasks and learn a general representation of the potential function of interest. The meta-training outputs an efficient regressor that leverages the learned information to quickly adapt to new functions it wants to approximate.

Borrowing the notation from ALPaCA (Harrison et al. (2018)), we denote the function of interest as $f : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$, parametrized with a latent variable θ. We observe noisy samples (x_t, y_t) that arrive in sequence, where $p(y_t|x_t,\theta) \sim \mathcal{N}(f(x;\theta),\Sigma_\varepsilon)$. We use $D_\tau(\theta_j) := \{(x_t,y_t\}_{t=1}^{\tau}$ to denote a dataset of τ samples for one specific task parametrized by θ_j, where $y_t \sim p(y_t|x_t,\theta)$. Let $\mathscr{D} := \{D_\tau(\theta_j)\}_{j=1}^{J}$ represent the meta-training dataset that contain J tasks. A general assumption we have for this dataset is that each $\theta_j \sim p(\theta)$, where $p(\theta)$ is the distribution of θ during online phase. This ensures that the task distribution during offline phase matches the task distribution during online phase. The goal of the offline training is to learn a regressor, such that after observing online data $\tilde{D}_\tau$, can produce a density $q(y|x,\tilde{D}_\tau$ that closly approximates the actual density of $p(y|x,\theta_j)$, where θ_j is the true latent parameter. Formally, we write the objective as minimizing the KL divergence between the learned density and the actual density.

$$\mathbb{E}_{\tilde{\theta}\sim p(\theta)}[D_{KL}(p(y|x,\tilde{\theta}||q(y|x,\tilde{D}_\tau], \tag{1}$$

In ALPaCA, the learned information is modeled as a neural network with Bayesian last layer. The prediction step can be seen as: first, use the neural network $\phi(\cdot;w)$ to map the input x_t into a feature space; then, use Bayesian linear regression (BLR) with weights K to compute the density prediction q. The basis function (represented by the neural network) $\phi : \mathbb{R}^{n_x} \to \mathbb{R}^{n_\phi}$ is parametrized by weight w. The BLR weights K are assumed to follow a matrix normal distribution $K \sim \mathscr{MN}(\bar{K}_0,\Lambda^{-1},\Sigma_\varepsilon)$. Thus the model can be written as

$$y_t^T = \phi^T(x_t;w)K + \varepsilon,$$

where $\varepsilon \sim \mathcal{N}(0,\Sigma_\varepsilon)$ is a noise term and Σ_ε is assumed to be known. After observing a data point, we can update our belief over the distribution of our BLR weights K by computing a new posterior distribution. Writing our observations as $Y^T = [y_1...y_\tau]$, $\Phi^T = [\phi(x_1)...\phi(x_\tau)]$, the posterior of K is given by a closed-form expression

$$\begin{aligned} \Lambda_\tau &= \Phi^T\Phi + \Lambda_0 \\ K_\tau &= \Lambda_\tau^{-1}(\Phi^T Y + \Lambda_0 K_0) \end{aligned} \tag{2}$$

Using the updated posterior, we can predict the output $y_{\tau+1}$ of input $x_{\tau+1}$ as

$$\begin{aligned} p(y_{\tau+1}|\phi(x_{\tau+1}),\Phi,Y) &= \mathcal{N}(K_\tau^T\phi(x_{\tau+1}),\Sigma_{\tau+1}) \\ \Sigma_{\tau+1} &= 1 + \phi^T(x_{\tau+1})\Lambda_\tau^{-1}\phi(x_{\tau+1})\Sigma_\varepsilon \end{aligned} \tag{3}$$

With this closed-form expression for our density prediction, we can rewrite the objective 1 in terms of the variables of interest $\bar{K}_0, \Lambda_0$, and w. We state the final derived loss function and optimization problem formulation, but omit the technical details. Interested readers can find step-by-step derivation in (Harrison et al. (2018)) for reference.

$$\hat{l}(\cdot) := \frac{1}{J}\sum_{j=1}^{J}(n_\phi log(1+\phi_{t_j}^T \Lambda_{t_j-1}^{-1}\phi_{t_j}) + (y_{t_j} - K_{t_j-1}^T \phi_{t_j})^T \Sigma_{t_j}^{-1}(y_{t_j} - K_{t_j-1}^T \phi_{t_j}) \tag{4}$$

Meta-training can be written as the optimization problem

$$\begin{aligned} &\min_{\bar{K}_0,\Lambda_0,w} \hat{l}(\bar{K}_0,\Lambda_0,w) \\ &\quad s.t. \Sigma_{t_j} = (1+\phi^T(x_{t_j}\Lambda_{t_j-1}^{-1}\phi(x_{t_j}))\Sigma_\varepsilon, j=1,...,J \\ &\qquad \Lambda t_j - 1 = \Phi_j^T \Phi_j + \Lambda_0, j=1,...,J \\ &\qquad K_{t_j-1} = (\Lambda t_j - 1)^{-1}(\Phi_j^T Y_j + \Lambda_0 \bar{K}_0), j=1,...,J \\ &\qquad \Lambda_0 \succeq 0 \end{aligned} \tag{5}$$

We can enforce the semi-positive property of Λ_0 by expressing it as $\Lambda_0 = L_0 L_0^T$, and then solve this problem using gradient descent methods by taking advantage of the auto-differential tools like Tensorflow.

In the online part of the algorithm, the prediction follows from 2 and 3. However, we might want to avoid invert Λ_τ at each time step for better computation efficiency. To do this, we invoke the Sherman-Morrison formula, a special case of the Woodbury identity, to turn this into an incremental update. We first define $Q_0 = \Lambda_0 \bar{K}_0$. For each step, we incrementally update Λ_t, Q_t, and K_t using

$$\begin{aligned} \Lambda_t^{-1} = \Lambda_{t-1}^{-1} - \frac{1}{1+\phi_t^T \Lambda_{t-1}^{-1}\phi_t}(\Lambda_{t-1}^{-1}\phi_t)(\Lambda_{t-1}^{-1}\phi_t)^T \\ Q_t = \phi_t^T y_t + Q_{t-1} \\ K_t = \Lambda_t^{-1} Q_t \end{aligned} \tag{6}$$

By avoiding inverting Λ_τ at each time step, ALPaCA improves the computation complexity for each update-prediction from $O(n_\phi^3)$ to $O(n_\phi^2)$.

4 METHOD

Algorithm 1: ALPaCA-Q Offline

Require: training data $\mathscr{D}$, noise variance Σ_ε Randomly initialize $\bar{K}_0, L_0, w$;
while *not converged and have not reached step limit* **do**
  for *j = 1 to J* **do**
    Sample t_j from uniform distribution over 1, . . ., τ;
    Sample dataset D_{t_j} uniformly from $\mathscr{D}$;
    Compute $\bar{K}_{t_j-1}, \Lambda_{t_j-1}, \Sigma_{t_j-1}$ using dataset D_{t_j};
  end
  Update $\bar{K}_0, L_0, w$ based on 5
end
Return $\bar{K}_0, L_0, w$;

The method used in this work is heavily built on ALPaCA, and we call it the ALPaCA-Q algorithm. In the case of ALPaCA-Q, we try to learn an action-value function $f : (S \times A) \to \mathbb{R}$, also parametrized by latent variable θ. Let $x_t = (s_t, a_t) \in SxA$ denote a state-action pair, and $f(x_t)$ maps this pair of state and action to its action value. The dataset $\mathscr{D}$ thus consists $\mathscr{D} := \{\{(s_t, a_t), y_t\}_{t=1}^{\tau}\}_{j=1}^{J}$. This formulation of Q-value meta-learning is directly compatible with the ALPaCA architecture. Thus, we directly apply the ALPaCA algorithm for the meta-training phase. The algorithm is given in 1,

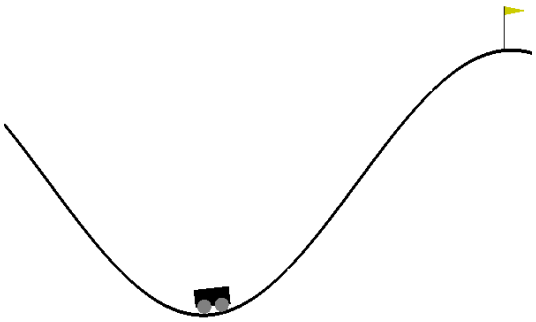

Figure 1: The goal of the task is to drive the car out of the valley to reach the goal position, labeled by the flag.

which is adapted from the ALPaCA offline algorithm given in (Harrison et al. (2018)) with minimal change.

In the online phase, we incorporate the idea from BDQN to perform Thompson sampling by sampling the last layer weights. For each step, the agent picks the action that maximizes the expected return predicted with the current sampled last layer. It then updates the model with observed reward. For every T_{sample} steps, it resamples its last layer weights. To make the learning algorithm more stable, we also have the agent keep a target last layer to generate the predictions used for updating the model. It updates the target weights to the current model every T_{target} steps. The algorithm is given in 2.

Algorithm 2: ALPaCA-Q Online

Require: $\bar{K}_0, \Lambda_0, \phi, \Sigma_\varepsilon$ counter = 0;
$K_\tau := \bar{K}_0, \Lambda_\tau := \Lambda_0$;
$K_{target} := \bar{K}_0$;
while *not converged and have not reached step limit* **do**
  Initialize s to state of new environment;
  for $t = 1$ *to end of episode* **do**
    if *count mod* $T_{sample} = 0$ **then**
      Sample $K \sim MN(\bar{K}_\tau, \Lambda_\tau^{-1}, \Sigma_\varepsilon)$;
    Select action $a_t = argmax_{a'}[K^T \phi(x_t)]_{a'}$;
    Execute action a_t and observe reward r_t and next state s_{t+1};
    if s_{t+1} *is terminal state* **then**
      $y_t = r_t$
    else
      $y_t = r_t + max_{a'}[K_{target} \phi(x_{t+1}]_{a'}$
    end
    $\Lambda_t^{-1} = \Lambda_{t-1}^{-1} - \frac{1}{1+\phi_t^T \Lambda_{t-1}^{-1} \phi_t} (\Lambda_{t-1}^{-1} \phi_t)(\Lambda_{t-1}^{-1} \phi_t)^T$;
    $Q_t = \phi_t^T y_t + Q_{t-1}$;
    $K_t = \Lambda_t^{-1} Q_t$;
    if *count mod* $T_{target} = 0$ **then**
      $K_{target} = K_t$
  end
end

5 EXPERIMENT

5.1 ENVIRONMENT SETUP

To test our method, we use a modified version of the MountainCar environment, a classical control problem widely-used as a toy example in the evaluation of reinforcement learning algorithms. In this problem, the agent's goal is to navigate a car stuck in a valley between two peaks to reach the goal position on the right peak. A visual illustration from a rendered step is shown in 1. At any given point, the agent can observe the position and velocity of the car as a 2-tuple of real numbers. The agent can affect the state of the car by choosing from three actions: accelerate in the positive direction, accelerate in the negative direction, or not applying any changes. The agent receives a constant penalty of -1 until it reaches the goal state, where it receives a reward of 0 and the episode ends. What makes this problem difficult is that, to reach the goal position, the car has to first build up enough speed by engaging in a pendulum-like motion at the bottom of the valley. At the same time, the agent cannot tell where the goal state is from its reward function. It can only learn about the objective when it stumbles upon the goal for the first time. This makes the role of efficient exploration more important than perhaps in some other environments.

To adapt the MountainCar environment to the meta-learning setting, we vary the thrust of the car and the gravity of the environment. More specifically, the state of the car is update with the formula

$$\begin{aligned}\text{velocity} &= \text{velocity} + (\text{action} - 1) * \text{thrust} + cos(3 * \text{position}) * (-\text{gravity}) \\ \text{position} &= \text{position} + \text{velocity}\end{aligned}$$

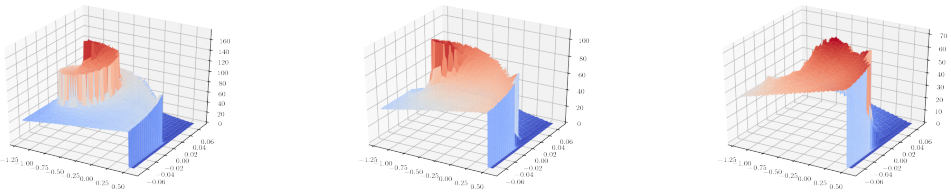

Figure 2: Different cost-to-go functions for three different environment configurations. **Left**: High gravity setting, **Middle**: Normal MountainCar, **Right**: High thrust setting

In our experiment setup, thrust is sampled uniformly from the range $[0.006, 0.0015]$, and gravity uniformly from $[0.002, 0.003]$. Thrust and gravity are sampled independently. We limit the parameters to their respective range to ensure that the optimal policies under different configurations have some variability but still share some common characteristics. One problem that we encountered from a more relaxed range of thrust and gravity was that the optimal policy in some of the configurations involve only straight-up accelerating, as the acceleration of thrust overpowers any deceleration from gravity. This results in a value function that is drastically different from other configurations both in terms of shape and magnitude. In 2, we plot the cost-to-go function of three different configurations: the high-gravity setting, the original MountainCar setting, and the high-thrust setting. An implementation of the environment, as well as the rest of the experiment, can be found on Github at https://github.com/jackyang96/ALPaCA-Q-learning.

5.2 META-LEARNING THE Q-FUNCTION

To gather the action value functions for meta-training, we sample 150 different configurations of the meta-learning MountainCar environment. We then use value iteration algorithm to find the value functions by discretizing the state space. The data is then injected with noise and normalized to have zero mean and unit variance before meta-training. The neural network used in this experiment is an MLP that consists of three hidden layers of size 128, 128, 16, respectively. Note that the basis function maps the state representation from $\mathbb{R}^2 \rightarrow \mathbb{R}^16$. The reason behind this is that learning a high-dimension representation of the states have been shown empirically in (Harrison et al. (2018)) to have good performance for ALPaCA. The learned cost-to-go function is plotted in 3.

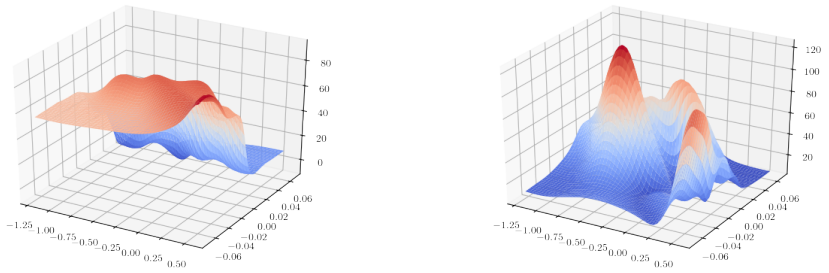

Figure 3: Cost-to-go function learned by ALPaCA. **Left**: Prediction of Mean. **Right**: Prediction of variance.

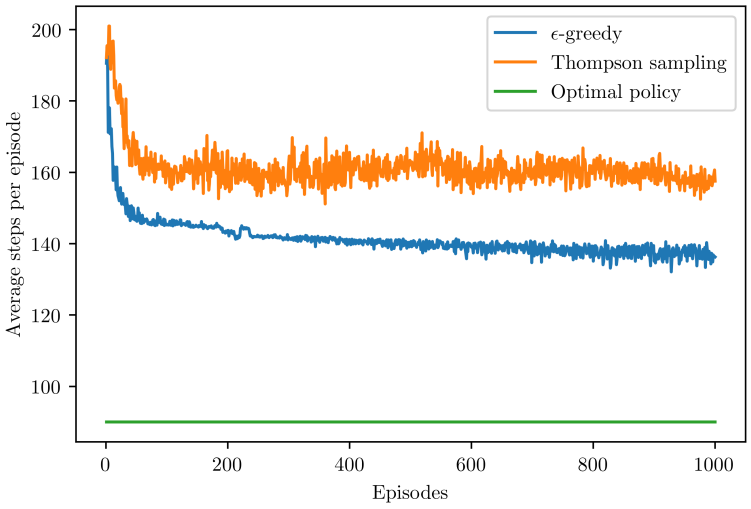

Figure 4: Number of episodes versus average steps taken to complete an episode

This result shows that the ALPaCA algorithm succeeds in capturing certain characteristics of the Q functions. First, the mean prediction captures some commonalities between different configurations. For example, it learns that having a high velocity is usually associated with a lower cost-to-go, while have a low velocity is usually bad and is associated with a higher cost-to-go. It also learns that being in the valley (position ~ -0.75) is less desirable than other locations (associated with higher cost-to-go). Second, the variance information captures certain differences between different environments. For example, the peak in the variance prediction in the center of the state-space correspond to the big difference between the magnitude of peaks in different environment configurations. Interestingly, we also notice that the sharp cliff at around goal position and low velocity in the true value functions is captured this by a smooth prediction surface coupled with high variance at this cliff. This suggests that ALPaCA might be able to deal with strong non-linearities without overfitting globally by modeling the non-linearity as a part of the variance.

5.3 Reinforcement Learning

The reinforcement learning experiment is still largely ongoing, and this subsection reports only our current progress. In the reinforcement learning experiment, we use the best meta-learned representation of the environment (shown in 3) to initialize our online RL agent. We deploy two agents: one uses Thompson sampling (TS) exploration strategy, and the other uses ϵ-greedyexploration strategy.

The agents' learned value functions are evaluated at the end of each episode. To evaluate a learned Q function, we deploy an agent that behaves greedily according to the learned value. We then record the average number of steps that the greedy agent takes to finish an episode over several trials. (The agent will have a slightly different initial position every time around.) The result is shown in 4. Surprisingly, the TS agent is outperformed by the ε-greedyagent, which is unexpected considering that the TS agent is utilizing more information than the ε-greedyagent. We acknowledge that the reason for this is not yet clear.

One suspicion is that the TS agent is more susceptible to the inaccuracies in the meta-learned action value function. During online update, the variance in the prediction is reduced by the observation of new data points. However, the algorithm does not distinguish between real returns from the environment at the end of the episode and other inaccurate bootstrapped returns. This might cause the algorithm to pre-maturely believe in a prediction that is not correct. This does not affect ε-greedyexploration, however, as the ε-greedyagent's exploration behavior does not depend on the variance information. If the learned Q function is very inaccurate in a certain neighborhood, TS might be convinced about the wrong prediction and prematurely give up exploration. One counter-argument for this hypothesis is that if TS agent is convinced that an action has significantly higher mean and smaller variance, it will choose this action consistently, and thus get disillusioned faster. However, in the process of getting disillusioned, the agent accumulate a large of amount of different prediction of the same point, expanding the variance yet again, and this time around will take more exploration for the variance to go down.

Another suspicion is that this is a result of implementation issues. Specifically, the problem of numerical stability comes to mind. Since we use the Sherman-Morrison Formula for incremental update of the mean and covariance matrices, numerical errors can compound. In the case of this specific learned representation, the Λ_τ^{-1} matrix contain numbers that are usually on the magnitude of $10^{-3} to 10^{-5}$, which makes numerical instability a possibility. We came across numerical issues in the earlier phase of the project, which led to the agents giving highly unstable predictions. This problem was patched by switching the precision of all arrays to long double. However, whether this fully addressed the numerical issue is unclear.

6 DISCUSSION AND FUTURE WORK

To summarize, we propose an meta-RL algorithm that combines the ALPaCA algorithm with the idea of Thompson sampling that takes action value functions as input and outputs an agent that can leverage the learned uncertainty information to guide exploration. However, empirically, the agent does not achieve the expected level of performance. Currently, we are working on explaining the reason behind this suboptimal performance. We are currently implementing a Monte-carlo version of TS agent. If the result is caused by TS agent being prematurely convinced about inaccurate bootstrapped prediction, the unbiased Monte-carlo approach might be a fix.

Moving forward, there are several interesting directions. First, although the ALPaCA algorithm learns the meta-trained representation with the assumption that during the online phase the weights of the neural network will not change, the learned representation can also be a good starting point for BDQN. It would be interesting to see whether this learned prior can speed up BDQN training and compare the performance between fixing the neural network weights versus updating them online. Secondly, it would be interesting to see a comparison between a better version of ALPaCA-Q and some policy gradient-based meta-RL methods like MAML in terms of sample efficiency. Policy gradient methods are widely regarded as less sample efficient than other methods. Whether we can increase the sample efficiency by learning the value function instead is also worthwhile.

ACKNOWLEDGMENT

Fengjun thanks James Harrison and Apoorva Sharma for their patient explanation of the ALPaCA algorithm, Professor Emma Brunskill for her guidance and suggestions, Sam Sokota for helpful discussions. Fengjun is supported by the Stanford Graduate Engineering Fellowship.

REFERENCES

Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. *arXiv preprint arXiv:1802.04412*, 2018.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. *arXiv preprint arXiv:1806.02817*, 2018.

Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*, 2018.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies, 2018.

James Harrison, Apoorva Sharma, and Marco Pavone. Meta-learning priors for efficient online bayesian regression. *arXiv preprint arXiv:1807.08912*, 2018.

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, 2015.

Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*, pp. 3003–3011, 2013.

Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 2377–2386. JMLR.org, 2016. URL `http://dl.acm.org/citation.cfm?id=3045390.3045641`.

Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. Promp: Proximal meta-policy search, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Bradly C Stadie, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever. Some considerations on learning to explore via meta-reinforcement learning. *arXiv preprint arXiv:1803.01118*, 2018.

Malcolm Strens. A bayesian framework for reinforcement learning. In *ICML*, pp. 943–950, 2000.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.