VARIBAD: A VERY GOOD METHOD FOR BAYES-ADAPTIVE DEEP RL VIA META-LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Trading off exploration and exploitation in an unknown environment is key to maximising expected return during learning. A Bayes-optimal policy, which does so optimally, conditions its actions not only on the environment state but on the agent's uncertainty about the environment. Computing a Bayes-optimal policy is however intractable for all but the smallest tasks. In this paper, we introduce variational Bayes-Adaptive Deep RL (variBAD), a way to meta-learn to perform approximate inference in an unknown environment, and incorporate task uncertainty directly during action selection. In a grid-world domain, we illustrate how variBAD performs structured online exploration as a function of task uncertainty. We also evaluate variBAD on MuJoCo domains widely used in meta-RL and show that it achieves higher return during training than existing methods.

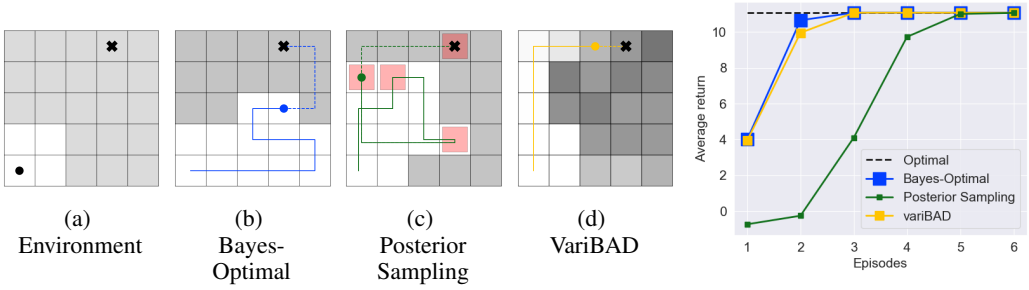
1 INTRODUCTION

Reinforcement learning (RL) is typically concerned with finding an *optimal policy* that maximises expected return for a given Markov decision process (MDP) with an unknown reward and transition function. If these were known, the optimal policy could in theory be computed without environment interactions. By contrast, learning in an *unknown* environment usually requires trading off exploration (learning about the environment) and exploitation (taking promising actions). Balancing this trade-off is key to maximising expected return *during learning*, which is desirable in many settings, particularly in high-stakes real-world applications like healthcare and education (Yauney & Shah, 2018; Liu et al., 2014). A *Bayes-optimal policy*, which does this trade-off optimally, conditions actions not only on the environment state but on the agent's own uncertainty about the current MDP.

In principle, a Bayes-optimal policy can be computed using the framework of Bayes-adaptive Markov decision processes (BAMDPs) (Martin, 1967; Duff & Barto, 2002), in which the agent maintains a belief distribution over possible environments. Augmenting the state space of the underlying MDP with this belief yields a BAMDP, a special case of a belief MDP (Kaelbling et al., 1998). A Bayes-optimal agent maximises expected return in the BAMDP by systematically seeking out the data needed to quickly reduce uncertainty, but only insofar as doing so helps maximise expected return. Its performance is bounded from above by the optimal policy for the given MDP, which does not need to take exploratory actions but requires prior knowledge about the MDP to compute.

Unfortunately, planning in a BAMDP, i.e., computing a Bayes-optimal policy that conditions on the augmented state, is intractable for all but the smallest tasks. A common shortcut is to rely instead on *posterior sampling* (Thompson, 1933; Strens, 2000; Osband et al., 2013). Here, the agent periodically samples a single hypothesis MDP (e.g., at the beginning of an episode) from its posterior, and the policy that is optimal *for the sampled MDP* is followed until the next sample is drawn. Planning is far more tractable since it is done on a regular MDP, not a BAMDP. However, posterior sampling's exploration can be highly inefficient and far from Bayes-optimal.

Consider the example of a gridworld in Figure 1, where the agent must navigate to an *unknown* goal located in the grey area (1a). To maintain a posterior, the agent can uniformly assign non-zero probability to cells where the goal could be, and zero to all other cells. A Bayes-optimal strategy strategically searches the set of goal positions that the posterior considers possible, until the goal is found (1b). Posterior sampling by contrast samples a possible goal position, takes the shortest route there, and then resamples a different goal position from the updated posterior (1c). Doing so is much less efficient since the agent's uncertainty is not reduced optimally (e.g., states are revisited).



(a) Environment (b) Bayes-Optimal (c) Posterior Sampling (d) VariBAD (e) Performance

Figure 1: Illustration of different exploration strategies. **(a)** Environment: The agent starts at the bottom left. There is a goal somewhere in the grey area, unknown to the agent. **(b)** A Bayes-optimal exploration strategy that systematically searches possible grid cells to find the goal, shown in solid (interactions so far) and dashed (future interactions) blue lines. A simplified posterior is shown in the background in grey ($p = 1/(\text{number of possible goal positions left})$ of containing the goal) and white ($p = 0$). **(c)** Posterior sampling, which repeatedly samples a possible goal position (red squares) and takes the shortest route there. Once the goal is found, every sample matches the true goal position and the agent acts optimally. **(d)** Exploration strategy learned by variBAD. The grey background represents the approximate posterior the agent has learned. **(e)** Average return over all possible environments, over six episodes with 15 steps each (after which the agent is reset to the starting position). The performance of any exploration strategy is bounded above by the optimal behaviour (of a policy with access to the true goal position). The Bayes-optimal agent matches this behaviour from the third episode, whereas posterior sampling needs six rollouts. VariBAD closely approximates Bayes-optimal behaviour in this environment.

As this example illustrates, Bayes-optimal policies can explore much more efficiently than posterior sampling. Hence, a key challenge is to find ways to learn approximately Bayes-optimal policies while retaining the tractability of posterior sampling. In addition, many complex tasks pose another key challenge: the inference involved in maintaining a posterior belief, needed even for posterior sampling, may itself be intractable.

In this paper, we combine ideas from Bayesian reinforcement learning, approximate variational inference, and meta-learning to tackle these challenges, and equip an agent with the ability to strategically explore unseen (but related) environments for a given distribution, in order to maximise its expected return.

More specifically, we propose *variational Bayes-Adaptive Deep RL* (variBAD), a way to meta-learn to perform approximate inference on a new task[1], and incorporate task uncertainty directly during action selection. We represent a single MDP using a learned, low-dimensional stochastic latent variable m. Given a set of tasks sampled from a distribution, we jointly meta-train: (1) a variational auto-encoder that can infer the posterior distribution over m in a new task while interacting with the environment, and (2) a policy that conditions on this posterior *distribution* over the MDP embeddings, and thus learns how to trade off exploration and exploitation when selecting actions *under task uncertainty*. Figure 1e shows the performance of our method versus the hard-coded optimal (i.e., given privileged goal information), Bayes-optimal, and posterior sampling exploration strategies. VariBAD's performance closely matches that of Bayes-optimal action selection, matching optimal performance from the third rollout.

Previous approaches to BAMDPs are only tractable in environments with small action and state spaces. VariBAD offers a tractable and dramatically more flexible approach for learning Bayes-adaptive policies tailored to the task distribution seen during training, with the only assumption that such a task distribution is available for meta-training. We evaluate our approach on the grid-world shown above and on MuJoCo domains that are widely used in meta-RL, and show that variBAD exhibits superior exploratory behaviour at test time compared to existing meta-learning methods, achieving higher returns during learning. As such, variBAD opens a path to tractable approximate Bayes-optimal exploration for deep reinforcement learning.

[1] We use the terms environment, task, and MDP, interchangeably.

2 BACKGROUND

We define a Markov decision process (MDP) as a tuple $M = (\mathcal{S}, \mathcal{A}, R, T, T_0, \gamma, H)$ with $\mathcal{S}$ a set of states, $\mathcal{A}$ a set of actions, $R(r_{t+1}|s_t, a_t, s_{t+1})$ a reward function, $T(s_{t+1}|s_t, a_t)$ a transition function, $T_0(s_0)$ an initial state distribution, γ a discount factor, and H the horizon. In the standard RL setting, we want to learn a policy π that maximises $\mathcal{J}(\pi) = \mathbb{E}_{T_0,T,\pi}\left[\sum_{t=0}^{H-1} \gamma^t R(r_{t+1}|s_t, a_t, s_{t+1})\right]$, the expected return. Here, we consider a multi-task meta-learning setting, which we introduce next.

2.1 META-LEARNING

We adopt the standard meta-learning setting in which we consider a distribution $p(M)$ over MDPs, where an MDP $M_i \sim p(M)$ is defined by a tuple $M_i = (\mathcal{S}, \mathcal{A}, R_i, T_i, T_{i,0})$. Across tasks, the reward function R_i and transition function T_i can vary, and typically some structure is shared across tasks. The index i represents either task description (e.g., a goal position; a natural language instruction; the leg length of a humanoid) or a task ID. Sampling an MDP from $p(M)$ is typically done by sampling a reward and transition function from a distribution $p(R, T)$.

The distribution over MDPs is unknown to the agent, but it is possible to sample single MDPs for meta-training. At each meta-training iteration, a batch $\mathbf{M} = \{M_i\}_{i=1}^N$ of tasks is drawn, $M_i \sim p(M)$. For each task $M_i \in \mathbf{M}$, we are given a limited number of environment interactions for learning (how to learn), i.e., to maximise performance within that initially unknown MDP (the task description or ID i is unknown). How those environment interactions are used depends on the meta-learning method (see Section 4 for an overview). At meta-test time, the agent is evaluated based on the average return it achieves *during learning*, across a set of tasks drawn from p. Doing this well requires at least two things: (1) incorporating prior knowledge obtained in related tasks, and (2) reasoning about (task) uncertainty when selecting actions to trade off exploration and exploitation. In the following, we combine ideas from meta-learning and Bayesian RL to tackle these challenges.

2.2 BAYESIAN REINFORCEMENT LEARNING

When the MDP is unknown, optimal decision making has to trade off exploration and exploitation when selecting actions. In principle, this can be done by taking a Bayesian approach to reinforcement learning (Bellman, 1956; Duff & Barto, 2002; Ghavamzadeh et al., 2015). In the Bayesian formulation of RL, we assume that the transition and reward functions are distributed according to a prior $b_0 = p(R, T)$. Since the agent does not have access to the true reward and transition function, it can maintain a belief $b_t(R, T)$ at every timestep, which is the posterior over the MDP given the agent's experience: given a trajectory of states, actions, and rewards, $\tau_{:t} = \{s_0, a_0, r_1, s_1, a_1, \ldots, s_t\}$, the prior $p(R, T)$ can be updated to form a posterior belief $b_t(R, T) = p(R, T|\tau_{:t})$. This is often done by maintaining a distribution over the model parameters.

To allow the agent to incorporate the task uncertainty into its decision-making, this belief can be augmented to the state space, $\mathcal{S}^+ = \mathcal{S} \times \mathcal{B}$ where $\mathcal{B}$ is the belief space. States in $\mathcal{S}^+$ are often called *hyper-states*, and they transition according to

$$\begin{aligned} T^+(s_{t+1}^+|s_t^+, a_t, r_t) &= T^+(s_{t+1}, b_{t+1}|s_t, a_t, r_t, b_t) \\ &= T^+(s_{t+1}|s_t, a_t, b_t)\, T^+(b_{t+1}|s_t, a_t, r_t, b_t, s_{t+1}) \\ &= \mathbb{E}_{b_t}\left[T(s_{t+1}|s_t, a_t)\right]\ \delta(b_{t+1} = p(R, T|\tau_{:t+1})) \end{aligned} \tag{1}$$

i.e., the new environment state s_t is the expected new state w.r.t. the current posterior distribution of the transition function, and the belief is updated deterministically according to Bayes rule. The reward function on hyperstates is defined as the expected reward under the current posterior (after the state transition) over reward functions,

$$\begin{aligned} R^+(s_t^+, a_t, s_{t+1}^+) &= R^+(s_t, b_t, a_t, s_{t+1}, b_{t+1}) \\ &= \mathbb{E}_{b_{t+1}}\left[R(s_t, a_t, s_{t+1})\right]. \end{aligned} \tag{2}$$

We can now formulate the Bayes-Adaptive Markov decision process (BAMDP, Duff & Barto (2002)), which consists of the tuple $M = (\mathcal{S}^+, \mathcal{A}, R^+, T^+, T_0^+, \gamma, H)$. This is a special case of a belief MDP, i.e, the MDP formed by taking the posterior beliefs maintained by an agent in a partially observable MDP (POMDP) and reinterpreting them as Markov states (Cassandra et al., 1994).

In an arbitrary belief MDP, the belief is over a hidden state that can change at each timestep. In a BAMDP, the belief is over the transition and reward functions, which are constant for a given task. The agent's objective is now to maximise the expected return in the BAMDP,

$$\mathcal{J}^+(\pi) = \mathbb{E}_{b_0, T_0^+, T^+, \pi}\left[\sum_{t=0}^{H-1} \gamma^t R^+(r_{t+1}|s_t^+, a_t, s_{t+1}^+)\right], \tag{3}$$

i.e., maximise the expected return in an initially unknown environment, while learning, within the horizon H. This objective is maximised by the Bayes-optimal policy, which automatically trades off exploration and exploitation: it takes exploratory actions to reduce its task uncertainty *only insofar as it helps to maximise the expected return within the horizon*.

The BAMDP framework is powerful because it provides a principled way of formulating Bayes-optimal behaviour. However, solving the BAMDP is hopelessly intractable for most interesting problems. The main challenges are as follows.

- We typically do not have access to the prior distribution $p(M)$ but can only sample from it.
- We typically do not know the parameterisation of the true reward and/or transition model. Instead, we can choose to approximate them, for example using deep neural networks.
- The belief update (computing the posterior $p(R, T|\tau_{:t})$) is often intractable.
- Even given the posterior over the reward and transition model, planning in belief space is typically intractable.

In the following, we propose a method that uses meta-learning to do Bayesian reinforcement learning, which amounts to meta-learning a prior over tasks and performing inference over reward and transition functions using deep learning. Crucially, our Bayes-adaptive policy is learned end-to-end with the inference framework, which means that no planning is necessary at test time. It can be applied to the typical meta-learning setting and makes minimal assumptions (no task ID or description is required), resulting in a highly flexible and scalable approach to Bayes-adaptive Deep RL.

3 BAYES-ADAPTIVE DEEP RL VIA META-LEARNING

In this section, we present variBAD, and describe how we tackle the challenges outlined above. We start by describing how to represent reward and transition functions, and (posterior) distributions over these. We then consider how to meta-learn to perform approximate variational inference in a given task, and finally put all the pieces together to form our training objective.

In the typical meta-learning setting, the reward and transition functions that are unique to each MDP are unknown, but also share some structure across the MDPs M_i in $p(M)$. We know that there exists a true i which represents either a task description or task ID, but we do not have access to this information. We therefore represent this value using a learned stochastic latent variable m_i. For a given MDP M_i we can then write

$$R_i(r_{t+1}|s_t, a_t, s_{t+1}) \approx R(r_{t+1}|s_t, a_t, s_{t+1}; m_i), \tag{4}$$

$$T_i(s_{t+1}|s_t, a_t) \approx T(s_{t+1}|s_t, a_t; m_i), \tag{5}$$

where R and T are shared across tasks. Since we do not have access to the true task description or ID, we need to *infer* m_i given the agent's experience up to time step t collected in M_i,

$$\tau_{:t}^{(i)} = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{t-1}, a_{t-1}, r_t, s_t), \tag{6}$$

i.e., we want to infer the posterior distribution $p(m_i|\tau_{:t}^{(i)})$ over m_i given $\tau_{:t}^{(i)}$ (from now on, we drop the sub- and superscript i for ease of notation).

Recall that our goal is to *learn a distribution over the MDPs, and given a posteriori knowledge of the environment compute the optimal action*. Given the above reformulation, it is now sufficient to reason about the embedding m, instead of the transition and reward dynamics. This is particularly useful when deploying deep learning strategies, where the reward and transition function can consist of millions of parameters, but the embedding m can be a small vector.

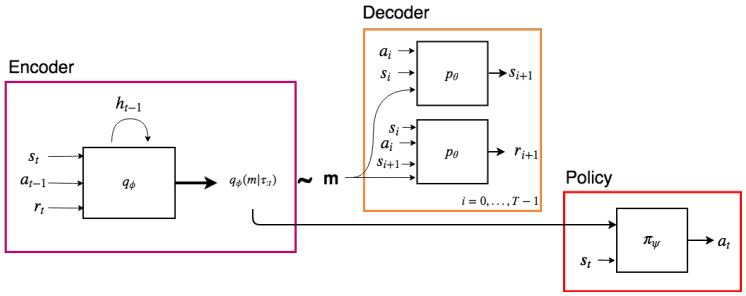


Figure 2: VariBAD: A trajectory of states, actions and rewards is processed online using an RNN to produce the posterior over task embeddings, $q_\phi(m|\tau_{:t})$. The posterior is trained using a decoder which attempts to predict future states and rewards from current states and actions. The policy conditions on the posterior in order to act in the environment and is trained using RL.

3.1 Approximate Inference

Computing the exact posterior is typically not possible: we do not have access to the MDP (and hence the transition and reward function), and marginalising over tasks is computationally infeasible. Consequently, we need to learn a model of the environment $p_\theta(\tau_{:H}|a_{:H-1})$, parameterised by θ, together with an amortised inference network $q_\phi(m|\tau_{:t})$, parameterised by ϕ, which allows fast inference at runtime *at each timestep* t. The action-selection policy is not part of the MDP, so an environmental model can only give rise to a distribution of trajectories when conditioned on actions, which we typically draw from our current policy, $a \sim \pi$. At any given time step t, our model learning objective is thus to maximise

$$\mathbb{E}_{\rho(M,\tau_{:H})}\left[\log p_\theta(\tau_{:H}|a_{:H-1})\right], \tag{7}$$

where $\rho(M, \tau_{:H})$ is the trajectory distribution induced by our policy and we slightly abuse notation by denoting by τ the state-reward trajectories, excluding the actions. In the following, we drop the conditioning on $a_{:H-1}$ to simplify notation.

Instead of optimising (7), which is intractable, we can optimise a tractable lower bound, defined with a learned approximate posterior $q_\phi(m|\tau_{:t})$ which can be estimated by Monte Carlo sampling (for the full derivation see AppendixA):

$$\begin{aligned}\mathbb{E}_{\rho(M,\tau_{:H})}\left[\log p_\theta(\tau_{:H})\right] &\geq \mathbb{E}_\rho\left[\mathbb{E}_{q_\phi(m|\tau_{:t})}\left[\log p_\theta(\tau_{:H}|m)\right] - KL(q_\phi(m|\tau_{:t})||p_\theta(m))\right] \\ &= ELBO_t.\end{aligned} \tag{8}$$

The term $\mathbb{E}_q[\log p(\tau_{:H}|m)]$ is often referred to as the reconstruction loss, and $p(\tau_{:t}|m)$ as the decoder. The term $KL(q(m|\tau_{:t})||p_\theta(m))$ is the KL-divergence between our variational posterior q_ϕ and the prior over the embeddings $p_\theta(m)$. For sufficiently expressive decoders, we are free to choose $p_\theta(m)$. We set the prior to our previous posterior, $q_\phi(m|\tau_{:t-1})$, with initial prior $q_\phi(m) = \mathcal{N}(0, I)$.

As can be seen in Equation 8 and Figure 2, when the agent is at timestep t, we encode the *past trajectory* $\tau_{:t}$ to get the current posterior $q(m|\tau_{:t})$ since this is all the information available to perform inference about the current task. We then decode the entire trajectory $\tau_{:T}$ *including the future*, i.e., model $\mathbb{E}_q[p(\tau_{:T}|m)]$. This is different than the conventional VAE setup (and possible since we have access to this information during training). Decoding not only the past but also the future is important because this way, variBAD learns to perform inference about unseen states given the past. The reconstruction term $\log p(\tau_{:H}|m)$ factorises as

$$\begin{aligned}\log p(\tau_{:H}|m, a_{:H-1}) &= \log p((s_0, r_0, \ldots, s_{t-1}, r_{t-1}, s_t)|m, a_{:H-1}) \\ &= \log p(s_0|m) + \sum_{i=0}^{H-1}\left[\log p(s_{i+1}|s_i, a_i, m) + \log p(r_{i+1}|s_i, a_i, s_{i+1}, m)\right].\end{aligned} \tag{9}$$

Here, $p(s_0|m)$ is the initial state distribution T'_0, $p(s_{i+1}|s_i, a_i; m)$ the transition function T', and $p(r_{i+1}|s_t, a_t, s_{i+1}; m)$ the reward function R'. From now on, we include T'_0 in T' for ease of notation.

3.2 TRAINING OBJECTIVE

We can now formulate a training objective for learning the approximate posterior distribution over task embeddings, the policy, and the generalised reward and transition functions R' and T'. We use deep neural networks to represent the individual components. These are:

1. The encoder $q_\phi(m|\tau_{:t})$, parameterised by ϕ;
2. An approximate transition function $T' = p_\theta(s_{i+1}|s_i, a_i; m)$ and an approximate reward function $R' = p_\theta(r_{i+1}|s_t, a_t, s_{i+1}; m)$ which are jointly parameterised by θ; and
3. A policy $\pi_\psi(a_t|s_t, q_\phi(m|\tau_{:t}))$ parameterised by ψ and dependent on ϕ.

The policy is conditioned on both the environment state and the posterior over m, $\pi(a_t|s_t, q(m|\tau_{:t}))$. This is similar to the formulation of BAMDPs introduced in 2.2, with the difference that we learn a unifying distribution over MDP embeddings, instead of the transition/reward function directly. This makes learning easier since there are fewer parameters to perform inference over, and we can use data from all tasks to learn the shared reward and transition function. The posterior can be represented by the distribution's parameters (e.g., mean and standard deviation if q is Gaussian).

Our overall objective is to maximise

$$\mathcal{L}(\phi, \theta, \psi) = \mathbb{E}_{p(M)} \left[\mathcal{J}(\psi, \phi) + \lambda \sum_{t=0}^{H} ELBO_t(\phi, \theta) \right]. \tag{10}$$

Expectations are approximated by Monte Carlo samples, and the ELBO can be optimised using the reparameterisation trick (Kingma & Welling, 2014). For $t = 0$, we use the prior $q_\phi(m) = \mathcal{N}(0, I)$. Past trajectories can be encoded using, e.g., a recurrent network as in Duan et al. (2016); Wang et al. (2016) (which we did in our experiments) or using an encoder that computes an encoding per (s, a, s', r)-tuple and aggregates them in some way (Zaheer et al., 2017; Garnelo et al., 2018; Rakelly et al., 2019). The network architecture is shown in Figure 2.

By training the VAE with different context lengths t, variBAD can learn how to perform inference online (while the agent is interacting with an environment), and decrease its uncertainty over time given more data. In practice, we may subsample a fixed number of ELBO terms (for random time steps t) for computational efficiency if H is large.

Equation (10) is trained end-to-end, and both the VAE and the RL agent are trained simultaneously. The parameter λ weights the supervised model learning objective against the RL loss. This is necessary since parameters ϕ are shared between the model and the policy. However, we found that not backpropagating the RL loss through the encoder is typically sufficient in practice, which speeds up training considerably. In practice, we therefore optimise the policy and the VAE using different optimisers with different learning rates. The RL agent and the VAE do not necessarily have to be trained with the same data batches. In our implementation, we typically use on-policy algorithms so the policy is only trained with the most recent data. The VAE however is independent of this and can be trained with any of the previous rollouts. To this end, we maintain a separate buffer of trajectories used to compute the ELBO.

4 RELATED WORK

Meta Reinforcement Learning. A prominent model-free meta-RL approach is to utilise the dynamics of recurrent networks for fast adaptation (RL^2, Wang et al. (2016); Duan et al. (2016)). At every time step, the network gets an auxiliary comprised of the preceding action and reward. This allows learning within a task to happen online, entirely in the dynamics of the recurrent network. If we remove the decoder (Fig 2) and the VAE objective (Eq (7)), variBAD reduces to this setting, i.e., the main differences are that we use a stochastic latent variable (an inductive bias for representing uncertainty) together with a decoder to reconstruct previous *and future* transitions / rewards (which acts as an auxiliary loss (Jaderberg et al., 2017) to encode the task in latent space and deduce information about unseen states). Ortega et al. (2019) provide an in-depth discussion of meta-learning sequential strategies and how tp recast memory-based meta-learning within a Bayesian framework.

Another popular approach to meta RL is to learn an initialisation of the network parameters, such that at test time, only a few gradient steps are necessary to achieve good performance (Finn et al.,

2017; Nichol & Schulman, 2018). These methods do not directly account for the fact that the initial policy needs to explore, a problem addressed, a.o., by Stadie et al. (2018) (E-MAML) and Rothfuss et al. (2019) (ProMP). In terms of model complexity, MAML and ProMP are relatively lightweight, since they typically consist of a feedforward policy. RL^2 and variBAD use recurrent modules, which increases model complexity but allows online adaptation. Other methods that perform gradient adaptation at test time are, e.g., Houthooft et al. (2018) who meta-learn a loss function conditioned on the agent's experience that is used at test time so learn a policy (from scratch); and Sung et al. (2017) who learn a meta-critic that can criticise any actor for any task, and is used at test time to train a policy. Compared to variBAD, these methods usually separate exploration (before gradient adaptation) and exploitation (after gradient adaptation) at test time by design, making them less sample efficient.

Skill / Task Embeddings. Learning (variational) task or skill embeddings for meta / transfer reinforcement learning is used in a variety of approaches. Hausman et al. (2018) use approximate variational inference learn an embedding space of skills (with a different lower bound than variBAD). At test time the policy is fixed, and a new embedder is learned that interpolates between already learned skills. Arnekvist et al. (2019) learn a stochastic embedding of optimal Q-functions for different skills, and condition the policy on (samples of) this embedding. Adaptation at test time is done in latent space. Co-Reyes et al. (2018) learn a latent space of low-level skills that can be controlled by a higher-level policy, framed within the setting of hierarchical RL. This embedding is learned using a VAE to encode state trajectories and decode states and actions. Zintgraf et al. (2019) learn a deterministic task embedding trained similarly to MAML (Finn et al., 2017). Similar to variBAD, Zhang et al. (2018) use learned dynamics and reward modules to learn a latent representation which the policy conditions on and show that transferring the (fixed) encoder to new environments helps learning. Perez et al. (2018) learn dynamic models with auxiliary latent variables, and use them for model-predictive control. Lan et al. (2019) learn a task embedding with an optimisation procedure similar to MAML, where the encoder is updated at test time, and the policy is fixed. Sæmundsson et al. (2018) explicitly learn an embedding of the environment model, which is subsequently used for model predictive control (and not, like in variBAD, for exploration). In the field of imitation learning, some approaches embed expert demonstrations to represent the task; e.g., Wang et al. (2017) use variational methods and Duan et al. (2017) learn deterministic embeddings.

VariBAD differs from the above methods mainly in what the embedding represents (i.e., task uncertainty) and how it is used: the policy conditions on the posterior *distribution* over MDPs, allowing it to reason about task uncertainty and trade off exploration and exploitation online. Our objective (8) explicitly optimises for Bayes-optimal behaviour. Unlike some of the above methods, we do not use the model at test time, but model-based planning is a natural extension for future work. Recent work by Humplik et al. (2019) also exploits the idea of conditioning the policy on a latent distribution over task embeddings, which is trained using privileged information such as a task ID, or an expert policy. VariBAD on the other hand can be applied even when such information is not available.

Bayesian Reinforcement Learning. Bayesian methods for RL can be used to quantify uncertainty to support action-selection, and provide a way to incorporate prior knowledge into the algorithms (see Ghavamzadeh et al. (2015) for a review). A Bayes-optimal policy is one that optimally trades off exploration and exploitation, and thus maximises expected return during learning. While such a policy can in principle be computed using the BAMDP framework, it is hopelessly intractable for all but the smallest tasks. Existing methods are therefore restricted to small and discrete state / action spaces (Asmuth & Littman, 2011; Guez et al., 2012; 2013), or a discrete set of tasks (Brunskill, 2012; Poupart et al., 2006). VariBAD opens a path to tractable approximate Bayes-optimal exploration for deep RL by leveraging ideas from meta-learning and approximate variational inference, with the only assumption that we can meta-train on a set of related tasks. Existing approximate Bayesian RL methods often require us to define a prior / belief update on the reward / transition function, and rely on (possibly expensive) sample-based planning procedures. Due to the use of deep neural networks however, variBAD lacks the formal guarantees enjoyed by some of the methods mentioned above.

Posterior sampling (Strens, 2000; Osband et al., 2013), which extends Thompson sampling (Thompson, 1933) from bandits to MDPs, estimates a posterior distribution over MDPs (i.e., model and reward functions), in the same spirit as variBAD. This posterior is used to periodically sample a single hypothesis MDP (e.g., at the beginning of an episode), and the policy that is optimal *for the sampled MDP* is followed subsequently. This approach is less efficient than Bayes-optimal behaviour and therefore typically has lower expected return during learning.

A related approach for inter-task transfer of abstract knowledge is to pose policy search with priors as Markov Chain Monte Carlo inference (Wingate et al., 2011). Similarly Guez et al. (2013) propose a Monte Carlo Tree Search based method for Bayesian planning to get a tractable, sample-based method for obtaining approximate Bayes-optimal behaviour. Osband et al. (2018) note that non-Bayesian treatment for decision making can be arbitrarily suboptimal and propose a simple randomised prior based approach for structured exploration. Some recent deep RL methods use stochastic latent variables for structured exploration (Gupta et al., 2018; Rakelly et al., 2019), which gives rise to behaviour similar to posterior sampling. Other ways to use the posterior to guide exploration are, e.g., certain reward bonuses Kolter & Ng (2009); Sorg et al. (2012) and approaches based on optimism in the face of uncertainty (Kearns & Singh, 2002; Brafman & Tennenholtz, 2002). Non-Bayesian methods for exploration are often used in practice, such as other exploration bonuses (e.g., via state-visitation counts) or by ensuring exploration via uninformed sampling of actions (e.g., ϵ-greedy action selection). In general, such methods are prone to wasteful exploration that does not help maximise expected reward.

Variational Inference and Meta-Learning. A main difference of variBAD to many existing Bayesian RL methods is that we meta-learn the inference procedure, i.e., both the (meaning of the) prior and how to update the posterior, instead of assuming that we have access to the prior or well-behaved distributions for which we can update the posterior analytically. Apart from (RL) methods mentioned above, related work in this direction can be found, a.o., in Garnelo et al. (2018); Gordon et al. (2019); Choi et al. (2019). By comparison, variBAD has a different inference procedure tailored to the setting of learning Bayes-optimal policies for a given distribution over MDPs.

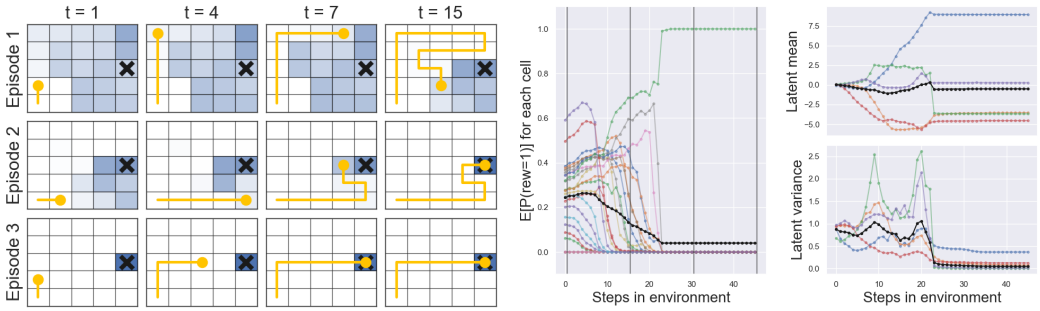
POMDPs. Several deep learning approaches to model-free reinforcement learning (Igl et al., 2019) and model learning for planning (Tschiatschek et al., 2018) in partially observable Markov decision processes have recently been proposed and utilise approximate variational inference methods. VariBAD by contrast focuses on BAMDPs (Martin, 1967; Duff & Barto, 2002; Ghavamzadeh et al., 2015), a special case of POMDPs where the transition and reward functions constitute the hidden state and the agent must maintain a belief over them. While in general the hidden state in a POMDP can change at each time-step, in a BAMDP the underlying task, and therefore the hidden state, is fixed per task. We exploit this property by learning an embedding that is *fixed* over time, unlike approaches like Igl et al. (2019) which use filtering to track the changing hidden state. While we utilise the power of deep approximate variational inference, other approaches for BAMDPs often use more accurate but less scalable methods, e.g., Lee et al. (2019) discretise the latent distribution and use Bayesian filtering for the posterior update.

5 EXPERIMENTS

In this section we first investigate the properties of variBAD on a didactic gridworld domain. We show that variBAD performs *structured* and *online* exploration as it attempts to identify the task at hand. Then we consider more complex meta-learning settings by employing on two MuJoCo continuous control tasks commonly used in the meta-RL literature. We show that variBAD can learn to adapt to the task during the first rollout, unlike most existing meta-learning techniques. Details and hyperparameters can be found in the appendix. We will provide an open-source reference implementation together with the paper.

5.1 GRIDWORLD

We start with a didactic gridworld environment, to gain insight into variBAD's properties and demonstrate the learned policy's ability to perform structured online exploration. The task is to go to a goal (selected uniformly at random) in a 5×5 gridworld. Crucially, the goal is unobserved by the agent, inducing task uncertainty and necessitating exploration. The goal can be anywhere except around the starting cell, which is at the bottom left. The horizon within this environment is $H = 45$, i.e., we train an agent to maximise performance within 45 steps. Actions are: *up, right, down, left, stay* (executed deterministically), and after 15 steps the agent is reset (which looks like an episode break, hence we say that this task has 3 episodes, although strictly speaking it is only one episode with horizon 45). The agent gets a sparse reward signal: -0.1 on non-goal cells, and $+1$ on the goal cell. The best strategy is to explore until the goal is found, and stay at the goal or return to it when reset to the initial position. We use a latent dimensionality of 5.



(a) Example Rollout (b) Reward Predictions (c) Latent Space

Figure 3: Behaviour of variBAD in the gridworld environment. **(a)** Hand-picked but representative example test rollout. The blue background indicates the posterior probability of receiving a reward at that cell. **(b)** Probability of receiving a reward for each cell, as predicted by the decoder, over the course of interacting with the environment (average in black, goal state in green). **(c)** Visualisation of the latent space; each line is one latent dimension, the black line is the average.

Figure 3 illustrates how variBAD behaves at test time with deterministic actions (i.e., all exploration is done by the policy). In 3a we see how the agent interacts with the environment, with the blue background visualising the posterior belief by using the learned reward function. VariBAD learns the correct prior and adjusts its belief correctly over time. It predicts no reward for cells it has visited, and explores the remaining cells until it finds the goal.

A nice property of variBAD is that we can gain insight into the agent's belief about the environment by analysing what the decoder predicts, and how the latent space changes while the agent interacts with the environment. Figure 3b show the reward predictions: each line represents a grid cell and its value the probability of receiving a reward at that cell. As the agent gathers more data, more and more cells are excluded ($p(rew = 1) = 0$), until eventually the agent finds the goal. In Figure 3c we visualise the 5-dimensional latent space. We see that once the agent finds the goal, the posterior concentrates: the variance drops close to zero, and the mean settles on a value.

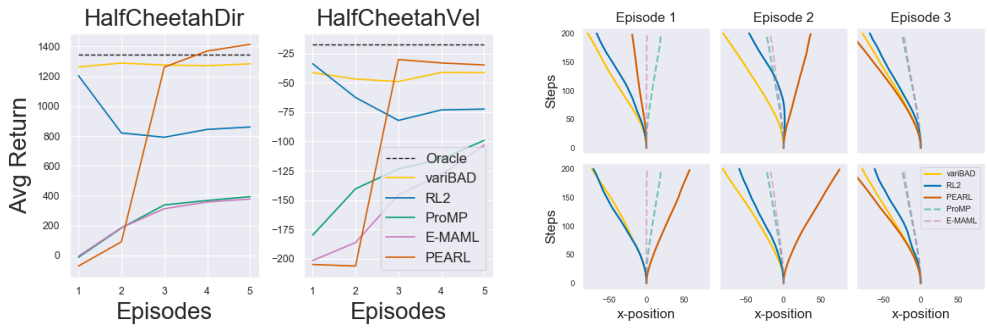
As we showed in Figure 1e, the behaviour of variBAD closely matches that of the Bayes-optimal policy. Recall that the Bayes-optimal policy is the one which optimally trades off exploration and exploitation in an unknown environment, and outperforms posterior sampling. Our results on this gridworld indicate that variBAD is an effective way to approximate Bayes-optimal control. We also tried a similar approach to Duan et al. (2016); Wang et al. (2016), by using a similar architecture as variBAD but a deterministic embedding (of size 64) and no decoder. We observed that this performs worse overall compared to variBAD (see Appendix B for a qualitative and quantitative comparison).

5.2 MuJoCo Continuous Control Meta-Learning Tasks

We show that variBAD is capable of scaling to more complex meta learning settings by employing it on MuJoCo (Todorov et al., 2012) locomotion tasks commonly used in the meta-RL literature.[2] We consider the HalfCheetahDir environment where the agent has to run either forwards or backwards (i.e., there are only two tasks), and the HalfCheetahVel environment where the agent has to run at different velocities (i.e., there are infinitely many tasks). Both environments have a horizon $H = 200$, and we aim to maximise performance within this horizon, i.e., within a single rollout.

Figure 4a shows the performance at test time compared to existing methods. While we show performance for multiple rollouts for the sake of completeness, anything beyond the first rollout is not directly relevant to our goal, which is to maximise performance on a new task, *while learning, within a single episode*. Only variBAD and RL^2 are able to adapt to the task at hand within a single episode. RL^2 underperforms variBAD on the HalfCheetahDir environment, and learning is slower and less stable (see learning curves and runtime comparisons in Appendix C). Even though the first rollout includes exploratory steps, this matches the optimal oracle policy (which is conditioned on

[2]Environments taken from `https://github.com/katerakelly/oyster`.



(a) Performance at test time

(b) Behaviour at test time

Figure 4: **(a)** Performance at test time for HalfCheetahDir and HalfCheetahVel, for the first 7 rollouts. Shown is the average return over different environments, averaged over five seeds. **(b)** Behaviour at test time for the HalfCheetahDir environment for the task "walk left". The x-axis reflects the position of the agent, and the y-axis the steps in the environment (to be read from bottom to top). Rows are separate examples, columns the number of rollouts.

the true task description) up to a small margin. The other methods (PEARL Rakelly et al. (2019), E-MAML Stadie et al. (2018) and ProMP Rothfuss et al. (2019)) are not designed to maximise reward during a single rollout, and perform poorly in this case. They all require substantially more environment interactions in each new task to achieve good performance. PEARL, which is akin to posterior sampling, only starts performing well starting from the third episode (Note that PEARL outperforms our oracle slightly, likely because our oracle is based on PPO, whereas PEARL is based on SAC). E-MAML and ProMP typically use around 20-40 rollouts for the gradient update, which is far less sample efficient than variBAD. In Figure 4a, we show the performance with smaller batch sizes ($0 - 4$ rollouts), collected with the initial policy, and by performing a gradient update also on the learned initialisation.

To get a sense for where these differences might stem from, consider Figure 4b which shows example behaviour of the policies during the first three rollouts in the HalfCheetahDir environment, when the task is "go left". Both variBAD and RL^2 adapt to the task online, whereas PEARL acts according to the current sample, which in the first two rollouts can mean walking in the wrong direction. While we outperform at meta-test time, PEARL is more sample efficient *during meta-training* (see learning curves in Appendix C), since it is an off-policy method. Extending variBAD to off-policy methods is an interesting but orthogonal direction for future work. Overall, our empirical results confirm that variBAD can scale up to current benchmarks and maximise expected reward within a single episode.

6 Conclusion

In this paper we presented variBAD, a novel deep reinforcement learning method to approximate Bayes-optimal behaviour. We used the meta-learning framework to utilise knowledge obtained in related tasks, and perform approximate inference over a learned, low-dimensional latent representation of the MDP. In a didactic gridworld environment, we showed that our agent closely matches the behaviour of the Bayes-optimal policy, which is the policy that optimally trades off exploration and exploitation. We further showed that variBAD can be scaled up to challenging MuJoCo tasks, and that it outperforms existing methods in terms of achieved reward during a single episode. In summary, we believe variBAD opens a path to tractable approximate Bayes-optimal exploration for deep reinforcement learning.

REFERENCES

Isac Arnekvist, Danica Kragic, and Johannes A Stork. Vpe: Variational policy embedding for transfer reinforcement learning. In *International Conference on Robotics and Automation (ICRA)*, 2019.

John Asmuth and Michael L Littman. Learning is planning: near bayes-optimal reinforcement learning via monte-carlo tree search. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2011.

Richard Bellman. A problem in the sequential design of experiments. *Sankhyā: The Indian Journal of Statistics (1933-1960)*, 16(3/4):221–229, 1956.

Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, pp. 3:213–231, 2002.

Emma Brunskill. Bayes-optimal reinforcement learning for discrete uncertainty domains. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 3:1385–1386, 2012.

Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Twelfth National Conference on Artificial Intelligence*, 1994. AAAI Classic Paper Award, 2013.

Kristy Choi, Mike Wu, Noah Goodman, and Stefano Ermon. Meta-amortized variational inference and learning. In *International Conference on Learning Representation (ICLR)*, 2019.

John D Co-Reyes, YuXuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *International Conference on Machine Learning (ICML)*, 2018.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1087–1098, 2017.

Michael O'Gordon Duff and Andrew Barto. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusetts at Amherst, 2002.

Akram Erraqabi, Alessandro Lazaric, Michal Valko, Emma Brunskill, and Yun-En Liu. Rewards and errors in multi-arm bandits for interactive education. 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, 2017.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. In *ICML 2018 Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, Aviv Tamar, et al. Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 8(5-6):359–483, 2015.

Jonathan Gordon, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard E Turner. Meta-learning probabilistic inference for prediction. In *International Conference on Learning Representation (ICLR)*, 2019.

Arthur Guez, David Silver, and Peter Dayan. Efficient bayes-adaptive reinforcement learning using sample-based search. In *Advances in Neural Processing Systems (NeurIPS)*, pp. 1025–1033, 2012.

Arthur Guez, David Silver, and Peter Dayan. Scalable and efficient bayes-adaptive reinforcement learning based on monte-carlo tree search. *Journal of Artificial Intelligence Research*, 48:841–883, 2013.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Processing Systems (NeurIPS)*, 2018.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representation (ICLR)*, 2018.

Rein Houthooft, Yuhua Chen, Phillip Isola, Bradly Stadie, Filip Wolski, OpenAI Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5400–5409, 2018.

Jan Humplik, Alexandre Galashov, Leonard Hasenclever, Pedro A Ortega, Yee Whye Teh, and Nicolas Heess. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*, 2019.

Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for pomdps. In *International Conference on Machine Learning (ICML)*, 2019.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representation (ICLR)*, 2017.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2-3):209–232, 2002.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representation (ICLR)*, 2014.

J Zico Kolter and Andrew Y Ng. Near-bayesian exploration in polynomial time. In *International Conference on Machine Learning (ICML)*, pp. 513–520, 2009.

Lin Lan, Zhenguo Li, Xiaohong Guan, and Pinghui Wang. Meta reinforcement learning with task embedding and shared policy. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

Gilwoo Lee, Brian Hou, Aditya Mandalika, Jeongseok Lee, and Siddhartha S Srinivasa. Bayesian policy optimization for model uncertainty. In *International Conference on Learning Representation (ICLR)*, 2019.

Yun-En Liu, Travis Mandel, Emma Brunskill, and Zoran Popovic. Trading off scientific knowledge and user learning with multi-armed bandits. In *EDM*, pp. 161–168, 2014.

James John Martin. *Bayesian decision problems and Markov chains*. Wiley, 1967.

Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2018.

Pedro A Ortega, Jane X Wang, Mark Rowland, Tim Genewein, Zeb Kurth-Nelson, Razvan Pascanu, Nicolas Heess, Joel Veness, Alex Pritzel, Pablo Sprechmann, et al. Meta-learning of sequential strategies. *arXiv preprint arXiv:1905.03030*, 2019.

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

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8626–8638, 2018.

Christian F Perez, Felipe Petroski Such, and Theofanis Karaletsos. Efficient transfer learning and online adaptation with latent variable models for continuous control. In *Continual Learning Workshop, NeurIPS 2018*, 2018.

Pascal Poupart, Nikos Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete bayesian reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 697–704, 2006.

Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International Conference on Machine Learning (ICML)*, 2019.

Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. Promp: Proximal meta-policy search. In *International Conference on Learning Representation (ICLR)*, 2019.

Steindór Sæmundsson, Katja Hofmann, and Marc Peter Deisenroth. Meta reinforcement learning with latent variable gaussian processes. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

Jonathan Sorg, Satinder Singh, and Richard L Lewis. Variance-based rewards for approximate bayesian reinforcement learning. In *Conference on Uncertainty in Artificial Intelligence (UAI2010)*, 2012.

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. In *Advances in Neural Processing Systems (NeurIPS)*, 2018.

Malcolm Strens. A bayesian framework for reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 2000, pp. 943–950, 2000.

Flood Sung, Li Zhang, Tao Xiang, Timothy Hospedales, and Yongxin Yang. Learning to learn: Meta-critic networks for sample efficient learning. *arXiv preprint arXiv:1706.09529*, 2017.

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.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pp. 5026–5033. IEEE, 2012. ISBN 978-1-4673-1737-5. URL `http://dblp.uni-trier.de/db/conf/iros/iros2012.html#TodorovET12`.

Sebastian Tschiatschek, Kai Arulkumaran, Jan Stühmer, and Katja Hofmann. Variational inference for data-efficient model learning in pomdps. *arXiv preprint arXiv:1805.09281*, 2018.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. In *Annual Meeting of the Cognitive Science Community (CogSci)*, 2016.

Ziyu Wang, Josh S Merel, Scott E Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess. Robust imitation of diverse behaviors. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5320–5329, 2017.

David Wingate, Noah D Goodman, Daniel M Roy, Leslie P Kaelbling, and Joshua B Tenenbaum. Bayesian policy search with policy priors. In *International Joint Conference on Artificial Intelligence (ICJAI)*, 2011.

Gregory Yauney and Pratik Shah. Reinforcement learning with action-derived rewards for chemotherapy and clinical trial dosing regimen selection. In *Machine Learning for Healthcare Conference*, pp. 161–226, 2018.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Processing Systems (NeurIPS)*, pp. 3391–3401, 2017.

Amy Zhang, Harsh Satija, and Joelle Pineau. Decoupling dynamics and reward for transfer learning. In *ICLR workshop track*, 2018.

Luisa M Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning (ICML)*, 2019.

Bayes-Adaptive Deep Reinforcement Learning via Meta-Learning

Supplementary Material

A FULL ELBO DERIVATION

Equation (8) can be derived as follows.

$$
\begin{aligned}
\mathbb{E}_{\rho(M,\tau_{:H})}\left[\log p_\theta(\tau_{:H})\right] &= \mathbb{E}_\rho\left[\log \int p_\theta(\tau_{:H},m)\frac{q_\phi(m|\tau_{:t})}{q_\phi(m|\tau_{:t})}dm\right] \\
&= \mathbb{E}_\rho\left[\log \mathbb{E}_{q_\phi(m|\tau_{:t})}\left[\frac{p_\theta(\tau_{:H},m)}{q_\phi(m|\tau_{:t})}\right]\right] \\
&\geq \mathbb{E}_{\rho,\ q_\phi(m|\tau_{:t})}\left[\log \frac{p_\theta(\tau_{:H},m)}{q_\phi(m|\tau_{:t})}\right] \\
&= \mathbb{E}_{\rho,\ q_\phi(m|\tau_{:t})}\left[\log p_\theta(\tau_{:H}|m) + \log p_\theta(m) - \log q_\phi(m|\tau_{:t})\right] \\
&= \mathbb{E}_\rho\left[\mathbb{E}_{q_\phi(m|\tau_{:t})}\left[\log p_\theta(\tau_{:H}|m)\right] - KL(q_\phi(m|\tau_{:t})||p_\theta(m))\right] \qquad (11) \\
&= ELBO_t.
\end{aligned}
$$

B EXPERIMENTS: GRIDWORLD

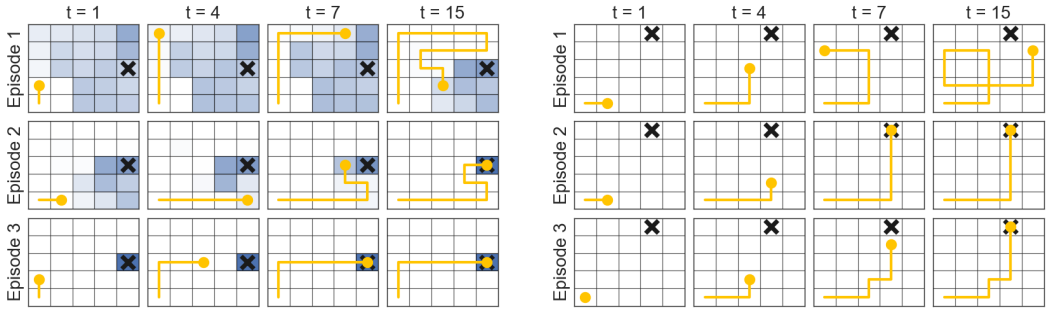
B.1 ADDITIONAL REMARKS

Figure 3c visualises how the latent space changes as the agent interacts with the environment. As we can see, the value of the latent dimensions starts around mean 1 and variance 0, which is the prior we chose for the beginning of an episode. Given that the variance increases for a little bit before the agent finds the goal, this prior might not be optimal. A natural extension of variBAD is therefore to also learn the prior to match the task at hand.

B.2 HYPERPARAMETERS

We used the PyTorch framework for our experiments. Hyperparameters can be found below.

RL Algorithm	A2C
Number of policy steps (A2C)	10
Epsilon (A2C)	1e-5
Gamma (A2C)	0.99
Max grad norm (A2C)	0.5
Value loss coefficient (A2C)	0.5
Entropy coefficient (A2C)	0.01
GAE parameter tau (A2C)	0.95
Number of parallel processes (A2C)	16
ELBO loss coefficient	1.0
Policy LR	0.001
Policy VAE	0.001
Task embedding size	5
Policy architecture	2 hidden layers, 32 nodes each, TanH activations
Encoder architecture	FC layer with 40 nodes, GRU with hidden size 64, output layer with 10 outputs (μ and σ), ReLu activations
Reward decoder architecture	2 hidden layers, 32 nodes each, 25 outputs heads, ReLu activations
Decoder loss function	Binary cross entropy



(a) variBAD

(b) Recurrent policy

Figure 5: Hand-picked but representative example test rollouts for variBAD (a) and an RNN-based policy (b). The blue background indicates the posterior probability of receiving a reward at that cell.

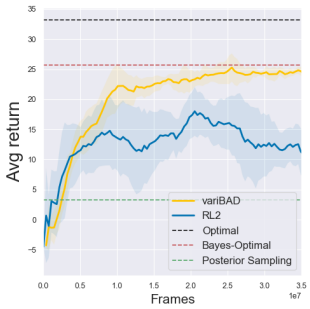


Figure 6: Learning curves for the gridworld environment. Results are averages over 5 seeds, with 95% confidence intervals.

B.3 Comparison to RL2

Figure 5 shows the behaviour of variBAD in comparison to a recurrent policy, an architecture which resembles the approach presented by Duan et al. (2016) and Wang et al. (2016). As we can see, the recurrent policy re-visits states it has already seen before, indicating that its does task inference less efficiently. We believe that the stochastic latent embedding helps the policy express task uncertainty better, and that the auxiliary loss of decoding the embedding help learning.

Figure 6 shows the learning curves for this environment (for the full horizon 45) for variBAD and RL2, in comparison to the hard-coded optimal policy (which has access to the goal position), Bayes-optimal policy, and posterior sampling policy (evaluated on a horizon of $H = 45$, i.e., the first three "episodes" shown in Figure 1). As we can see, variBAD closely approximates Bayes-optimal behaviour, and outperforms RL2.

C MuJoCo

C.1 Learning Curves

Figure 7 shows the learning curves for the MuJoCo environments for all approaches. The oracle policy was trained using PPO. Our approach for hyperparameter-tuning was to tune the hyperparameters for PPO using the oracle policy, and then using these for both variBAD and RL2, only further tuning those aspects particular to those methods (mostly VAE parameters and latent dimension). PEARL (Rakelly et al., 2019) was trained using the reference implementation provided by the authors. The environments we used are also taken from this implementation. E-MAML (Stadie et al., 2018) and ProMP (Rothfuss et al., 2019) were trained using the reference implementation provided by Rothfuss et al. (2019).

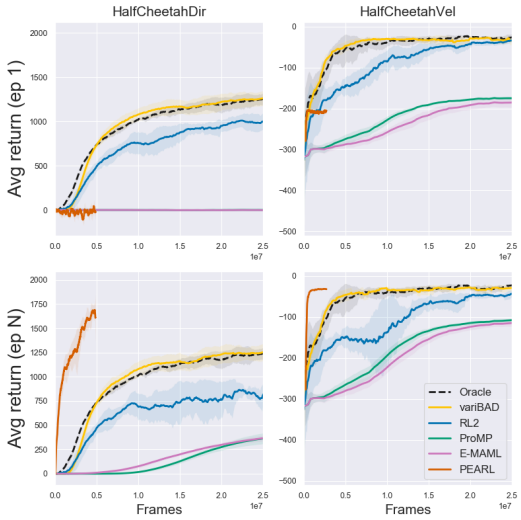


Figure 7: Learning curves for the MuJoCo results presented in Section 5.2. The top row shows performance evaluated at the first rollout, and the second row shows the performance at the N-th rollout. For variBAD and RL2, $N = 2$. For ProMP and E-MAML, $N = 20$. For PEARL, $N = 10$.

As we can see, PEARL is much more sample efficient in terms of number of frames than the other methods (Fig 7), which is because it is an off-policy method. On-policy vs off-policy training is an orthogonal issue to our contribution, but an extension of variBAD to off-policy methods is an interesting direction for future work. Doing posterior sampling using off-policy methods also requires PEARL to use a different encoder (to maintain order invariance of the sampled trajectories) which is non-recurrent (and hence faster to train, see next section) but restrictive since it assumes independence between individual transitions.

C.2 Runtime Comparison

The following are rough estimates of average run-times for the HalfCheetah-Dir environment (from what we have experienced; we often ran multiple experiments per machine, so some of these might be overestimated and should be mostly understood as giving a relative sense of ordering).

- ProMP, E-MAML: 5-8 hours
- variBAD: 48 hours
- RL^2: 60 hours
- PEARL: 24 hours

E-MAML and ProMP have the advantage that they do not have a recurrent part such as variBAD or RL^2. Forward and backward passes through recurrent networks can be slow, especially with large horizons.

Even though both variBAD and RL^2 use recurrent modules, we observed that variBAD is faster when training the policy with PPO. This is because we do not backpropagate the RL-loss through the recurrent part, which allows us to make the PPO mini-batch updates without having to re-compute the embeddings (so it saves us a lot of forward/backward passes through the recurrent model). Additionally, we This difference should be less pronounced with other RL methods that do not rely on this many forward/backward passes per policy update.

C.3 Additional Remarks

Note that even though we maximise performance for a horizon of $H = 200$ for both RL2 and variBAD, we sometimes keep the posterior distribution (or the hidden state of the RNN) which was

obtained during one rollout when resetting the environment. This is to make sure that the agent can learn how to act for more than one episode.

We observe that RL2 is unstable when it comes to maintaining its performance over multiple rollouts. We hypothesize this is due to the specific property of the HalfCheetah environments that the state can give information about the task if the agent has already adapted. E.g., in the HalfCheetahDir environment, the x-position is part of the state observed by the agent. At the beginning of the episode, when starting close to $x = 0$, the agent has to infer in which direction to run. However, once it moves further and further away from the origin, it is sufficient to rely on the actual environment state to infer which task the agent is currently in. This could lead to the hidden state of the recurrent part of the network being ignored (and taking values that the agent cannot interpret), such that when reset to the origin, the inference procedure has to be re-done. VariBADdoes not have this problem, since we train the latent embedding to represent the task, and only the task. Therefore, the agent does not have to do the inference procedure again when reset to the starting position, but can rely on the latent task description that is given by the approximate posterior.

C.4 HYPERPARAMETERS

We used the PyTorch framework for our experiments. Hyperparameters can be found below.

RL Algorithm	PPO
Number of policy steps	200
Epochs (PPO)	2
Minibatches	4
Max grad norm (PPO)	0.5
Clip parameter (PPO)	0.2
Value loss coefficient (PPO)	0.5
Entropy coefficient (PPO)	0.01
Number of parallel processes (PPO)	16
Notes	We use a huber loss in the RL loss
ELBO loss coefficient	1.0
Policy LR	0.0007
Policy VAE	0.001
Task embedding size	5
Number of frames used for training	5e+7
Policy architecture	2 hidden layers, 128 nodes each, TanH activations
Encoder architecture	FC layer with 208 nodes, GRU with hidden size 64, output layer with 5 outputs, ReLu activations
Reward decoder architecture	2 hidden layers, 32 nodes each, ReLu activations
Reward decoder loss function	Mean squared error
State decoder architecture	2 hidden layers, 64 and 64 nodes, ReLu activations
State decoder loss function	Mean squared error