Fast Adaptation via Meta Reinforcement Learning

Luisa Zintgraf

Linacre College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2022

To my family

Gabriele, Reinhold, and Chiara

♡

Acknowledgements

I have been very fortunate to work with Shimon Whiteson and Katja Hofmann as my DPhil supervisors. Thank you for taking a chance on me, for your generous guidance and support, and your extensive contributions to the research in this thesis. I have learned a lot from you both, and I'm going to miss you.

Shimon, thank you for your perspectives and critical feedback throughout the years. You gave me all the time and freedom I needed to choose my own direction, yet influenced my research profoundly - not least by introducing me to BAMDPs. Your intellect and high standards continue to inspire me.

Katja, thank you for being a great role model and for helping me become a more confident and better researcher! You taught me how to strategise about my research and work through problems, and introduced me to many amazing researchers and opportunities over the years. Your support and encouragement means a lot to me.

For the work in this thesis I was lucky to collaborate with Kyriacos Shiarlis, Vitaly Kurin, Maximilian Igl, Sebastian Schulze, Yarin Gal, Cong Lu, Leo Feng, Sam Devlin, Kamil Ciosek, Kristian Hartikainen, Zheng Xiong, Jacob Beck, and Risto Vuorio. Thank you for your contributions - I could not have done it without you! To everyone else at WhiRL, thanks for all the happy times: Jakob Foerster, Supratik Paul, Tabish Rashid, Mingfei Sun, Matthew Fellows, Matt Smith, Jelena Luketina, Christian Schroeder de Witt, Wendelin Boehmer, Ben Ellis, Greg Farquhar, Anuj Mahajan, Bei Peng, Shangtong Zhang, Charline Le Lan, Tim Rocktäschel, and Tarun Gupta.

My sincere appreciation goes to Pieter Abbeel and Phil Blunsom for their time and expertise to examine this thesis.

I did two wonderful internships during my DPhil, at Microsoft Research Cambridge and DeepMind London. Thank you to my colleagues at MSR, including Katja Hofmann, Sam Devlin, Kamil Ciosek, Hisham Husain, and the Game Intelligence team, for making this internship so enjoyable. I loved coming to the office and am glad we got to spend some in-person time together before we started working from home "for two weeks" due to COVID. To my colleagues at DeepMind, including

Tom Schaul, Iurii Kemaev, Zita Marinho, Junhyuk Oh, Louis Kirsch, and the RL team – thank you for making me feel so welcome despite the physical distance, being invested in my internship project, and our engaging discussions.

I am indebted to Microsoft Research for funding my research generously, with the 2017 Microsoft Research PhD Scholarship Program, and the 2020 Microsoft Research EMEA PhD Award.

Personal

To Joost van Amersfoort, thank you for all your love, encouragement, and support! I'm happy we went through this DPhil experience together and that I could always rely on you. I loved celebrating our successes at the Chester Arms, our pre-pandemic travels, our post-pandemic coffee and cracking-the-cryptic obsession, the puzzle hunts we did for each other, and our joint thesis-writing sessions in the Duke Humfrey's Library. I look forward to post-student-life and laptop-free holidays with you.

My deepest gratitude to Diederik Roijers for being an amazing mentor and friend. Your encouragement and guidance throughout the years - from publishing my first paper while at the University of Amsterdam, to suggesting to apply for a DPhil at Oxford - had a huge positive impact on my life. I wouldn't be here without it!

To my friends near and far, thank you for bearing with me and for enriching my life in so many ways over the past few years, especially to Laura Fichtner, Marie Schölmerich, Saskia Schmidt, Sissy Rajan, Lisa Schut, Feryal Behbahani, Kyriacos Shiarlis, Jonas von Hoffmann, Eva Schlindwein, Panagiotis Tigas, Bas Veeling, Hüseyin Can Akpınar, Otto Fabius, and Yvonne Koper. Thank you to my extended family Deborah Mijlof, Emma, Irene and Steef van Amersfoort, and Wilma van Baarsen, for their support and Oxford visits.

To my parents Gabriele and Reinhold, I am forever grateful for the opportunities and encouragement that you gave me, and that ultimately led me here. Your unconditional love and trust in me means more than words can express. To my sister and Lieblingsmensch Chiara, thank you for being my biggest fan and for all the video stories, Mr Bean gifs, phone calls, and flowers that kept me going!

Like all good things, my DPhil – and these acknowledgements – must come to an end. I loved every moment of it, even the hard parts (and there were a few), and I'm excited for my onward journey. See you on the other side!

Princes Street, Oxford, April 2022

Abstract

Reinforcement Learning (RL) is a way to train artificial agents to autonomously interact with the world. In practice however, RL still has limitations that prohibit the deployment of RL agents in many real world settings. This is because RL takes long, typically requires human oversight, and produces specialised agents that can behave unexpected in unfamiliar situations. This thesis is motivated by the goal of making RL agents more flexible, robust, and safe to deploy in the real world. We develop agents capable of *Fast Adaptation*, i.e., agents that can learn new tasks efficiently.

To this end, we use *Meta Reinforcement Learning* (Meta-RL), where we teach agents not only to act autonomously, but to *learn* autonomously. We propose four novel Meta-RL methods based on the intuition that adapting fast can be divided into "task inference" (understanding the task) and "task solving" (solving the task). We hypothesise that this split can simplify optimisation and thus improve performance, and is more amenable to downstream tasks. To implement this, we propose a context-based approach, where the agent conditions on a context that represents its current knowledge about the task. The agent can then use this to decide whether to learn more about the task, or try and solve it.

In Chapter 5, we use a deterministic context and establish that this can indeed improve performance and adequately captures the task. In the subsequent chapters, we then introduce Bayesian reasoning over the context, to enable decision-making under *task uncertainty*. By combining Meta-RL, context-based learning, and approximate variational inference, we develop methods to compute approximately Bayes-optimal agents for single-agent settings (Chapter 6) and multi-agent settings (Chapter 7). Finally, Chapter 8 addresses the challenge of meta-learning with sparse rewards, which is an important setting for many real-world applications. We observe that existing Meta-RL methods can fail entirely if rewards are sparse, and propose a way to overcome this by encouraging the agent to explore during meta-training. We conclude the thesis with a reflection on the work presented in the context of current developments, and a discussion of open questions.

In summary, the contributions in this thesis significantly advance the field of Fast Adaptation via Meta-RL. The agents develop in this thesis can adapt faster than any previous methods across a variety of tasks, and we can compute approximately Bayes-optimal policies for much more complex task distributions than previously possible. We hope that this helps drive forward Meta-RL research and, in the long term, using RL to address important real world challenges.

Contents

1 Introduction

Artificial Intelligence (AI) plays an increasingly large role in our lives, from real-time route planning, to shopping recommendations, to video game bots. In the future, autonomous agents might even drive our cars, curate personalised treatment plans, or support government policy-making. A general approach to solving these types of problems is Reinforcement Learning (RL), where an agent learns to interact with the world via trial-and-error. The agent's goal is to accumulate as many rewards as possible, taking both the short and long-term effects of its actions into account. Decades of RL research, on topics ranging from policy gradients (Sutton et al., 1999; Schulman et al., 2015a) to exploration (Thrun, 1992; Bellemare et al., 2016), have enabled us to develop agents that play Go at super-human level (Silver et al., 2016), navigate stratospheric balloons (Bellemare et al., 2020), or control robot hands to manipulate objects with high dexterity (Andrychowicz et al., 2020).

Part of the reason that these agents can be trained to such high performance is that in the above settings, we can control, predict, or simulate the impact of an agent's actions well. In most other real world settings, however, this is *not* the case, and RL agents could behave in an unpredictable, undesired, or outright dangerous way. To safely deploy RL agents in those settings, we still have a long way to go. This thesis is motivated by the quest of enabling RL to develop agents that are more flexible, robust, and safe to deploy in the real world.

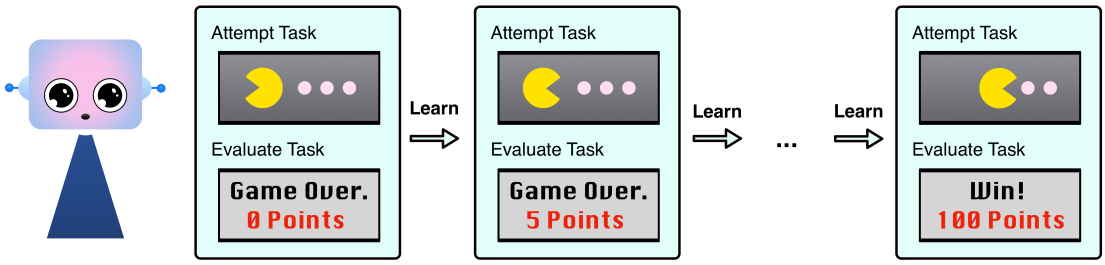


Figure 1.1: Simplified illustration of RL training. The agent learns to play the video game Ms PacMan by repeatedly playing and learning from the game outcomes. Over time, the agent improves until it eventually masters the game. This may take tens of thousands of game plays. Humans, on the other hand, can pick up games like Ms PacMan in a handful of trials by transferring knowledge and skills from other areas of life.

To understand how Reinforcement Learning is performed, consider the simplified schematic in Figure 1.1 of how an RL agent learns to play the game Ms PacMan. At the beginning, the agent will behave randomly, jitter back and forth, and lose the game. Over time it becomes better by avoiding actions with negative consequences, and reinforcing behaviour that results in positive game points. After up to *tens of thousands* of games, the agent is proficient in Ms PcMan – and only Ms PacMan. For a different game, we would have to train it from scratch.

This example illustrates how powerful, but also how brittle, RL agents can be. The fact that a computer algorithm can learn a game like this on its own, by trial and error and receiving game points, is spectacular. But it also has limitations, and is far from how humans – or any intelligent species – learn. By contrast, it is **slow**: many RL algorithms require millions or even billions of environment interactions to learn, rendering them practically useless for real world settings unless high fidelity simulators exist. Most algorithms also learn from scratch, starting with a **random behaviour policy** and little domain knowledge. This makes them flexible, allowing us to use the same algorithm across many applications. But it also prevents us from training agents directly in the real world, where the consequences of random behaviour could be disastrous. Lastly, RL algorithms tend to produce **specialised** agents that cannot handle even the slightest deviations from what they have seen during training. The real world is not as predictable as Ms PacMan, and an RL agent would take forever to achieve meaningful capabilities.

1.1 Autonomous Agents for the Real World

When using Reinforcement Learning to develop and deploy autonomous agents, we can consider two ends of a spectrum: deploying fully-trained agents that have been pre-trained (e.g., in simulation), or training them from scratch directly in the real world. We discuss these options below, and use them to contextualise this thesis.

Deploying Fully Trained Agents In some cases, it is possible to pre-train RL agents before letting them interact directly with the real world. One such case is if we have simulators available to train our agents. The fact that we can let agents practice in simulation for millions of steps before letting them interact with the real world is part of the success story of AlphaGo (Silver et al., 2016), the RL agent that beat the world's strongest Go player in 2016. Simulators are also used in many robotics applications (Peng et al., 2018; Du et al., 2021; Körber et al., 2021; Collins et al., 2021), to train controllers before deploying them on real robots. More real world examples where simulators are used include stratospheric balloon navigation (Bellemare et al., 2020), thermal control in smart buildings (Gao et al., 2019), or autonomous driving (Kiran et al., 2021). Another way to pre-train agents is to not learn via interaction with an environment (i.e., a simulator or the real world), but instead learn *offline* from a dataset (Levine, 2021; Hussein et al., 2017). For example, the aforementioned algorithm AlphaGo (Silver et al., 2016) also learned from human player data, and only a later variant could learn to master the game entirely on its own (Silver et al., 2017). Other examples of using human data to train RL agents are simulated humanoid football (Liu et al., 2021b), StarCraft II (Vinyals et al., 2019), and autonomous driving (Seita, 2018).

However, deploying fully trained agents is not always feasible (e.g., if we do not have access to simulators or real world data), or desirable (e.g., it may produce agents that are overspecialised, and unpredictable if they encounter new situations). In this case, we can consider training agents directly in the real world.

Learning from Scratch in the Real World The alternative of training agents directly in the real world comes with many requirements. First and foremost, it should be safe. Research on AI Safety (Amodei et al., 2016) is concerned with developing risk-sensitive agents (Mihatsch et al., 2002; Delétang et al., 2021), robustness to distribution shift (Quiñonero-Candela et al., 2008; Li et al., 2011), or preventing agents from collecting high rewards in unintended ways (Everitt et al., 2016; Hadfield-Menell et al., 2017), to name a few. Moreover, learning in the real world should happen within reasonable time, require limited human oversight, and be resource and cost-efficient. If training a robot to bartend takes several months with constant supervision and thousands of broken bottles and glasses, we would probably not consider this a serious option. Strategies to overcome this and make RL more efficient include integrating domain knowledge (Teng et al., 2014; Silva et al., 2019), making use of natural language (Luketina et al., 2019; Mu et al., 2022), learning from human preferences (Christiano et al., 2017; Griffith et al., 2013), or learning from demonstrations (Hussein et al., 2017; Argall et al., 2009). To date, however, these do not provide the full answer to training agents from scratch in the real world, especially for safety-critical settings like autonomous driving or healthcare.

Continued Learning in the Real World Naturally, most practical solutions will lie somewhere between deploying fully trained agents, and training agents from scratch in the real world. This is already common practice in robotics, where controllers are often pre-trained in simulation and then fine-tuned it in the real world (Smith et al., 2021; Kolter et al., 2007; Julian et al., 2021). This, however, typically requires controlled settings and human oversight, which is unfeasible in many other applications. In general, we are far from a general and robust way to deploy or train RL agents in our chaotic and ever-changing world. We need agents to learn substantially faster than currently possible, ideally without the need for constant human oversight and intervention. This thesis addresses key aspects of these challenges using an approach called *Meta Reinforcement Learning*, where we pre-train agents to not only *act* on their own, but *learn* on their own.

1.2 Meta Reinforcement Learning

Central to this thesis is the idea of *Meta-Learning*, or *Learning to Learn*, which allows for dramatically more efficient learning of new tasks. The idea of learning to learn goes back to at least until 1987 (Schmidhuber, 1987; Thrun et al., 1998), with renewed interest and research since about 2016, thanks to modern advances in Deep Learning (for recent surveys see Hospedales et al., 2020; Huisman et al., 2021). Unlike standard Machine Learning algorithms, a Meta-Learning algorithm does not just learn to solve individual tasks – instead, it learns how to learn. In other words, rather than inventing and implementing all components of an algorithm by hand, we now *learn* some parts using a data-driven approach. This is a way to include domain knowledge into the algorithm (Botvinick et al., 2019; Hessel et al., 2019), and can be much easier than doing so by hand. In Meta Reinforcement Learning (Meta-RL), the idea of learning to learn can be used to develop RL agents that can learn efficiently and independently.

We illustrate the Meta-RL procedure in Figure 1.2. Contrary to standard RL (Figure 1.1), we do not just optimise performance for a single task. In Meta-RL, we optimise *learning performance* across different tasks. Learning performance can be defined in several ways, such as the end performance after learning for a fixed duration, or the time it takes for the agent to learn a task (see Section 3.2). By repeatedly attempting to learn new tasks and evaluating learning performance (horizontal axis in Figure 1.2), the agent gets better at learning (vertical axis). Eventually, it can learn new tasks much faster than existing RL algorithms.

While much of the research in Meta-RL is motivated by the vision of real world applications, many other exciting reasons exist – like scientific curiosity, the prospect of discovering new RL algorithms (which may have consequences far beyond what we can imagine today), and speeding up RL research itself. In this thesis, we leverage Meta-RL for Fast Adaptation, where the goal is to develop *agents* that can *adapt efficiently* to new tasks. Such an agent should learn a game like Ms PacMan much faster than current RL algorithms, within just a handful of trials.

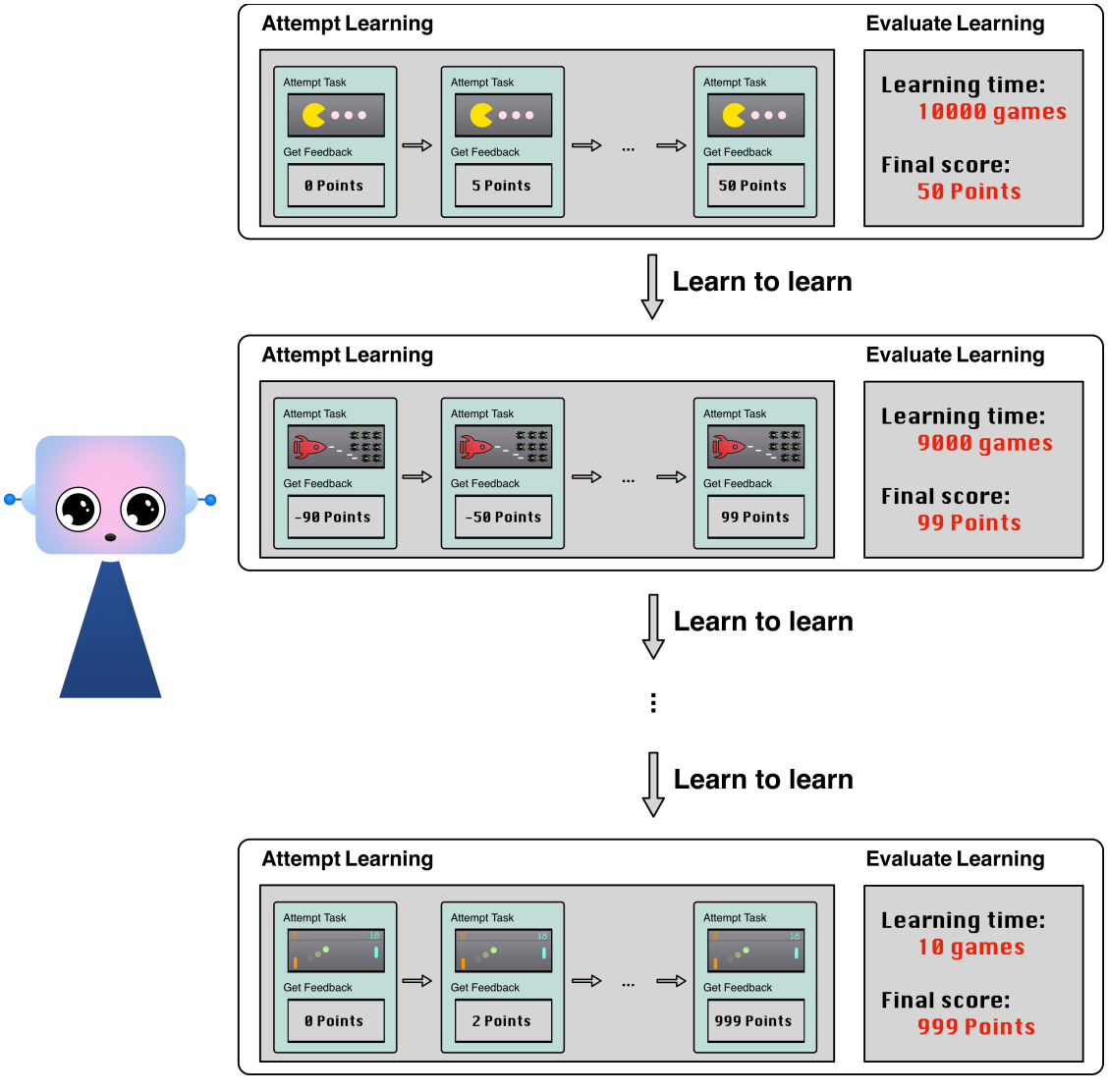


Figure 1.2: Simplified illustration of Meta-RL training. The agent repeatedly performs an entire learning procedure on different tasks, and learns from the provided feedback on how well it learned (horizontal axis). This "learning performance" can be measured, e.g., using the final performance or how long it took for the agent to master the game. Over time, the agent gets better at learning (vertical axis), until it can pick up entirely new games in a fraction of the time that a standard RL algorithm would take (see Figure 1.1). This learning behaviour is much closer to the efficiency with which humans learn video games like Ms PacMan.

1.3 Contributions

This thesis studies the Meta Reinforcement Learning for Fast Adaptation problem setting. We propose several Meta-Learning algorithms that allow us to meta-learn agents that can adapt to new tasks faster compared to existing methods across a wide range of tasks and settings. We hope that this helps drive forward Meta-RL research and, in the long term, using RL to address important real world challenges.

Meta-RL for Fast Adaptation Many of the contributions in this thesis build on the intuition that "picking up new tasks" can be split into the following two steps.

1. **Task Inference:** Understand the given task (e.g., in Ms PacMan this means understanding the game dynamics like how the controls work, how the ghosts move, and that coins give positive rewards).

2. **Task Solving:** Learn how to solve the task (e.g., in Ms PacMan this means coming up with a strategy to win the game which should include walking strategically to collect coins, while avoiding ghosts and walls).

We posit that in many settings, with enough meta-training on sufficiently many similar tasks, the agent will be proficient at step (2): it has acquired enough skills and understanding of the world in order to know how to solve tasks. Therefore, all that is left to do when adapting to new tasks, is to infer *what* the task is (1).

Take the example of a house cleaning robot that has been trained on a variety of houses. It will already know how to mop a floor and clean a window, but when entering a new house, it has to learn about that house's layout, size, and properties. Another example is a poker-playing agent: when trained on sufficiently many and diverse matches, the agent knows what strategies work against which opponents. When deployed, it just has to figure out what the other players' strategies are.

Most existing Meta-RL methods do not explicitly separate task inference from task solving, but take a *black-box* approach where the agent implicitly does both. We propose to make this split explicit, and use it to efficiently learn new tasks.

Thesis Structure This thesis studies ways to split task inference and task solving through the design of the model architecture (by representing tasks as a *context vector* that the rest of the model conditions on), and the training procedure (by specialising each model part using different objectives). The thesis consists of three main parts.

Part I: Foundations The first part provides the background that is relevant for the work in this thesis.

- **Chapter 2** summarises the relevant RL background.
- **Chapter 3** introduces and motivates in detail the Meta-RL for Fast Adaptation problem setting. We discuss how efficient learning depends on the *adaptation horizon* (how much time the agent has to adapt to a task before we evaluate it), and use this to formalise two learning objectives.
- **Chapter 4** then presents the two main existing approaches to Fast Adaptation, which we build on and compare to throughout the thesis: adapting with gradients, and adapting with recurrent neural networks.

Part II: Methods In the main part of this thesis, we introduce four methods for solving the Fast Adaptation problem setting.

- **Chapter 5** starts out by looking at *Supervised Learning*, which allows us to sidestep some of the additional challenges of RL, like exploration or long-term credit assignment. We derive a gradient-based method, where the model is comprised of a main part (that learns to solve the task) and an input context vector (representing the task). Empirically we find that this leads to improved performance, and opportunity for model introspection. To conclude the chapter, we apply the method to the *Few-Shot Adaptation* Reinforcement Learning setting, where the agent has a fixed budget of environment interactions to learn before being evaluated. This work was published at ICML 2019 (Zintgraf et al., 2019).

- **Chapter 6** considers the more challenging *Online Adaptation* setting, where the agent has to perform well from the very first timestep when it attempts a task. Since all rewards during learning count towards its performance, the agent has to carefully balance exploration (gathering new information to infer the task) and exploitation (using current knowledge to perform well). There is a sweet spot when trading off these aspects, at which the agent learns optimally. To achieve this, we introduce Bayesian reasoning over the context vector. This allows us to compute approximately Bayes-optimal behaviour for domains where this was previously not computationally feasible. This work was published at ICLR 2020 (Zintgraf et al., 2020), and as an extended version in JMLR (Zintgraf et al., 2021c).

- **Chapter 7** looks at the multi-agent setting where the agent does not adapt to changing environments or tasks, but rather to changing other agents that it interacts with. Here, the agent's own actions influence the other agents' future actions. Modelling this introduces additional challenges, such as tracking and reasoning over the current states of mind of other agents. This work was published as an extended abstract at AAMAS 2021 (Zintgraf et al., 2021a).

- **Chapter 8** focuses on the additional challenge of having very sparse rewards. Since many real world tasks are best described using sparse rewards, this is an important problem setting. Solving these problems requires sufficient exploration *during* meta-training, on a meta-level: the agent must collect data from which it can meta-learn successfully. We show that most existing Meta-RL methods fail when rewards are sparse, and propose a way to overcome this. This work was published at ICLR 2021 (Zintgraf et al., 2021b).

Part III: Discussion We conclude the thesis with a reflection on the work presented in the context of current developments, and a discussion of open questions. This includes results previously published as Xiong et al. (2021) at the Workshop on Meta-Learning at NeurIPS 2021.

Part I

Foundations

2

Reinforcement Learning Background

Contents

This chapter introduces the relevant background from Reinforcement Learning and Bayesian RL. Where necessary, we provide more background in the respective future chapters of the thesis.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a way to solve sequential decision making problems, where an agent repeatedly interacts with an environment by taking actions, and receiving observations and rewards in return. We summarise the concepts relevant to the thesis here, and refer to Sutton et al. (2018) for a detailed introduction.

2.1.1 Markov Decision Processes

Most commonly, the RL problem setting is formalised as a Markov Decision Process (MDP) (Bellman, 1966), expressed as a tuple $M = (\mathcal{S}, \mathcal{A}, R, T_0, T, \gamma, H)$, where we call $\mathcal{S}$ a set of states, $\mathcal{A}$ a set of actions, R the reward function, T_0 the initial state distribution, T the transition function, γ a discount factor, and H the horizon.

Environment Dynamics An agent interacts with an MDP by starting out in an initial state $s_0 \in \mathcal{S}$ according to the initial state distribution $T_0(s{=}s_0) : \mathcal{S} \to [0, 1]$.[1] In a sequence of discrete timesteps t up to a horizon $H \in \mathbb{N}_{>0} \cup \infty$, the agent takes actions $a_t \in \mathcal{A}$, and transitions according to a probabilistic transition function $T(s = s_{t+1}|s_t, a_t) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$.[2] This transition function is *Markov*, i.e., the next state s_{t+1} only depends on the previous state s_t and action a_t – it does *not* depend on states $s_{<t}$ or actions $a_{<t}$ that lie further in the past. After each transition, the agent receives a reward according to the reward function $R(r_{t+1}|s_t, a_t, s_{t+1}) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. Rewards are Markov as well. Throughout the thesis, we consider finite MDPs with $H \in \mathbb{N}_{>0}$ and denote full trajectories as $\tau = (s_0, a_0, r_1, \ldots, s_H)$, also called a *rollout* or *episode*. Partial trajectories of length t are denoted $\tau_{:t} = (s_0, a_0, r_1 \ldots, s_t)$.

Maximising Returns The expected *return* $R(\tau)$ of an episode is defined as the expected discounted sum of rewards, $\mathbb{E}_{p(\tau)}[R(\tau)] \equiv \mathbb{E}_{p(\tau)}[\sum_{t=0}^{H-1} \gamma^t R(r_{t+1}|s_t, a_t, s_{t+1})]$. The discount factor $\gamma \in [0, 1]$ determines how much the agent favours immediate rewards over delayed ones. The goal in RL is to learn a probabilistic policy $\pi(a_t|s_t) : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that maximises the expected return

$$\mathcal{J}(\pi) = \mathbb{E}_{p(\tau)}\left[R(\tau)\right]. \tag{2.1}$$

Because the transitions and rewards are Markov, the policy can be Markov as well, and condition its actions only on the current state.

[1] For ease of exposition, we assume that the sets $\mathcal{S}$ and $\mathcal{A}$ are discrete. An extension to continuous state and action spaces can be achieved by replacing the probabilities with corresponding densities. We encounter both discrete and continuous state and action spaces in this thesis.

[2] Henceforth $T(s{=}s_{t+1}|s_t, a_t)$ is denoted $T(s_{t+1}|s_t, a_t)$ and includes T_0 for brevity.

We generally do not know the transition function T or the reward function R upfront. Hence, the agent has to learn via trial and error through environment interactions.

2.1.2 Solving MDPs

One way to learn a policy $\pi(a_t|s_t)$ that maximises expected return is using *policy gradient* algorithms, which often make additional use of *value functions*. We use two such algorithms in this thesis, and therefore introduce the basic concepts of policy gradients and value functions below.

Policy Gradients To learn a policy π, we can parameterise it with a parameter θ, and follow the policy gradient $\nabla_\theta J(\pi_\theta)$ to find the θ that produces the highest expected return. Using the policy gradient theorem (Sutton et al., 2000), the policy gradient can be computed as

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{T,\pi}\left[\nabla_\theta \log \pi_\theta(\tau) R(\tau)\right]. \tag{2.2}$$

A simple approach to estimate this objective is the REINFORCE algorithm (Williams, 1992), which uses a single Monte Carlo estimate of Equation (2.2) by rolling out the policy once. This estimate suffers from high variance, and many variants to estimate the above objective have been proposed since. Many of these make use of so-called value functions, which we introduce next.

Value Functions The value of a given state and policy is defined as

$$V^\pi(s) = \mathbb{E}_{p(\tau)}\left[R(\tau)|s_0 = s\right],$$

i.e., the future expected return when following the policy π from a given state s_t. We further define the value of a state-action pair as

$$Q^\pi(s,a) = \mathbb{E}_{p(\tau)}\left[R(\tau)|s_0 = s, a_0 = a\right],$$

i.e., the expected return when starting in state s, taking action a, and following the policy π thereafter.

Value functions define an ordering over policies: $\pi \geq \pi'$ if $V^{\pi}(s) \geq V^{\pi}(s)\ \forall s \in \mathcal{S}$. There always exists at least one optimal policy π^* (Puterman, 2014), which is defined as a policy whose value is better or equal than that of any other policy across all states, $\forall s \in \mathcal{S}, \pi \in \Pi : V^{\pi^*}(s) \geq V^{\pi}(s)$. Here, Π denotes the set of all possible policies. There can be multiple optimal policies, but they all share the same optimal value function; $V^*(s) := \max_{\pi} V^{\pi}(s)$ for the state value function, and $Q^*(s,a) := \max_{\pi} Q^{\pi}(s,a)$ for the state-action value function.

A useful property of value functions is that they satisfy a recursive relationship known as the Bellman equations (Bellman, 1957),

$$V^{\pi}(s_t) = \mathbb{E}_{\pi,T}\left[r_t + \gamma V^{\pi}(s_{t+1})\right] \quad \text{and} \tag{2.3}$$

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi,T}\left[r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1})\right]. \tag{2.4}$$

These are the basis for many algorithms that learn value functions. Further, the Bellman equations allow us to estimate future returns as the sum of immediate rewards (r_t) and the expected future discounted returns *bootstrapped* from the value functions ($\gamma V^{\pi}(s_{t+1})$ or $\gamma Q^{\pi}(s_{t+1}, a_{t+1})$). This has lower variance than using Monte Carlo rollouts, but is biased towards our current (possibly wrong) value estimates.

Learning Value Functions Learning value functions can be useful in many ways. For example, if we learn the *optimal* state-action value function, we can derive an optimal policy by choosing actions with maximal values. Alternatively, when learning policies using policy gradients, we can use value functions for bootstrapping returns or as a baseline (Williams, 1992) to reduce variance.

To learn a policy's state value function, we often parameterise it with a parameter ω and write V_{ω}. Learning can be done via stochastic gradient descent on the squared error between our current value estimate and a target estimate from samples,

$$\mathcal{L}^{VF}(V_{\omega}) = \mathbb{E}_{T,\pi}\left[(V_{\omega}(s_t) - V_t^{target})^2\right]. \tag{2.5}$$

The target value V_t^{target} can be estimated using Monte Carlo rollouts, or using a mix of environment samples and bootstrapped values as in Equation (2.3). In the same way, the state-action value function Q can be learned.

2.1.3 Deep Learning Primer

Most interesting RL problems have large or continuous state and action spaces, which often means that we have to resort to function approximation to represent policies or value functions. In Deep Reinforcement Learning (Deep RL), we use deep neural networks to parameterise the policy π_θ and value function V_ω or Q_ω, i.e., θ and ω denote the weights of deep neural networks. These are usually trained using stochastic gradient ascent or descent. We refer the reader to Goodfellow et al. (2016) for an introduction to Deep Learning, and briefly introduce three building blocks from Deep Learning that are relevant to this thesis here: feedforward neural networks, recurrent neural networks, and variational auto-encoders.

Feedforward Neural Networks One way to represent a function (e.g., policy or value function) is using a fully connected feedforward neural network. Such a network consists of several fully connected layers, each of which can be described as

$$z = g(Wx + b), \tag{2.6}$$

where $x \in \mathbb{R}^{d_x}$ is an input vector, $W \in \mathbb{R}^{d_z \times d_x}$ is the weight matrix, $b \in \mathbb{R}^{d_z}$ a bias vector, g a non-linear activation function, and $z \in \mathbb{R}^{d_z}$ the output vector. For a policy $\pi_\theta(a|s)$, this neural network would take the state s as an input to the first layer, and output an action a from the last layer (or a probability distributions over actions, e.g., represented by a mean and variance).

If the states s have a specific structure, we might be able to take advantage of this by using other network architectures. For images for example, we can use *convolutional neural networks*, which learn feature extractors specific to visual inputs (see Goodfellow et al. (2014) for an introduction). Important for this thesis are recurrent neural networks (RNNs), which can be used to process sequential data such as as natural language, music, or trajectories of agent-environment interactions τ. We introduce these next.

Recurrent Neural Networks Recurrent neural networks (RNNs) are deep neural networks with a form of memory. RNNs have at least one recurrent layer, that has – as the name suggests – some form of recurrence. There are different mechanisms to implement this, with the most prominent ones being the Long Short-Term Memory (LSTM, Hochreiter et al., 1997) or Gated Recurrent Unit (GRU, Cho et al., 2014). We outline the mathematical formulation of GRUs here, since we use these in our experiments. A GRU layer is initialised with a hidden vector $h_0 = \mathbf{0} = (0, \dots, 0) \in \mathbb{R}^{d_h}$. For a sequence of layer inputs $(x_1, \dots, x_t)$, at timestep $t \geq 1$, the GRU computes an output vector h_t using

$$z_t = g_\sigma(W_z x_t + U_z h_{t-1} + b_z), \tag{2.7}$$

$$r_t = g_\sigma(W_r x_t + U_r h_{t-1} + b_r), \tag{2.8}$$

$$\hat{h}_t = g_{tanh}(W_h + U_h(r_t \odot h_{t-1}) + b_h), \tag{2.9}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t, \tag{2.10}$$

where g_σ is the sigmoid activation function, g_{tanh} the hyperbolic tangent activation function, $\hat{h}$ a candidate activation vector, z_t the update gate vector, r_t the reset gate vector, and h_t the output vector which is used in the next update. In this thesis, we use RNNs to summarise trajectories $\tau_{:t}$ of agent-environment interactions.

Variational Auto-Encoders An important building block for this thesis are latent variable models (Bishop et al., 2006), optimised using the reparametrisation trick which was introduced in the context of the variational auto-encoder (VAEs; Kingma et al., 2014). We use these models in Chapters 6-8 to quantify an agent's uncertainty over what task it is meant to be performing. This uncertainty is then taken into account when taking actions. We briefly summarise the standard VAE setup here, and refer the reader to Kingma et al. (2014) for a thorough introduction. VAEs consist of an encoder (a neural network q_ϕ), a decoder (a neural network p_ψ), and a loss function. In the standard case presented in Kingma et al. (2014), the encoder takes as input a datapoint $x \in \mathbb{R}^{d_x}$ (e.g., an image of a cat) and produces a

distribution over a latent representation $m \in \mathbb{R}^{d_m}$ (a vector representing a low-level representation of the input with $d_m \ll d_x$). Usually this is implemented by the encoder producing a mean and a variance vector, $\mu_m \in \mathbb{R}^{d_m}$ and $\sigma_m \in \mathbb{R}^{d_m}$, of a diagonal Gaussian distribution $\mathcal{N}(m|\mu_z, \sigma_z\mathbb{I})$, and we write $q_\phi(m|x)$. The decoder network, p_ψ, takes as input a latent representation $m \in \mathbb{R}^{d_m}$ (sampled from the probability distribution we just described), and produces an output x. This is also called a *reconstruction*, and we generally want this to be as similar as possible to the encoder input, which can be measured with the log-likelihood $\log p_\psi(x|m)$. To train q_ϕ and p_ψ, the VAE maximises the so-called Evidence Lower Bound (ELBO):

$$\mathcal{L}(\phi, \psi) = \mathbb{E}_{q_\phi(m|x)} \left[\log p_\psi(x|m)\right] - KL(q_\phi(m|x)||p(m)). \tag{2.11}$$

The first term is the log-likelihood, that measures how well we reconstructed the input (e.g., for images we can use negative pixel-wise mean squared errors). The second term is the Kullback-Leibler divergence (KL divergence; Kullback, 1997), which is a way to measure how similar two probability distributions are, defined as $KL(q_\phi(m|x)||p(m)) = \int_m q_\phi(m|x) \log \frac{q_\phi(m|x)}{p(m)}$. The distribution $p(x)$ is called a *prior*, and often set to $\mathcal{N}(\mathbf{0}, \mathbb{I})$. In the above equation, it acts as a regulariser.

When using VAEs in the context of this thesis, we consider the case where m is a latent *task* description (e.g., representing a goal position; see Figure 3.1a), and where p_ψ is a distribution over MDPs. We do not use the standard VAE setup like here, but a sequential version (Fabius et al., 2015; Chung et al., 2015) adapted to our problem setting, where datapoints x are agent trajectories $\tau_{:t}$ of varying length t. We get back to these and derive the ELBO terms for our proposed VAE architecture in Chapters 6 and Chapter 7.

2.1.4 Deep Reinforcement Learning

Two Deep RL algorithms that we use in this thesis are Trust Region Policy Optimisation (TRPO; used in Chapter 5; Schulman et al., 2015a) and Proximal Policy Optimisation (PPO; used in Chapters 6-8; Schulman et al., 2017). We introduce these here, and refer to Arulkumaran et al. (2017) for a survey on Deep RL.

TRPO Trust-Region Policy Optimisation (Schulman et al., 2015a) maximises the objective in Equation (2.1) subject to a "trust region constraint", which enforces that the distance between the old and the updated policy is small. This distance is estimated using the KL divergence (Kullback, 1997) between the old and the new policy, $KL(\pi_\theta(.|s)||\pi_{\theta_k}(.|s)) = \sum_{a\in\mathcal{A}} \pi_\theta(a|s) \log \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$. The theoretical TRPO update is

$$\theta_{k+1} = \arg\max_\theta \mathcal{L}(\theta_k, \theta) \tag{2.12}$$

$$\text{s.t. } \mathbb{E}_{s\in\pi_{\theta_k}} \left[KL(\pi_\theta(.|s)||\pi_{\theta_k}(.|s))\right] \leq \delta. \tag{2.13}$$

Here, k is the number of times we have updated θ so far, and δ is a threshold for the KL term. The idea is that this restriction avoids updates that change the policy too much at one step, thereby reducing instability issues. Since the above constrained optimisation problem is difficult to work with, Schulman et al. (2015a) propose an approximate objective function. For details see the original paper.

PPO Proximal Policy Optimisation builds on similar intuition, but uses a clipped surrogate objective which is easier to estimate and retains similar performance empirically (Schulman et al., 2017). The overall objective is given by

$$\mathcal{L}^{PPO}(\theta, \omega) = \mathbb{E}_{T,\pi}\left[L^{CLIP}(\theta) - \lambda_1 L^{VF}(\omega) + \lambda_2 S(\pi_\theta)(s_t)\right]. \tag{2.14}$$

Here, L^{VF} is the value function loss from Equation (2.5), $S(\pi_\theta)$ is an entropy term that encourages exploration, and λ_1 and λ_2 weigh off the different terms relative to each other. The term $\mathcal{L}^{CLIP}$ is a surrogate objective for the policy gradient, given by

$$\mathcal{L}^{CLIP}(\theta) = \mathbb{E}_{\pi,T}\left[\min(r_t A_t, \text{clip}(r_t, 1-\varepsilon, 1+\varepsilon)A_t)\right], \tag{2.15}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is a probability ratio between the new and old policy, $A_t = Q_t - V_t$ is the advantage function at timestep t, and ε is a hyperparameter that influences how far the new policy can deviate from the old one. The expectations are estimated per-timestep, using multiple trajectories collected with the current policy. For more details we refer the reader to the original paper, or the source code accompanying the work in this thesis.

2.1.5 Partially Observable MDPs

In MDPs, transitions are Markov: the next state only depends on the previous state and action, and the MDP can be solved by a policy $\pi(a_t|s_t)$ that conditions only on the current state s_t. The Markov assumption holds, e.g., for games like Ms PacMan. For many real world problems however, this is an unrealistic assumption. In poker for example, a player only sees their own hand and the cards on the table, but playing well requires remembering played cards to reason about other players' hands. In this case we say that the agent has access only to observations o_t, rather than the full environment state s_t. Partial observability also occurs when an agent encounters new situations that it still has to learn about. It is therefore an important concept for this thesis.

When the Markov assumption does not hold, we can formulate the problem as a Partially Observable Markov Decision Process (POMDP) (Astrom, 1965; Kaelbling et al., 1998). The optimal policy then has to either condition on the observed history $\hat{\tau}_{:t} = (o_0, a_0, r_1, \ldots, o_t)$, such that $\pi(a_t|\hat{\tau}_{:t})$, or on its belief over the true state $b_t \equiv p(s = s_t|\hat{\tau}_{:t})$, such that $\pi(a_t|b_t)$. The belief here is defined as the posterior distribution over the current state given the history. If beliefs are available, the POMDP can be reformulated into a Belief MDP (Cassandra et al., 1994), which is the MDP formed by taking the posterior beliefs maintained by an agent in a POMDP and reinterpreting them as Markov states. In practice, past trajectories can be summarised using RNNs (Hausknecht et al., 2015; Zhu et al., 2017), by remembering features of the past (McCallum, 1993), or by doing inference over the possible latent states (Kaelbling et al., 1998; Igl et al., 2018).

While POMDPs are general and theoretically well justified, computing solutions is often intractable and it is difficult to make general progress on them. A tangible way forward is often to make use of additional structure, leading to a richer yet feasible setting compared to standard MDPs. This is what Bayes-Adaptive Markov Decision Processes (BAMDPs) do, where the environment state is fully observable, but the task is unknown to the agent. BAMDPs are central to the Meta-RL methods developed in the thesis, and are introduced in Section 2.2.

2.1.6 The Exploration-Exploitation Trade-Off

The efficiency of learning in an RL setting often depends on how well the agent trades off exploration and exploitation, and is therefore an important aspect in the context of Fast Adaptation. We briefly discuss this trade-off here, and come back to it throughout the thesis.

In Reinforcement Learning – in contrast to Supervised Learning – the agent itself collects the data that it learns from. Training often alternates between a phase of data collection and updating the policy or value function, until a stop criterion is reached. When interacting with the environment, the agent therefore has to choose how much to explore, and how much to exploit, given everything it has learned so far. *Exploration* refers to taking information-seeking actions, that help the agent gather data from which it can learn about the environment, and the short and long-term consequences of its actions. Exploratory actions can be costly and therefore sub-optimal in the short term, but may pay off in the long term, since the agent learns more about its environment. *Exploitation* refers to acting greedily given the agent's *current* knowledge about which actions have the highest expected return.

Balancing exploration and exploitation carefully during learning is key to learning efficiently. Early in training, the agent does not know much about its environment, and should therefore explore more. Exploration can happen in many ways, such as adding random noise (Sutton, 1995; Lillicrap et al., 2016) or using state-visitation counts to explore under-visited states (Sutton, 1990; Bellemare et al., 2016). For a recent survey on exploration in RL see Amin et al. (2021) and Yang et al. (2021). As training progresses, the agent becomes more certain about which actions lead to high returns, and should exploit that knowledge and explore less. If the agent has learned the optimal value function or policy, it does not need to explore, but can instead simply act greedily.

In Chapter 6-8 of this thesis we use task uncertainty to drive exploration, so that the agent can learn new tasks efficiently. To this end, we build on concepts from Bayesian Reinforcement Learning, which we introduce in the next section. We come back to exploration in the context of Meta-RL in Chapter 3.

2.2 Bayesian Reinforcement Learning

Bayesian Reinforcement Learning approaches quantify uncertainty (e.g., about states, rewards, or tasks) to support action-selection. Bayesian RL is central to this thesis, and we use it in Chapters 6-8 as a way for the agent to incorporate uncertainty about the task when selecting actions. We refer the reader to Neal (2012) for a introduction in Bayesian Learning, and Ghavamzadeh et al. (2015) for a review on Bayesian RL, and focus this section on one of the central concepts from Bayesian RL that this thesis builds on: Bayes-adaptive policies. Bayes-adaptive policies are interesting because they optimally trade off exploration and exploitation during learning.

2.2.1 Bayes-Adaptive MDPs

When the MDP is unknown, optimal decision making has to trade off exploration and exploitation when selecting actions (see Section 2.1.6. In principle, this can be done by taking a Bayesian approach to RL formalised as a Bayes-Adaptive MDP (BAMDP), the solution to which is a Bayes-optimal policy (Bellman, 1956; Duff et al., 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 $b_0 = p(R, T)$. The agent does not have access to the true reward and transition function, but can maintain a belief

$$b_t(R, T) = p(R, T|\tau_{:t}),$$

which is the posterior over the MDP given the agent's experience $\tau_{:t}$ up until the current timestep t.

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

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

i.e., the new environment state s_t is the expected new state with respect to the current posterior distribution of the transition function, and the belief is updated deterministically according to Bayes rule. The reward function on hyper-states is defined as the expected reward under the current posterior,

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

This results in a BAMDP $M^+ = (\mathcal{S}^+, \mathcal{A}, R^+, T^+, T_0^+, \gamma, H^+)$ (Duff et al., 2002), which is a special case of a belief MDP (Cassandra et al., 1994, see Section 2.1.5). In an arbitrary belief MDP, the belief is over a hidden state that can change over time. 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,

$$\mathcal{J}^+(\pi) = \mathbb{E}_{b_0, T_0^+, T^+, \pi}\left[\sum_{t=0}^{H^+-1} \gamma^t R^+(r_{t+1}|s_t^+, a_t, s_{t+1}^+)\right], \tag{2.16}$$

i.e., maximise the expected return in an initially unknown environment, while learning, within the horizon H^+. How to trade off exploration and exploitation optimally depends on how much time the agent has left (e.g., to decide whether information-seeking actions are worth it). The BAMDP horizon H^+ can be distinct from the MDP horizon H, e.g., if we want the agent to act Bayes-optimal within the first N MDP episodes, so $H^+ = N \times H$.

While a Bayes-optimal policy can in principle be computed using the BAMDP framework, this is unfortunately hopelessly intractable for all but the smallest tasks. Existing methods are therefore restricted to small and discrete state and action spaces (Asmuth et al., 2011; Guez et al., 2012, 2013), or a discrete set of tasks (Brunskill, 2012; Poupart et al., 2006).

2.2.2 Comparison to Other Exploration Strategies

The Bayes-optimal agents we introduced in the previous section optimally trade off exploration and exploitation when learning a new task. Computing them however is often intractable, and in practice, other exploration strategies may be used.

To illustrate how Bayes-optimal exploration works, we compare it to two other strategies – posterior sampling and random exploration – using an example.

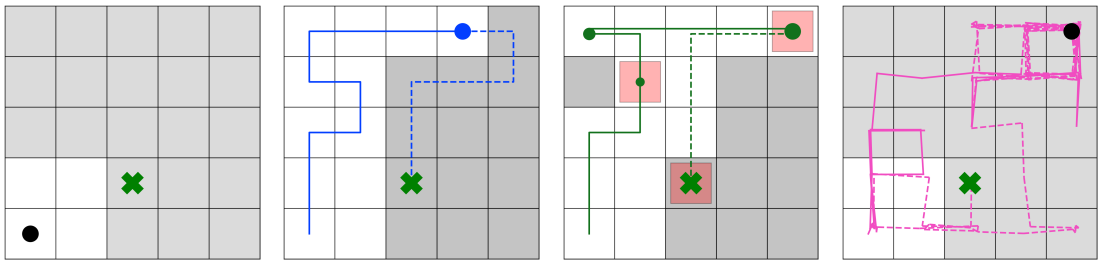
Posterior Sampling A common shortcut to Bayes-optimal exploration is to rely instead on *posterior sampling* (Thompson, 1933; Strens, 2000; Osband et al., 2013). Posterior sampling, which extends Thompson sampling (Thompson, 1933) from bandits to MDPs, estimates a posterior distribution over MDPs, in the same spirit as in a BAMDP. The difference lies in how this belief is used to select actions: the 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. 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. It might incur large penalties, especially early in learning, and therefore typically has lower expected online return.

Random Exploration Posterior sampling still requires the agent to maintain a belief over the tasks, which is intractable for complex domains. A much simpler strategy is to do random exploration, such as ε-greedy (where the agent takes an action sampled uniformly at random with probability $\varepsilon > 0$, and the greedy action otherwise; Sutton, 1995) or adding noise to the predicted actions. Random exploration can be very inefficient, but is often used in practice in Deep RL due to its simplicity.

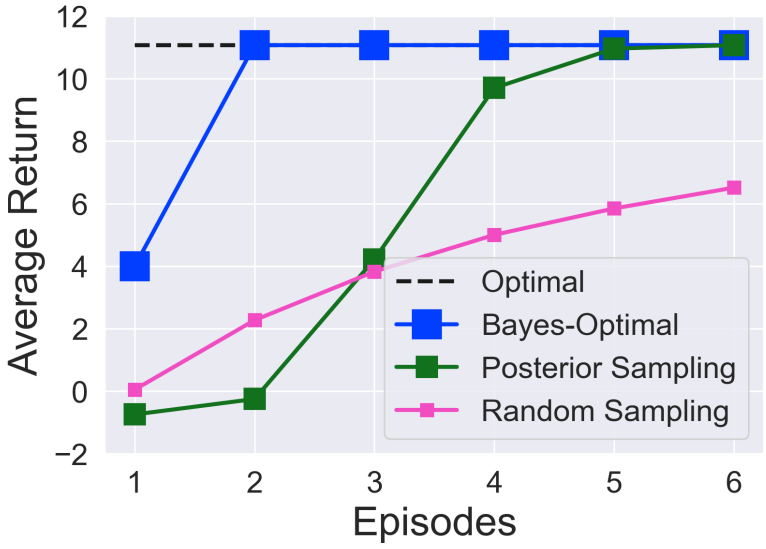
Many other exploration strategies that are more informed than random exploration, and more feasible to compute than Bayes-optimal exploration or posterior sampling, exist in the literature (Amin et al., 2021; Yang et al., 2021). These are mostly developed for the standard RL setting, where an agent learns over up to millions of environment interactions. Since we consider the Fast Adaptation problem setting in this thesis, we therefore only compare Bayes-optimal, posterior sampling, and random exploration in the following example.

Example Consider the example of the Gridworld environment in Figure 2.1, where the agent must navigate to an *unknown* goal located in the grey area (Fig 2.1a). 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 searches the set of goal positions that the posterior considers possible, until the goal is found (Fig 2.1b). Posterior sampling samples a possible goal position, takes the shortest route there, and then resamples a new goal from the updated posterior (Fig 2.1c). Random exploration displays jittery behaviour, re-visiting states multiple times, until it eventually reaches the goal (Fig 2.1d). When comparing these strategies quantitatively by the expected return they achieve in each episode (Fig 2.1e), we see that the Bayes-optimal agent performs best and matches optimal behaviour from the second episode. We also see that posterior sampling is much less efficient, which is because the agent's uncertainty is not reduced optimally (e.g., states are revisited unnecessarily). Random exploration performs poorly, as expected.

Most existing methods for computing Bayes-optimal agents are restricted to small state and action spaces like this Gridworld. A key challenge is to learn approximately Bayes-optimal policies in a tractable way. In Chapter 6 we return to this Gridworld example, and propose a Meta-RL method that closely matches the ground-truth Bayes-optimal behaviour in this Gridworld, and scales to more complex environments.



(a) Environment **(b)** Bayes-Optimal **(c)** Posterior Sampling **(d)** Random Sampling



(e) Performance

Figure 2.1: Illustration of different exploration strategies. **(a)** Environment: The agent starts at the bottom left and has to navigate to an unknown goal, located in the grey area. **(b)** A Bayes-optimal agent systematically searches possible grid cells to find the goal, shown in solid (past actions) and dashed (future actions) blue lines. A simplified posterior is shown in the background in grey ($p = 1/$(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) from the current posterior, takes the shortest route to the sampled goal, and then updates its posterior. **(e)** Random sampling, which chooses from all available actions uniform at random. **(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, see dashed line). The Bayes-optimal agent matches this behaviour from the second episode, whereas posterior sampling needs six rollouts. Random exploration (with optimal behaviour after the goal has been found) takes much longer to find the goal.

3
Fast Adaptation via Meta-RL

Contents

In the introduction we discussed how standard RL can be slow, learns from scratch using random behaviour policies, and overspecialises in the limit, rendering it impractical for many settings. To overcome this, we need RL agents that are flexible, learn autonomously, and adapt quickly to new circumstances.

A promising approach to this is *Meta Reinforcement Learning* (Meta-RL), which we briefly introduced in Section 1.2. In Meta-RL, instead of developing all parts of RL algorithms ourselves, we automate some choices by *learning* them, which can be an effective way of adding domain knowledge to the learning algorithm. We can do so for any aspect of RL, such as meta-learning update rules (Oh et al., 2020; Bechtle et al., 2020; Kirsch et al., 2022), reward functions (Zheng et al., 2018; Feng et al., 2019; Zheng et al., 2020; Alet et al., 2020), or how to select hyperparameters on-the-fly (Jaderberg et al., 2017; Xu et al., 2018b; Paul et al., 2019; Zahavy et al., 2020).

In this thesis, we are interested in the "Fast Adaptation" setting: we want to meta-learn *agents that can adapt quickly* to new tasks. Such an agent should be able to adapt efficiently to different (albeit similar) tasks, within only a few environment interactions. Below we give three motivating examples.

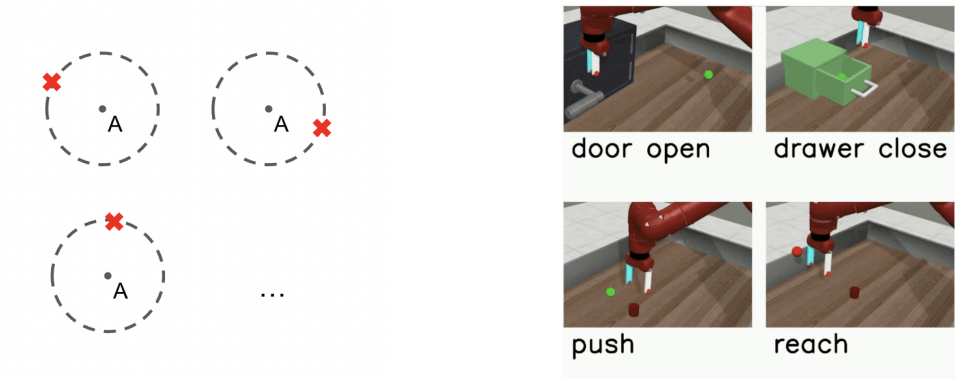
Robotics Robots are used across many economic sectors, but often a lot of work goes into specialising them for their individual use cases, like automated recycling (Chin et al., 2019), painting (Rola, 2007), or farming (Bakker et al., 2006). It would be more time and cost-efficient if we develop more general-purpose robots, that require only a small training session to specialise.

Video Game Bots Video game experiences are frequently enhanced by in-game AI agents that interact, compete, or collaborate with human players (Yannakakis et al., 2018). Ideally, we want those agents to adapt on-the-fly: to team up with a human, or to adapt to an individual player's ability in a competitive setting.

Assistive AI Technology Assistive AI is already used in products like tutoring systems (Iglesias et al., 2009; Roijers et al., 2012), recommender systems (Afsar et al., 2021; Portugal et al., 2018), or assistive technology for the blind (Lee et al., 2019b; Massiceti et al., 2021). In the future, these can become more personalised by learning about and incorporating an individual's preferences.

In the following sections, we introduce the ingredients needed for meta-learning how to adapt fast:

- **A task distribution** for meta-training. This represents the types of tasks we want our agent to adapt to when deploying it at meta-test time.
- **An objective function** that captures the desired behaviour of the agent, defined by how much time it has to learn a new task before being evaluated.
- **A meta-training procedure** to learn how to learn tasks from the given task distribution as efficiently as possible.



(a) 2D Navigation **(b)** Meta-World

Figure 3.1: Task distribution examples. (a): In this 2D navigation example, the agent A has to learn to navigate to a goal x. The goal location is unknown to the agent, placed on a unit circle around the agent's start position. **(b)**: Example tasks from the Meta-World benchmark (Yu et al., 2019), which consist of 50 different types of tasks, each of which itself is a distribution over different instances of the type of task.

3.1 Task Distribution

The task distribution provides the training environments for the agent, and defines the types of tasks for which we want fast-adapting agents. Meta-learning allows us to then translate this into an efficient learning algorithm, tailored to that task distribution. The task distribution therefore is the main way of providing domain knowledge about what types of tasks the agent can expect, and how to efficiently learn them. Formally, the task distribution is given as a probability distribution $p(M)$ over MDPs (see Section 2.1). Across tasks, the reward and transition functions can vary, but typically share some structure. Therefore we often express the task distribution $p(M)$ as $p(R, T)$, and sample an MDP from $p(M)$ by sampling a reward and transition function from $p(R, T)$. For a single sample $M_i \sim p(M)$, we use the index i to denote an unknown task description (e.g., a goal position or language instruction) or task ID. Two task distribution examples are shown in Figure 3.1.

Throughout this thesis, as is common in the Meta-RL literature, we assume access to unlimited samples from $p(M)$ for meta-training, and that the agent is evaluated on samples from that same distribution at meta-test time. Handling distribution shifts is a challenging open problem, as we discuss in Chapter 9.

3.1.1 Decoupling Task Inference from Task Solving

In this thesis, we argue the following (see also Section 1.3):

> In many cases, Fast Adaptation comes down to task inference. In this case, this can – and should – be decoupled from task solving.

We argue that this split is possible in many real world settings, and is reflected in existing Meta-RL benchmarks, as we establish in Chapters 5 and 6. Consider the example in Figure 3.1a, where fast adaptation consists of identifying the goal position (task inference), and then executing a policy that can repeatedly navigate there (task solving). The connection to the exploration-exploitation trade-off (Section 2.1.6) becomes apparent: the agent has to explore to infer the task, and then exploit this knowledge to solve the task. This can be meta-learned by interacting with samples from $p(M)$. Similarly, in our earlier example of assistive AI technology, adapting to different humans can be done by inferring the individual's personal preferences, and acting accordingly. This can be meta-learned as long as the meta-training distribution includes a representative set of human preferences.

This paradigm is central to the methods developed in this thesis, and stands in contrast to the majority of existing Meta-RL methods where the agent is a black-box that has to do both. This black-box approach is used by two of the most prominent algorithms, MAML (Finn et al., 2017a) and RL^2 (Duan et al., 2016; Wang et al., 2016), which we introduce in the next section and compare to throughout the thesis. We find empirically that separating task inference and task solving often leads to better performance and has useful properties like better interpretability (Chapters 5-7), and is amenable for downstream or auxiliary tasks (Chapter 8).

If the meta training and test distribution are not the same, the agent also has to not only infer the task, but also pick up new skills in order to solve it. We do not consider this setting in this thesis, but discuss it as part of future work in Chapter 9. We believe that *even in this case*, separating task inference and task solving could be beneficial for efficient adaptation: e.g., it may allow the agent to determine which part(s) it has to improve, and focus on them separately.

3.2 Objectives for Fast Adaptation

We want to meta-learn how an agent can adapt fast when deployed in an initially unknown task from $p(M)$. In order for the agent to get better *at learning* over time, we must define how to evaluate learning performance. To this end, we have to consider what behaviour we want from the agent at test time – more specifically, how much time we give it to learn the task, before evaluating it. We distinguish two scenarios:

1. **Few-Shot Adaptation:** The agent has a fixed number of timesteps to freely explore, and is evaluated afterwards. Only the rewards after this initial exploration phase count towards its performance.

2. **Online Adaptation:** The agent has to perform well from the time it first enters the environment. All rewards from the first timestep count towards its performance, including all exploratory actions.

The Online Adaptation setting is often preferable for real world applications, where we want agents to perform well from the moment they starts interacting with the world. It also generally leads to agents that learn more efficiently, and is the main focus of this thesis. It is, however, more challenging and we therefore consider the Few-Shot Adaptation setting as a first step in Chapter 5, before turning to the Online Adaptation setting in Chapters 6-8. We discuss the two objectives and problem settings in the following.

3.2.1 Few-Shot Adaptation

In the Few-Shot Adaptation setting, the agent has some time to freely explore and learn, before we want it to perform well. Take the example of a robot that is being deployed in a new factory. We might be able to train it under human supervision for a short time for its specific task, and only need it to perform well and autonomously afterwards.

The objective we want the agent to maximise in this case is

$$\max_{\pi} \mathbb{E}_{p(M)} \left[\mathbb{E}_{T,\pi} \left[\sum_{t=H^e}^{H^+-1} r_t \right] \right], \tag{3.1}$$

where $H^e < H^+$ is the time we give the agent to freely explore and learn. Essentially, the rewards during this initial phase are masked out by setting them to zero, $r_{<H^e}=0$, and the rewards only count for the timesteps $t = (H^e, \dots, H^+)$.[1] The rewards from the initial phase can still be used during learning, e.g., by doing gradient updates with the data from the first H^e steps, using standard RL objectives. We propose one such method in Chapter 5.

3.2.2 Online Adaptation

In the Online Adaptation setting, we want the agent to perform well from the moment it starts interacting with its environment. This is the case in many real world settings, where the agent could otherwise incur irreversible damage to itself or others. Formally, we want the agent to maximise the *online return* achieved *during task-learning*,

$$\max_{\pi} \mathbb{E}_{p(M)} \left[\mathbb{E}_{T,\pi} \left[\sum_{t=0}^{H^+-1} r_t \right] \right]. \tag{3.2}$$

In this equation, all rewards that the agent receives, from the very first timestep of *learning*, count. Any exploratory actions are included in the expected online return, and should only be taken if they ultimately lead to high returns, in expectation, within the horizon H^+.

Maximising learning performance from the first timestep that the agent interacts with its environment as in the equation above is exactly the objective in Bayes-Adaptive MDPs that we introduced in Section 2.2.1. Optimal learning here can therefore be formulated as learning in a BAMDP, and maximising the BAMDP objective from Equation (2.16). We make use of this when developing Meta-RL algorithms for Online Adaptation in Chapters 6 and 7.

[1] From now, we drop the discount factor in the objective function, since we only consider small and finite horizons H^+ and therefore usually set $\gamma = 1$.

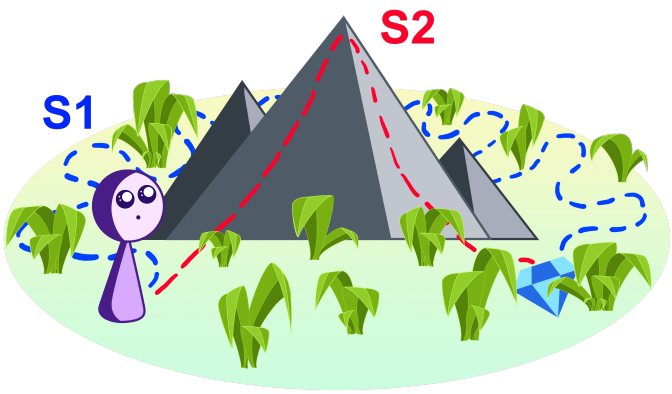


Figure 3.2: Illustration of the Online Adaptation setting. In this environment, the agent must find a hidden treasure in high grass. Two possible strategies are to (s1) search the grass for the treasure, or (s2) climb up the mountain, see the treasure, and go there. The latter strategy is costly in the short term, since climbing the mountain incurs a large penalty, but pays off since the agent reaches the treasure faster. Strategy s2 therefore maximises the expected online return from Eq (3.2). If the agent was instead maximising the few-shot adaptation Eq (3.1), and the training phase horizon H^e was long enough to search the entire grass, it could learn to follow either strategy.

Since the concept of Online Adaptation and its connection to the exploration-exploitation trade-off is central to this thesis, we illustrate this using an example, shown in Figure 3.2. In this environment, the agent must find a treasure that is hidden in high grass, for which it receives a high reward. The distribution of tasks is induced by different treasure locations. Whenever the agent is set into this environment, the treasure is at a new location and the agent therefore needs to explore. For every step in the environment, the agent incurs a small penalty. Climbing the nearby mountain incurs a large penalty per timestep.

One possible exploration strategy (denoted s1) is to search the grass until finding the treasure. This however takes a long time, and the expected return is relatively low at around 100, due to the accumulated penalties from the long search. A strategy with a higher expected return (denoted s2) is to climb the mountain, from where the agent can see the treasure, and then exploit that knowledge by going the treasure directly. The cost of exploration is higher in the short term, since going up the mountain incurs a large penalty, but pays off due to reaching to the treasure faster. This strategy maximises Equation (3.2) with an expected return of around 600.[2]

[2]Estimates of the expected returns for strategies s1 and s2 are taken from the results in Sec 8.4.

Meta-learning strategy s2 is challenging, because the rewards are sparse and the large penalties of climbing the mountain discourage the agent to go there. In fact, most existing Meta-RL methods only learn strategy s1 and fail to maximise Equation (3.2). We come back to this environment in Chapter 8 and show how encouraging the right type of exploration *during* meta-training can enable an agent to meta-learn the optimal strategy, s2.

3.3 Meta-Training

During meta-training, our goal is to teach agents to adapt fast. This is typically done by repeatedly sampling batches of tasks from $p(M)$, and performing a small training procedure on each of them, called the "inner loop" (see Figure 1.2). In each task, the agent is given H^+ environment interactions to learn. Inner-loop learning can, e.g., include gradient updates, or rolling out a recurrent neural network. The outcome of each learning procedure is then evaluated, and the agent's meta-parameters are updated so as to maximise the expected learning success, i.e., Equation (3.2) or (3.1). This update is also often called the "outer loop" or "meta-update". The exact split between which parameters are updated in the inner and outer loop depends on the specific Meta-RL algorithm.

At meta-test time, we evaluate the agent on new tasks drawn from $p(M)$, based on the average return from Equation (3.1) or (3.2) it achieves across test tasks. An agent that does well has meta-learned to efficiently explore and adapt. Throughout the thesis, we assume that the test tasks come from the same distribution as the training tasks. We discuss the challenging open problem of distribution shift in Chapter 9.

The next section introduces the two main approaches in Meta-RL for Fast Adaptation, adapting with gradients and adapting with RNNs, and one prototypical Meta-RL method per setting.

4
Main Approaches in Meta-RL

Contents

Meta-Learning can be applied to any aspect of Machine Learning algorithms. We can meta-learn algorithms or loss functions (Schmidhuber, 1987; Bengio et al., 1992; Andrychowicz et al., 2016; Ravi et al., 2017; Houthooft et al., 2018; Bechtle et al., 2020; Oh et al., 2020), network architectures (Liu et al., 2019; Lian et al., 2019), exploration strategies (Zheng et al., 2018; Veeriah et al., 2019; Xu et al., 2018a), or hyperparameters (Zahavy et al., 2020; Franceschi et al., 2018). Meta-learning can happen across multiple tasks (Finn et al., 2017a; Kirsch et al., 2020), or to accelerate RL in a single task (Zahavy et al., 2020; Xu et al., 2020; Flennerhag et al., 2022).

As discussed in Section 3, we consider Meta-Learning for the Fast Adaptation setting, with the goal to develop RL agents that can rapidly learn new tasks. Existing approaches can be roughly divided into "gradient-based" and "memory-based" adaptation. We present the most prevalent method for each here, which form the basis for the methods presented in this thesis. Specific work related to individual contributions is discussed in the respective Chapters 5-8.

4.1 Fast Adaptation with Gradients

Learning a new task can take place in several ways. One way is gradient-based optimisation, which is also the method of choice for most of Deep RL, with the difference that we now expect the agent to only take very few gradient steps to learn a task – typically 1-10, compared to hundreds of thousands in standard RL.

In 2017, Model-Agnostic Meta-Learning (Finn et al., 2017a, MAML) was proposed, which is a general and powerful gradient-based Meta-Learning algorithm. MAML learns a model initialisation that allows fast adaptation at test time. In other words, given a policy π_θ, MAML finds an initialisation θ_0 such that we can learn any task from a task distribution $p(M)$ within just a few gradient steps when starting from θ_0.

This can be done by optimising directly for post-adaptation performance in the inner loop as follows. Given a batch of B tasks $\{M_i\}_{i=0}^{B-1}$, we update the policy separately on each task once[1] to get a new parameter $\theta_1^{(i)}$ per task:

$$\theta_1^{(i)} \leftarrow \theta_0 + \alpha_{inner} \nabla_\theta \mathbb{E}_{\pi_{\theta_0}, T_i} \left[\sum_{t=0}^{H-1} r_t^{(i)} \right]. \tag{4.1}$$

For the outer-loop update, we then separately evaluate each new parameter $\theta_1^{(i)}$ on its task, and update the *initial* θ_0 to maximise the average *post-adaptation* performance,

$$\theta_0 \leftarrow \theta_0 + \alpha_{outer} \nabla_\theta \frac{1}{B} \sum_{i=0}^{B-1} \mathbb{E}_{\pi_{\theta_1^{(i)}}, T_i} \left[\sum_{t=0}^{H-1} r_t^{(i)} \right]. \tag{4.2}$$

This requires backpropagating through the inner-loop update, which is possible since it is fully differentiable in θ, and will include higher-order gradients in θ. After meta-training, we have a policy initialisation θ_0, for which we only have to do a few gradient updates to learn any task from $p(M)$.

There exists a vast number of gradient-based Meta-Learning methods, many of which build directly on MAML (e.g., Grant et al., 2018; Yoon et al., 2018; Finn et al., 2018; Lee et al., 2018; Li et al., 2017b; Al-Shedivat et al., 2018). In all these, adaptation at meta-test time is done doing gradient updates, and

[1] We outline everything for 1 gradient step for ease of exposition, but the same optimisation procedure also applies when doing multiple gradient steps.

meta-learning is done by backpropagating through the inner-loop update. These updates are also called *meta-gradient* updates.

Adapting fast with gradients is more suitable for the Few-Shot Adaptation setting (Section 3.2.1), rather than the Online Adaptation Setting (Section 3.2.2). This is because updates to the policy can usually only happen after entire batches of data have been collected, whereas for Online Adaptation the policy needs to typically incorporate new information after every timestep in order to do well. In Chapter 5 we consider the Few-Shot Adaptation setting, and build on MAML to develop a Meta-Learning method first for the Supervised Learning and then for the Reinforcement Learning setting. Unlike MAML, our method separates task inference and task solving, as motivated in Section 3.1.1, which leads to better performance.

4.2 Fast Adaptation with RNNs

Another way of adapting to new tasks is using recurrent neural networks (see Section 2.1.4), instead of explicit gradient updates. The inner workings of the RNN, i.e., the updating of the hidden state, can then be seen as a learning algorithm itself.

Supervised Learning In Supervised Learning, we learn from a dataset $\mathcal{D} = \{(x, y)^k\}_{k=1}^K$ a function $f_\theta : x \mapsto \hat{y}$ to map datapoints x to desired outputs y as closely as possible, measured by a loss $\mathcal{L}(y, \hat{y})$. RNNs are used in Supervised Meta-Learning to, e.g., learn algorithms or optimisers (Andrychowicz et al., 2016; Ravi et al., 2017; Li et al., 2017a) or to embed data (Duan et al., 2017). The idea goes back to Hochreiter et al. (2001), which later inspired similar approaches in Meta-RL. Here, a RNN receives at each step the *target output of the previous step* as an additional input, $f_\theta^{RNN}(x^{(i+1)}, y^{(i)}, h^{(i)})$. I.e., the network first makes a prediction $\hat{y}^{(i)}$ for inputs $x^{(i)}$ with true targets $y^{(i)}$ (e.g., two images, one labelled "cat" and one labelled "dog"). Then in the next step, when making a prediction for new inputs $x^{(i+1)}$, the model is also given the correct targets $y^{(i)}$ for the *previous* prediction as additional input. By having access to the correct targets, the network can learn solely by the dynamics of the recurrent network (while the network weights themselves stay fixed).

Reinforcement Learning In the field of RL, the idea of using a recurrent neural network for Meta-Learning has been explored concurrently by Wang et al. (2016, L2RL) and Duan et al. (2016, RL^2).[2] Here, the policy is a RNN, which gets the previous reward and action as an additional input, beside the current state.

More specifically, in the inner loop, given a batch of B tasks $\{M_i\}_{i=0}^{B-1}$, learning is done by simply rolling out the recurrent policy in each task separately. At each timestep t in the current task i, the policy acts according to

$$\pi_\theta^{RNN}(a_t^{(i)}|s_t^{(i)}, r_t^{(i)}, a_{t-1}^{(i)}, h_t^{(i)}), \tag{4.3}$$

where θ are the current policy's parameters, $r_t^{(i)}$ and $a_{t-1}^{(i)}$ are the most recent reward and actions, and $h_t^{(i)}$ is the current hidden vector of the RNN (with initial hidden vector $h^{(i)} = \mathbf{0}$). By having access to the effects its action have on the environment and the reward it gets, the policy can adapt to new tasks via the recurrent dynamics and by implicitly embedding the task in the RNN hidden state.

In the outer loop, the policy parameters θ are then updated to maximise the expected online return across tasks (Equation 3.2):

$$\max_\theta \frac{1}{B} \sum_{i=0}^{B-1} \mathbb{E}_{\pi_\theta^{RNN}, T_i} \left[R(\tau) \right], \tag{4.4}$$

where τ is the trajectory collected by the recurrent policy π_θ^{RNN}. RL^2 maximises the expected online return (see Sec 3.2.2) in the original papers by Duan et al. (2016) and Wang et al. (2016), but can also be used with Equation (3.1) for Few-Shot Adaptation, by masking out the rewards for the phase where we allow the agent to freely explore (Stadie et al., 2018).

RL^2 can learn a Bayes-optimal policy (Section 2.2.1), which has been shown theoretically (Ortega et al., 2019) and empirically (Mikulik et al., 2020). This is because the RNN can, in principle, do belief updates in the hidden state. Similarly to MAML however, the agent is a black-box, and we do not have much control over or insight into what the RNN actually implements. In Chapters 6 and 7 we propose algorithms that instead do explicit belief updates, and find that this can significantly improve performance, and give us some insight into the agent's current belief.

[2] We often use the acronym RL^2 to refer to both Wang et al. (2016) and Duan et al. (2016).

Part II

Methods

5

Fast Context Adaptation via Meta-Learning

Contents

In the previous chapters, we motivated our goal of developing autonomous agents that can learn new tasks efficiently and on their own. This is necessary so that we can someday deploy RL agents safely in the real world without human oversight, even when we cannot fully pre-train the agent on all possible situations.

In particular, we argued two things. First, in many Fast Adaptation settings we should separate task inference from task solving (Sec 3.2.2). Our hypothesis is that by doing this split explicitly, we can improve performance due to more supervision and specialisation of model parts, compared to fully black-box models. Second, for many

real world applications, we should consider the Online Adaptation setting (Sec 3.2), where the agent has to perform well from the moment it starts interacting with an environment. This produces agents that learn more efficiently, and need less human oversight while doing so. The optimal solution to this is a Bayes-optimal policy (Sec 2.2), which however is intractable to compute for all but the smallest tasks.

In this chapter, we take a first step towards this goal and understanding whether an explicit split between task inference and task solving can help. To this end, we consider the setting of Few-Shot Adaptation and Supervised Learning. Recall from Section 3.2.1 that in Few-Shot Adaptation, the agent has some time to freely learn and explore, before having to perform well. This is easier since it does not require Bayes-optimal exploration; in fact, even random exploration might suffice if the rewards are dense and informative enough for the agent to infer the task. We consider the Supervised Learning setting, since it does not have some of the challenges of RL, like exploration and long-term credit assignment. It is therefore an ideal starting point to study the proposed split between task inference and task solving, in a setting where we are given a fixed dataset to learn from. Classification and regression experiments show that our proposed method leads to improved performance, scales better with the network size, and is more interpretable. We conclude the chapter with RL experiments (Section 5.3), and a discussion on shortcomings and next steps.

5.1 Context Parameters for Task Inference

In Section 3.1.1, we hypothesised that it is helpful to decouple task inference from solving the task. Following this intuition, we concretely propose to implement this via a split in the model architecture:

> When learning a new task, we first try to find out what task we are in, and summarise this in a context vector $\xi \in \mathbb{R}^d$. We then condition on this context when making a prediction.

This split is illustrated in Figure 5.1. On the left (5.1a) is a black-box model, that takes an input x (e.g., an image in classification or a state in RL) and produces an

(a) Black-Box Model **(b)** Model with Context Parameter

Figure 5.1: Task inference and task solving in different model architectures. **(a):** A black-box model takes an input x and produces an output y. The model both infers the task and computes a solution. **(b):** The model conditions on an additional input, the context ξ, representing the task. Given this and an input x, it makes a prediction y.

output y (e.g., a class label or an action). This model has to both infer the task and compute a solution. Our proposed modification is shown on the right (5.1b). The task information is summarised in a context vector ξ, which is passed to the model in addition to the input x. This task representation is not given to us a priori (in contrast to, e.g., in multi-task learning): instead, we have to learn it. We hypothesise that this architectural change has the following advantages:

- It is easier to optimise, since we have specialised network parts. The additional inductive bias on how learning should occur can lead to improved performance.
- Compressing the task information into a single vector means we can use this vector to analyse the solution, or use it for downstream tasks.

We investigate our hypotheses in this chapter by modifying the MAML model (Finn et al., 2017a), since it is a general Meta-Learning method that can be applied to both Supervised and Reinforcement Learning. Our proposed method CAVIA – *Fast* ***C****ontext* ***A****daptation* ***via*** *Meta-Learning* – updates only a context vector at test time, instead of the entire network like MAML. We evaluate our approach on Supervised Learning benchmarks, ablate its performance with regards to learning rates and network/context size, and visualise how the task is captured in the context vector. To conclude the chapter, we apply CAVIA to a Reinforcement Learning problem, and discuss the advantages and shortcomings of this approach.

5.2 Supervised Learning

In this section we introduce the Supervised Learning setting, and propose our method that uses context adaptation for Fast Adaptation. Experiments on regression and classification tasks evaluate our proposed method.

5.2.1 Background

In Supervised Learning, we are given a dataset $\mathcal{D} = \{(x, y)^k\}_{k=1}^K$ of K data points x (e.g., images) and associated labels y (e.g., object classes), and we want to learn a model $f_\theta : x \mapsto \hat{y}$ that maps data points to predictions $\hat{y} \in \mathcal{Y}$ as accurately as possible. To train and evaluate a model, we use a loss function $\mathcal{L}(f_\theta(x), y)$ (e.g., the mean squared error for regression, or the binary cross entropy for classification). We typically divide the dataset into disjoint sets; a training set $\mathcal{D}^{train}$ to learn the parameter θ, and a test set $\mathcal{D}^{test}$ to evaluate the learned model. In standard Supervised Learning, the size K of the dataset is large (thousands to millions of datapoints). When this is not possible, and we are only given a small dataset (e.g., a handful of datapoints) from which to learn f_θ, we refer to this as the "Few-Shot Learning" setting. This is similar to the Few-Shot Adaptation RL setting that we introduced in Section 3.2.1, where the agent can freely collect some data before being evaluated. A general way to train Few-Shot Learning methods is to train on a set of related tasks, e.g., using Meta-Learning.

Meta-Learning for Few-Shot Supervised Learning We assume access to a distribution over tasks, $p(\mathcal{T})$, where a task is defined as a tuple $\mathcal{T}_i = (\mathcal{X}_i, \mathcal{Y}_i, \mathcal{L}_i, q_i)$. Here, $\mathcal{X}_i$ is the input space, $\mathcal{Y}_i$ the output space, $\mathcal{L}_i(y, \hat{y})$ a task-specific loss function, and $q_i(x, y)$ a distribution over labelled data points. Different tasks can be created by changing any element of $\mathcal{T}_i$. For example, in a 2-way classification task, we can change $\mathcal{X}_i$ and $\mathcal{Y}_i$ so that one task has images and labels of cats and dogs, and another task has images and labels of birds and snakes.

During meta-training, at each iteration, a batch of B tasks $\mathbf{T} = \{\mathcal{T}_i\}_{i=1}^{B}$ is sampled from p. For each task $\mathcal{T}_i \in \mathbf{T}$, we then sample two datasets $\mathcal{D}_i^{\text{train}}$ and $\mathcal{D}_i^{\text{test}}$:

$$\mathcal{D}_i^{\text{train}} = \{(x,y)^{i,m}\}_{m=1}^{M_i^{\text{train}}}, \quad \mathcal{D}_i^{\text{test}} = \{(x,y)^{i,m}\}_{m=1}^{M_i^{\text{test}}}, \tag{5.1}$$

where $(x,y) \sim q_i$, and M_i^{train} and M_i^{test} are the number of training and test datapoints. The training data is used to update f_θ, and the test data is then used to evaluate how good this update was, and adjust f_θ or the update rule accordingly. After meta-training, the model is evaluated on unseen tasks drawn from p.

MAML for Supervised Learning MAML (Finn et al., 2017a, see Section 4.1) can be applied to both Supervised and Reinforcement Learning problems. We briefly outline the MAML equations for the Supervised Learning setting here (for the RL ones see Section 4.1).

In the inner loop, MAML computes new task-specific parameters θ_i (starting from θ) via one gradient update[1],

$$\theta_i = \theta - \alpha \nabla_\theta \frac{1}{M_{\text{train}}^i} \sum_{(x,y)\in\mathcal{D}_i^{\text{train}}} \mathcal{L}_{\mathcal{T}_i}(f_\theta(x), y) \ . \tag{5.2}$$

For the meta-update in the outer loop, the *original* model parameters θ are then updated with respect to the performance after the inner-loop update, i.e.,

$$\theta \leftarrow \theta - \beta \nabla_\theta \frac{1}{N} \sum_{\mathcal{T}_i \in \mathbf{T}} \frac{1}{M_{\text{test}}^i} \sum_{(x,y)\in\mathcal{D}_i^{\text{test}}} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i}(x), y) \ . \tag{5.3}$$

Since the gradient is taken with respect to the parameters θ before the inner-loop update (5.2), the outer-loop update (5.3) involves higher order derivatives of θ.

5.2.2 CAVIA

In MAML (Finn et al., 2017a), the entire network is updated when learning a new task. It therefore resembles our illustration in Figure 5.1a, where the model is a black-box that does both task solving and task inference. Instead, we propose to split these two aspects when learning to tasks as illustrated in Figure 5.1b.

[1]We outline MAML for one gradient update step, but it can be used with several gradient update steps as well.

To this end, we propose *Fast **C**ontext **A**daptation **via** Meta-Learning* (CAVIA). CAVIA modifies MAML such that the model now has an additional input, a context vector $\xi \in \mathbb{R}^d$ of length d, that is separate from the model parameters θ. We train this model such that at meta-test time, when learning a new task, we adapt by simply updating the context parameter ξ. The training procedure is as follows.

At every meta-training iteration for the current batch of tasks $\mathbf{T}$, we use the training data $\mathcal{D}_i^{\text{train}}$ of each task $\mathcal{T}_i \in \mathbf{T}$ as follows. Starting from a fixed value ξ_0 (typically $\xi_0 = \mathbf{0}$), we learn *task-specific* parameters ξ_i via one gradient update:

$$\xi_i = \xi_0 - \alpha \nabla_\xi \frac{1}{M_i^{\text{train}}} \sum_{(x,y)\in\mathcal{D}_i^{\text{train}}} \mathcal{L}_{\mathcal{T}_i}(f_{\xi_0,\theta}(x), y). \tag{5.4}$$

While we only take the gradient with respect to ξ, the updated parameter ξ_i is also a function of θ, since during backpropagation, gradients flow through the model. Given updated parameters ξ_i for all sampled tasks, we update θ in the outer loop with

$$\theta \leftarrow \theta - \beta \nabla_\theta \frac{1}{N} \sum_{\mathcal{T}_i\in\mathbf{T}} \frac{1}{M_i^{\text{test}}} \sum_{(x,y)\in\mathcal{D}_i^{\text{test}}} \mathcal{L}_{\mathcal{T}_i}(f_{\xi_i,\theta}(x), y) \ . \tag{5.5}$$

This update includes higher order gradients in θ due to the dependency on (5.4). The model parameters θ are shared across tasks. At test time, *only* the context parameters are updated, and θ is held fixed. Algorithm 1 shows pseudo-code for CAVIA.

Algorithm 1 CAVIA for Supervised Learning

Require: Distribution over tasks $p(\mathcal{T})$
Require: Step sizes α and β
Require: Initial model $f_{\xi_0,\theta}$ with θ initialised randomly and $\xi_0 = 0$
while not done **do**
  Sample batch of tasks $\mathbf{T} = \{\mathcal{T}_i\}_{i=1}^B$ where $\mathcal{T}_i \sim p$
  for all $\mathcal{T}_i \in \mathbf{T}$ **do**
    $\mathcal{D}_i^{\text{train}}, \mathcal{D}_i^{\text{test}} \sim q_{\mathcal{T}_i}$
    $\xi_0 = 0$
    $\xi_i = \xi_0 - \alpha \nabla_\xi \frac{1}{M_i^{\text{train}}} \sum_{(x,y)\in\mathcal{D}_i^{\text{train}}} \mathcal{L}_{\mathcal{T}_i}(f_{\xi_0,\theta}(x), y)$
  end for
  $\theta \leftarrow \theta - \beta \nabla_\theta \frac{1}{N} \sum_{\mathcal{T}_i\in\mathbf{T}} \frac{1}{M_i^{\text{test}}} \sum_{(x,y)\in\mathcal{D}_i^{\text{test}}} \mathcal{L}_{\mathcal{T}_i}(f_{\xi_i,\theta}(x, y))$
end while

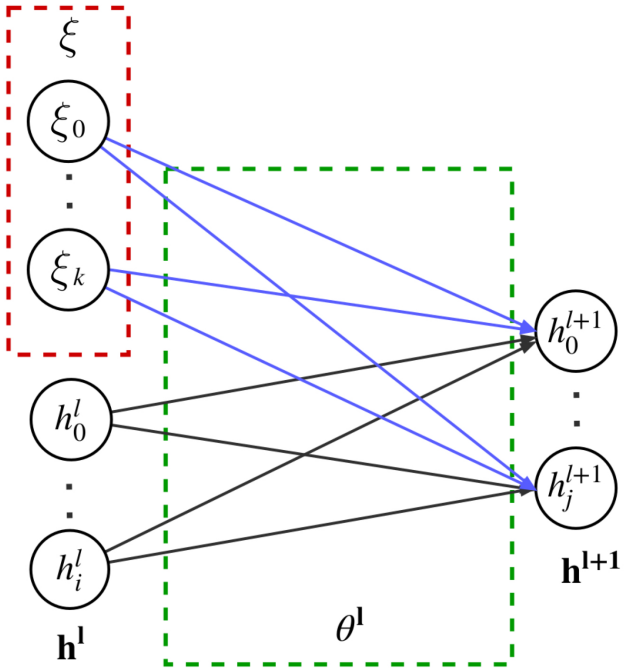


Figure 5.2: Context adaptation in CAVIA. The inputs to a network layer h^l are augmented with a context vector ξ (red). The context is initialised to $\mathbf{0}$ before each adaptation step and updated by gradient descent during the inner loop. Network parameters θ (green) are only updated in the outer loop and shared across tasks. Hence, they stay fixed at test time. By initialising ξ to $\mathbf{0}$, the network parameters associated with the context parameters (blue) do not affect the layer output before adaptation. After the first adaptation step they modulate the rest of the network for the given task.

Conditioning on Context Parameters Since ξ is independent of the network input, we are free to decide where and how to condition the network on the context. For a fully connected layer l, we can for example simply concatenate ξ to the layer inputs $h_j^{(l-1)}$, and get

$$h_i^{(l)} = g\left(\sum_{j=1}^{J} \theta_{j,i}^{(l,h)}\ h_j^{(l-1)} + \sum_{k=1}^{K} \theta_{k,i}^{(l,\xi)}\ \xi_{0,k} + b\right), \tag{5.6}$$

where g is a nonlinear activation function, b a bias parameter, $\theta_{j,i}^{(l,h)}$ the weights for the layer inputs $h_j^{(l-1)}$, $\theta_{k,i}^{(l,\xi)}$ the weights for the context parameter $\xi_{0,k}$, and $h_i^{(l)}$ the output. In our experiments, for fully connected networks, we add the context parameter at the first layer, i.e., concatenate them to the input.

For convolutional networks, we use *feature-wise linear modulation* (FiLM, Perez et al., 2017), which performs an affine transformation on the feature maps. Given

a context ξ and a convolutional layer that outputs L feature maps $\{h_i\}_{i=1}^{L}$, FiLM linearly transforms each feature map $FiLM(h_i) = \gamma_i h_i + \beta$, where $\gamma, \beta \in \mathbb{R}^M$ are a function of the context parameters. We use a fully connected layer $[\gamma, \beta] = \sum_{k=1}^{K} \theta_{k,i}^{(l,\xi)} \xi_{0,k} + b$ with the identity function at the output.

Context Parameter Initialisation When learning a new task, the context has to be initialised to some value, ξ_0. Instead of meta-learning this initialisation, a fixed ξ_0 suffices: in (5.6), if both $\theta_{j,i}^{(l,\xi)}$ and ξ_0 are meta-learned, the learned initialisation of ξ can be subsumed into the bias parameter b, and ξ_0 can be set to a fixed value. Hence, the initialisation of the context parameters does not have to be meta-learned and parameter copies are not required during training. In our experiments we set the initial context to a zero vector, $\xi_0 = \mathbf{0} = (0, \ldots, 0)$.

Furthermore, not updating the context parameters ξ in the outer loop allows for a more expressive gradient in the inner loop, and makes CAVIA more robust to the inner loop learning rate, α in (5.4). Before an inner loop update, the part of the model associated with ξ does not affect the output (since they are inputs and initialised at $\mathbf{0}$). During the inner update, only ξ changes and can affect the output of the network at test time. Even if this update is large, the parameters $\theta_{k,i}^{(l,\xi)}$ that connect ξ to the rest of the model, are automatically scaled during the outer loop. In other words, the weights $\theta_{k,i}^{(l,\xi)}$ which connect the context ξ to the model, can compensate for any excessively large inner loop update via their weight magnitude. However, doing large gradient updates in every outer loop update step as well would lead to divergence and numerical overflow. In Section 5.2.3, we show empirically that the decoupling of learning ξ and θ can indeed make CAVIA more robust to the initial learning rate compared to also learning the initialisation of the context parameters.

Next, we empirically evaluate CAVIA on regression and classification tasks. Code is available at `https://github.com/lmzintgraf/cavia`.

	Number of Additional Input Parameters						
Method	0	1	2	3	4	5	50
CAVIA	-	0.84	0.21	0.20	**0.19**	**0.19**	**0.19**
MAML	0.33	0.29	0.24	0.24	**0.23**	**0.23**	**0.23**

Table 5.1: Results for the sine curve regression task. Shown is the mean-squared error of CAVIA and MAML for varying number of input parameters. The 95% confidence intervals are 0.06 for CAVIA with context size 1, and 0.02 everywhere else.

5.2.3 Experiments: Regression

Sine Curves We start with the regression problem of fitting sine curves used in Finn et al. (2017a). A task is defined by the amplitude and phase of a sine curve and generated by uniformly sampling the amplitude from $[0.1, 0.5]$ and the phase from $[0, \pi]$. For training, we use a batch of 25 tasks, and for each, ten labelled datapoints (uniformly sampled from $x \in [-5, 5]$) are given. The inner-loop update is done using a mean-squared error (MSE) loss. We use a neural network with two hidden layers and 40 nodes each. The number of context parameters varies between 2 and 50. During testing we present the model with ten datapoints from 1000 newly sampled tasks and measure the MSE over 100 test points. To allow a fair comparison, we add additional input biases to MAML (the same number as context parameters that CAVIA uses), an extension that was also done by Finn et al. (2017b). These additional parameters are meta-learned together with the rest of the network.

Table 5.1 shows that CAVIA outperforms MAML even when MAML gets the same number of additional parameters, despite the fact that CAVIA adapts only 2-5 parameters, instead of around 1600. CAVIA's performance on the regression task correlates with how many variables are needed to encode the tasks. In these experiments, two parameters vary between tasks, which is exactly the context parameter dimensionality at which CAVIA starts to perform well. This suggests CAVIA indeed learns task descriptions in the context parameters via backpropagation at test time. Figure 5.3 illustrates this by plotting the value of the learned inputs against the amplitude/phase of the task in the case of two context parameters. The model learns a smooth embedding in which interpolation between tasks is possible.

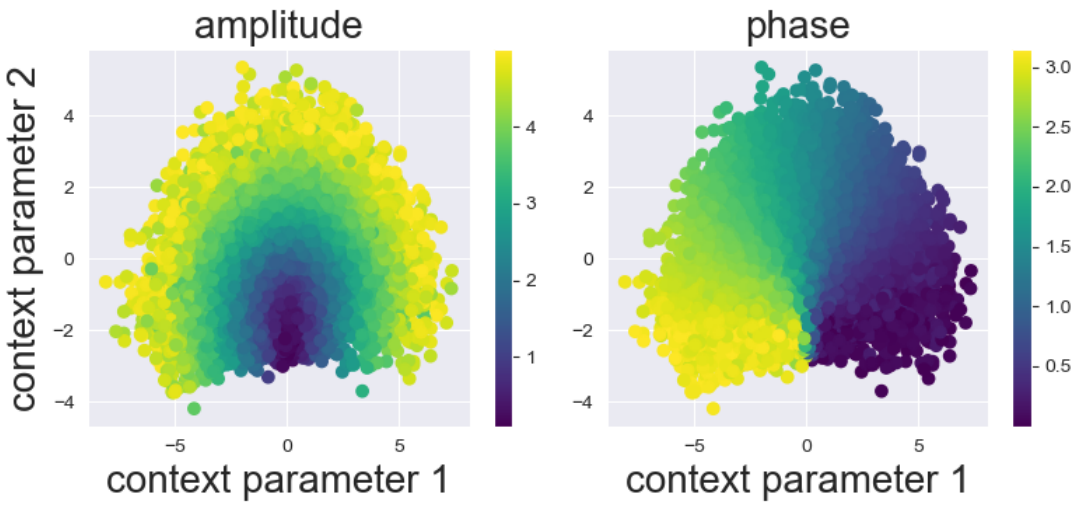
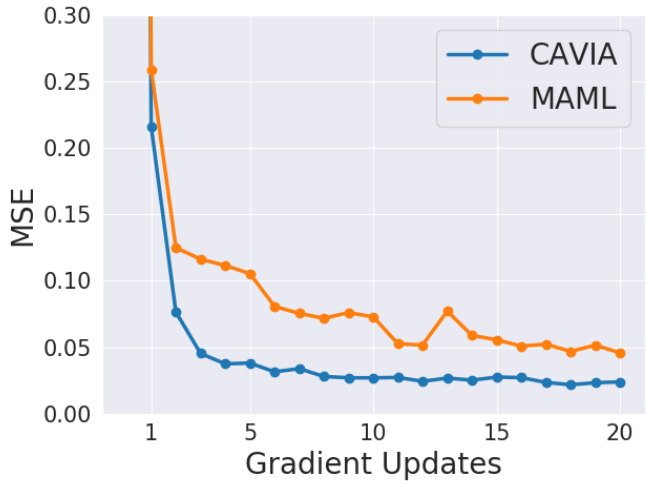


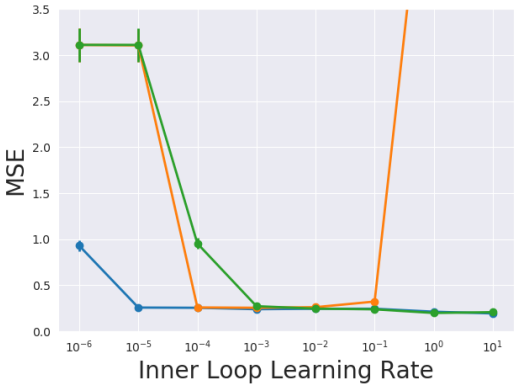
Figure 5.3: Visualisation of what two context parameters learn on a new task. Shown is the value they take after 5 gradient update steps on a new task. Each dot is one random task; its colour indicates the amplitude (left) or phase (right) of that task.

Figure 5.4 shows additional analyses on the experimental results. Figure 5.4a shows that when performing more gradient steps at test time compared to training, CAVIA outperforms MAML and has a more stable, monotonic learning curve. As described in Section 5.2.2, CAVIA can scale the gradients of the context parameters since they are inputs to the model and trained separately. Figure 5.4b shows the performance of CAVIA, MAML, and CAVIA when also learning the initialisation of ξ (i.e., updating the context parameters in the outer loop), for a varying learning rate from 10^{-6} to 10. CAVIA is robust to changes in learning rate while MAML performs well only in a small range. Figure 5.4c gives insight into how CAVIA does this: we plot the inner learning rate against the norm of the gradient of the context parameters at test time. The weights are adjusted so that lower learning rates bring about larger context parameter gradients and vice-versa. MT-Nets (Lee et al., 2018), which learn which subset of parameters to adapt on a new task, are also robust to the inner-loop learning rate, but in a smaller range than CAVIA.[2] Similarly, Li et al. (2017b) show that MAML can be improved by learning a parameter-specific learning rate, which, however, introduces a lot of additional parameters.

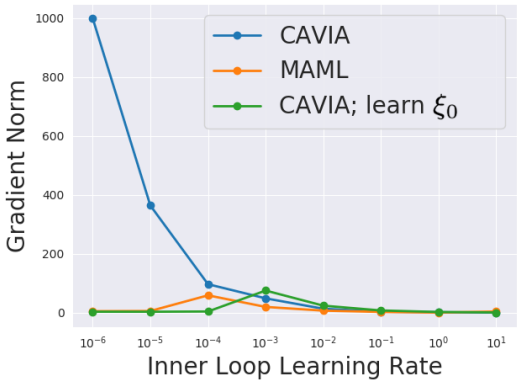
[2]See Lee et al. (2018). We do not show the numbers they report since we outperform them significantly, likely due to a different experimental protocol.



(a) Test Performance



(b) Learning Rates



(c) Gradient Norms

Figure 5.4: Analysis of the sine curve experiments. (a) Test performance after several gradient steps on the same batch, averaged over 1000 unseen tasks. Both CAVIA and MAML continue to learn, but CAVIA is more stable with a monotonic learning curve. (b) Test Performance after training with different inner loop learning rates for MAML, CAVIA, and CAVIA with a learned ξ_0. (c) CAVIA scales the model weights so that the inner learning rate is compensated by the context parameters gradients magnitude.

Image Completion To evaluate CAVIA on a more challenging regression task, we consider image completion (Garnelo et al., 2018a). The task is to predict pixel values from coordinates, i.e., learn a function $f : [0, 1]^2 \rightarrow [0, 1]^3$ which maps 2D pixel coordinates $x \in [0, 1]^2$ to pixel intensities $y \in [0, 1]^3$ (RGB values). An individual picture is considered a single task, and we are given a few pixels as a training set $\mathcal{D}^{\text{train}}$ and use the entire image as the test set $\mathcal{D}^{\text{test}}$ (including the training set). We train CAVIA on the CelebA (Liu et al., 2015) training set, perform model selection on the validation set, and evaluate on the test set.

	Random Pixels			Ordered Pixels		
	10	100	1000	10	100	1000
CNP*	0.039	0.016	0.009	0.057	**0.047**	0.021
MAML	0.040	0.017	**0.006**	0.055	**0.047**	0.007
CAVIA	**0.037**	**0.014**	**0.006**	**0.053**	**0.047**	**0.006**

Table 5.2: Quantitative image completion results. Average pixel-wise MSE for the image completion task on the CelebA dataset. We test different number of training points per image (10, 100, 1000). The training pixels are chosen either at random or ordered from the top-left corner to the bottom-right. (*Results from Garnelo et al. (2018a))

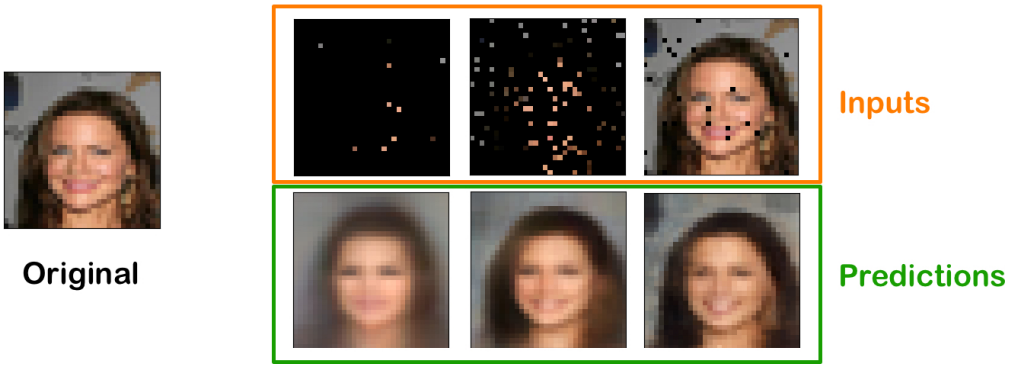


Figure 5.5: Qualitative image completion results. Left: True image. Top row: training pixels for 10, 100, and 1000 training points. Bottom row: prediction of CAVIA when 128 context parameters were updated for 5 gradient steps.

Garnelo et al. (2018a) use an MLP encoder with three hidden layers of width 128, a 128-dimensional embedding, and a five-layer decoder of width 128. To allow a fair comparison, we choose a context vector of size 128, and an MLP with five hidden layers of width 128 for the main network. We chose an inner-learning rate of 1.0 without tuning. For MAML we use the same five-layer MLP network including 128 additional input biases, and an inner-loop learning rate of 0.1 (other tested values: 1.0, 0.01). CAVIA and MAML were trained with five inner-loop gradient updates.

Table 5.2 shows the results in terms of pixel-wise MSE for different numbers of training pixels ($k = 10, 100, 1000$ shot), and for the case of randomly selected and ordered pixels (starting from the top left of the image). CAVIA outperforms CNPs and MAML in most settings. Figure 5.5 shows an example image reconstruction produced by CAVIA (more results in Appendix A). These results show that it is possible to learn an embedding only via backpropagation and with far fewer parameters than when using a separate embedding network.

5.2.4 Experiments: Classification

To evaluate how CAVIA scales to problems that require larger networks, we test it on the few-shot image classification benchmark Mini-Imagenet (Ravi et al., 2017). In N-way K-shot classification, a task is a random selection of N classes, for each of which the model gets to see K examples. From these it must learn to classify unseen images from the N classes. The Mini-Imagenet dataset consists of 64 training classes, 12 validation classes, and 24 test classes. During training, we generate a task by selecting N random classes and K examples of each, i.e., a batch of $N \times K$ images. The meta-update is done on a set of unseen images of the same classes.

On this benchmark, MAML uses a network with four convolutional layers with 32 filters each and one fully connected layer at the output (Finn et al., 2017a). We use the same network architecture, but we also test scaling up the number of filters, up to 512 filters per layer. We use 100 context parameters and add a FiLM layer that conditions on these after the third convolutional layer and whose parameters are meta-learned with the rest of the network, i.e., they are part of θ. All our models were trained with two gradient steps in the inner loop and evaluated with two gradient steps. Following Finn et al. (2017a), we ran each experiment for $60,000$ meta-iterations and selected the model with the highest validation accuracy for evaluation on the test set.

Table 5.3 shows our results on Mini-Imagenet held-out test data for 5-way 1-shot and 5-shot classification. Our smallest model (32 filters) underperforms MAML (within the confidence intervals), and our largest model (512 filters) clearly outperforms MAML. CAVIA benefits from increasing model expressiveness: since we only adapt the context parameters in the inner loop per task, we can substantially increase the network size without overfitting during the inner loop update. We tested scaling up MAML to a larger network size as well (see Table 5.3), but found that this hurt accuracy, which was also observed by Mishra et al. (2018). We also include results for the first order approximation of our largest models, where the gradient with respect to θ is not backpropagated through the inner loop. As expected, this results in a lower accuracy (a drop of 2 percentage points).

Method	5-way accuracy	
	1-shot	5-shot
Matching Nets (Vinyals et al., 2016)	46.6%	60.0%
Meta LSTM (Ravi et al., 2017)	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$
Prototypical Nets (Snell et al., 2017)	$46.61 \pm 0.78\%$	$65.77 \pm 0.70\%$
Meta-SGD (Li et al., 2017b)	$50.47 \pm 1.87\%$	$64.03 \pm 0.94\%$
REPTILE (Nichol et al., 2018)	$49.97 \pm 0.32\%$	$65.99 \pm 0.58\%$
MT-NET (Lee et al., 2018)	$\mathbf{51.70} \pm 1.84\%$	-
VERSA (Gordon et al., 2019)	$\mathbf{53.40} \pm 1.82\%$	$\mathbf{67.37} \pm 0.86$
MAML (32) (Finn et al., 2017a)	$48.07 \pm 1.75\%$	$63.15 \pm 0.91\%$
MAML (64)	$44.70 \pm 1.69\%$	$61.87 \pm 0.93\%$
CAVIA (32)	$47.24 \pm 0.65\%$	$59.05 \pm 0.54\%$
CAVIA (128)	$49.84 \pm 0.68\%$	$64.63 \pm 0.54\%$
CAVIA (512)	$\mathbf{51.82} \pm 0.65\%$	$65.85 \pm 0.55\%$
CAVIA (512, first order)	$49.92 \pm 0.68\%$	$63.59 \pm 0.57\%$

Table 5.3: Few-shot classification results on the Mini-Imagenet test set. Average accuracy with 95% confidence intervals on a random set of 1000 tasks For MAML with 32 filters, we show the results reported by Finn et al. (2017a). For 64 filters, we ran the author's public code. These results show that CAVIA is able to scale to larger networks without overfitting, and outperforms MAML by doing so. The table includes other CNN-based methods with similar experimental protocol. We did not tune CAVIA to compete with these methods, but focus on the comparison to MAML.

We focus these experiments on the comparison to MAML, since we aim to validate the context-based approach. We did not tune CAVIA in terms of network architecture or other hyperparameters, but only varied the number of filters at each convolutional layer. At the time of publication, the best concurrent method with similar architecture and experimental protocol was VERSA (Gordon et al., 2019), which learns to produce weights of the classifier, instead of modulating the network. The state-of-the-art results in Mini-Imagenet was the method LEO Rusu et al. (2019), which uses pre-trained feature representations from a deep residual network (He et al., 2016b) and a different experimental protocol.

In conclusion, CAVIA can achieve higher accuracies than MAML by increasing the network size, without overfitting. CAVIA adjusts only 100 parameters at test time, compared to $> 30,0000$ in MAML. These results confirm that separating task inference from solving with the context-based approach is valuable. Encouraged by these results, we turn to the Reinforcement Learning setting next.

5.3 Reinforcement Learning

Following the success of CAVIA across a variety of Supervised Learning settings, we now evaluate the context-based approach in an RL setting.

Meta-Training During each meta-training iteration of MAML and CAVIA, for each $\mathcal{T}_i \in \mathbf{T}$, we first collect a trajectory

$$\tau_i^{\text{train}} = \{s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{M_i^{\text{train}}-1}, a_{M_i^{\text{train}}-1}, r_{M_i^{\text{train}}}, s_{M_i^{\text{train}}}\}, \tag{5.7}$$

where actions are chosen by the current policy π, and M_i^{train} is the number of free environment interactions that we allow the agent (see Section 3.2.1). We unify several episodes in this formulation: if the MDP horizon H is reached within the trajectory, the environment is reset. Once the trajectory is collected, this data is used to update the policy. Another trajectory τ_i^{test} is then collected by rolling out the updated policy for M_i^{test} timesteps. This test trajectory is used to evaluate the quality of the update on that task, and to adjust π or the update rule accordingly.

5.3.1 CAVIA for RL

During each iteration, for a current batch of MDPs $\mathbf{T} = \{\mathcal{T}_i\}_{i=1}^{B}$, we proceed as follows. Given ξ_0, we collect a rollout τ_i^{train} by executing the policy $\pi_{\xi_0,\theta}$. We then compute task-specific parameters ξ_i via one gradient update:

$$\xi_i = \xi_0 + \alpha \nabla_\xi \tilde{\mathcal{J}}_{\mathcal{T}_i}(\tau_i^{\text{train}}, \pi_{\xi_0,\theta}), \tag{5.8}$$

where $\tilde{\mathcal{J}}(\tau, \pi)$ is an RL objective such as a policy gradients objective (see Sec 2.1). After updating the policy, we collect another trajectory τ_i^{test} to evaluate the updated policy, where actions are chosen according to the updated policy $\pi_{\xi_i,\theta}$. After doing this for all tasks in $\mathbf{T}$, the meta-update step updates θ to maximise the average performance across tasks (after individually updating ξ for them),

$$\theta \leftarrow \theta + \beta \nabla_\theta \frac{1}{N} \sum_{\text{MDP}_i \in \mathbf{T}} \tilde{\mathcal{J}}_{\mathcal{T}_i}(\tau_i^{\text{test}}, \pi_{\xi_i,\theta}). \tag{5.9}$$

This update includes higher order gradients in θ due to the dependency on (5.8). Algorithm 2 shows pseudo-code for CAVIA in the RL setting.

Algorithm 2 CAVIA for RL

Require: Distribution over tasks $p(\mathcal{T})$
Require: Step sizes α and β
Require: Initial policy $\pi_{\xi_0,\theta}$ with θ initialised randomly and $\xi_0 = 0$
while not done **do**
  Sample batch of tasks $\mathbf{T} = \{\mathcal{T}_i\}_{i=1}^{B}$ where $\mathcal{T}_i \sim p$
  for all $\mathcal{T}_i \in \mathbf{T}$ **do**
    Collect rollout τ_i^{train} using $\pi_{\xi_0,\theta}$
    $\xi_i = \xi_0 + \alpha \nabla_\xi \tilde{\mathcal{J}}_{\mathcal{T}_i}(\tau_i^{\text{train}}, \pi_{\xi_0,\theta})$
    Collect rollout τ_i^{test} using $\pi_{\xi_i,\theta}$
  end for
  $\theta \leftarrow \theta + \beta \nabla_\theta \frac{1}{N} \sum_{\mathcal{T}_i \in \mathbf{T}} \tilde{\mathcal{J}}_{\mathcal{T}_i}(\tau_i^{\text{test}}, \pi_{\xi_i,\theta})$
end while

5.3.2 Experiments

MuJoCo We apply CAVIA to two high-dimensional RL tasks using the MuJoCo physics engine (Todorov et al., 2012a) and setup from Finn et al. (2017a). In the first experiment, *CheetahDir*, a simulated HalfCheetah (shown in Figure 5.6) must run in a randomly chosen direction (forward/backward), and receives as reward its speed in that direction. In the second experiment, *CheetahVel*, the simulated HalfCheetah must run at a particular velocity, chosen uniformly at random between 0.0 and 2.0. The agent's reward is the negative absolute value between its current and the target velocity. The agent gets 20 rollouts of length to explore (of length 200 each). After collecting this data, we perform gradient updates, and then evaluate the agent. The meta-batchsize is 40 tasks per outer update. As in Finn et al. (2017a), our agents are trained for one gradient update, using policy gradient with generalised advantage estimation (Schulman et al., 2015b) in the inner loop and TRPO (Schulman et al., 2015a) in the outer loop update. Following the protocol of Finn et al. (2017a), both CAVIA and MAML are trained for up to 500 meta-iterations, and the models with the best average return during training were used for evaluation. For these tasks, we use 50 context parameters for CAVIA and an inner-loop learning rate of 10. We found that starting with a higher learning rate helps for RL problems, since the policy update in the outer loop has a stronger signal from the context parameters.

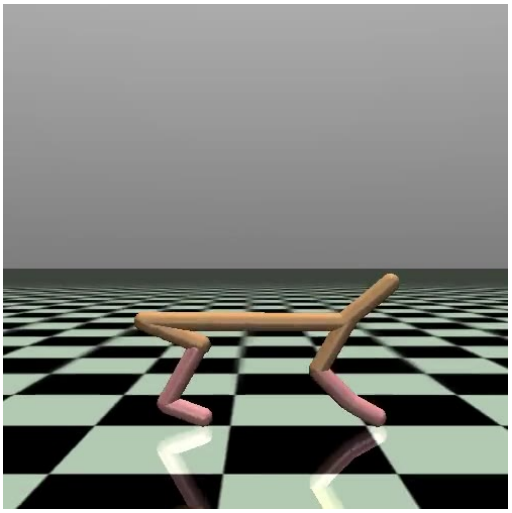
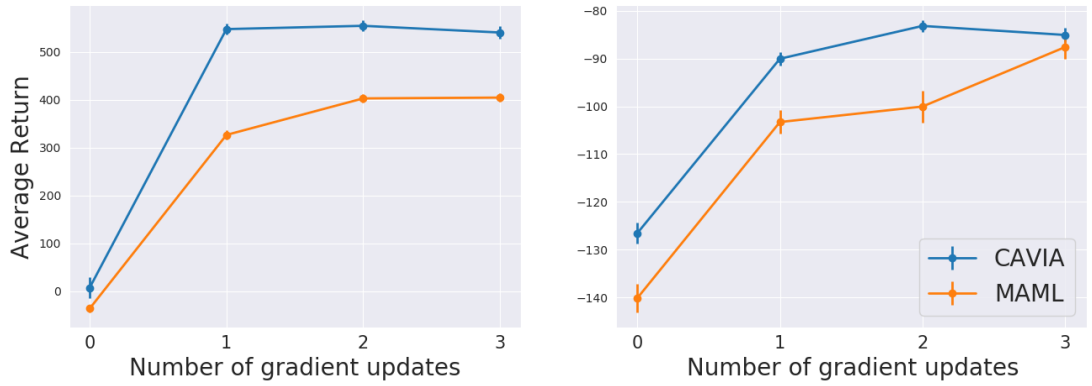


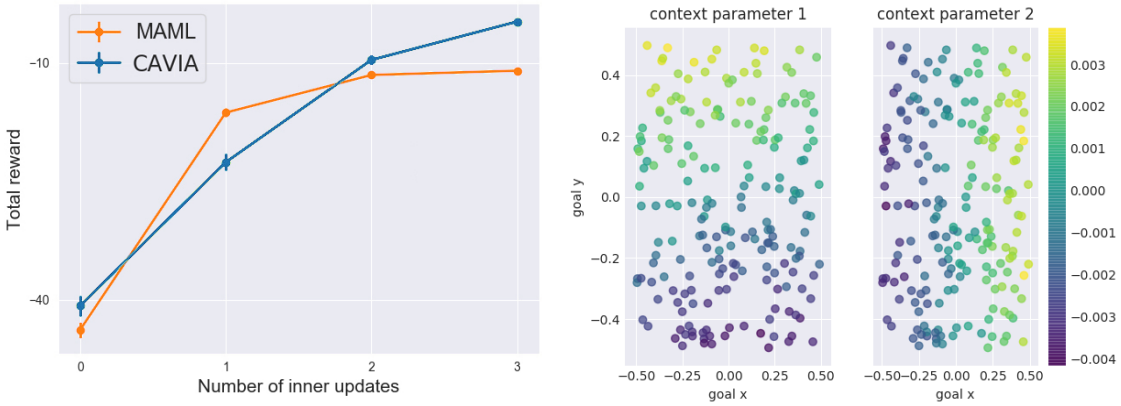
Figure 5.6: Screenshot of the HalfCheetah in the physics engine MuJoCo. The agent controls multiple joints of the HalfCheetah, which can run to the left or right.



(a) Cheetah Direction **(b)** Cheetah Velocity

Figure 5.7: Results for two MuJoCo HalfCheetah tasks. Both agents were trained to perform one gradient update, but are evaluated for several update steps. Results are averaged over 40 randomly selected tasks.

Figure 5.7 shows the performance of the CAVIA and MAML agents at test time, after up to three gradient steps (averaged over 40 randomly selected test tasks). Both models keep learning for several updates, although they were only trained for one update step. CAVIA outperforms MAML on both domains after one gradient update step, while updating only 50 parameters at test time per task compared to over $10,000$. For the Cheetah Velocity experiment, MAML catches up after three gradient update steps.



(a) Multiple gradient updates. **(b)** Goal position embedding.

Figure 5.8: Results for the 2D navigation task. Left (a): Performance at test time when updating MAML and CAVIA for up to three gradient steps. Right (b): Visualisation of the 2D context parameter in CAVIA after adaptation. The encoding of the two-dimensional goal coordinates are clearly reflected in the learned context vector.

2D Navigation We also perform RL experiments on the 2D Navigation task of Finn et al. (2017a). The agent moves in a 2D world using continuous actions and at each timestep is given a negative reward proportional to its distance from a pre-defined goal position. Each task has a new unknown goal position.

We follow the same procedure as Finn et al. (2017a). Goals are sampled from an interval of $(x, y) = [-0.5, 0.5]$. At each step we sample 20 tasks for both the inner and outer loops and testing is performed on 40 new unseen tasks. We learn for 500 iterations and optimise for one gradient update in the inner loop. The best performing policy during training is then presented with new test tasks and allowed two gradient updates. For each update, the total reward over 20 rollouts per task is measured. We use a two-layer network with 100 units per layer and ReLU nonlinearities to represent the policy and a linear value function approximator. For CAVIA we use five context parameters at the input layer. Figure 5.8a shows that the two methods are highly competitive. Notably, CAVIA adapts only five parameters at test time, whereas MAML adapts the entire network which consists of around $10,000$. As with regression, the optimal task embedding is low dimensional enough to visualise. To this end, we train CAVIA with only two context parameters and plot how these correlate with the actual position of the goal for 200 test tasks.

Figure 5.8b shows that the context parameters obtained after two policy gradient updates represent a disentangled embedding of the actual task. Specifically, context parameter 1 encodes the y position of the goal, while context parameter 2 encodes the x position. This confirms that CAVIA can indeed learn compact interpretable task embeddings via backpropagation through the inner loss.

We return to a similar environment in a more difficult form in Chapter 6 for the Online Adaptation setting. There we consider a 2D navigation task where goals can be along a half-circle (similar to what we showed in Figure 3.1a), and where the rewards are sparse in the sense that the agent only gets a reward signal if it is close to the goal. This requires the agent to explore efficiently along the 2D half-circle.

5.4 Related Work

Compared to many existing gradient-based Meta-Learning approaches like MAML (Finn et al., 2017a), CAVIA adapts only a few parameters at test time: a context vector ξ that the model conditions on. Closely related are MT-Nets (Lee et al., 2018), which learn *which* parameters to update in MAML. MT-Nets learn an M-Net which is a mask indicating which parameters to update in the inner loop, sampled (from a learned probability distribution) for each new task; and a T-net which learns a task-specific update direction and step size. CAVIA is a simpler, more interpretable alternative where the task-specific and shared parameters are disjoint sets.

Additional input biases to MAML were considered by Finn et al. (2017b), who show that this improves performance on a robotic manipulation setting. By contrast, we update *only* the context parameters in the inner loop, and initialise them to zero before adaptation to a new task. Rei (2015) propose a similar approach in the context of neural language models, where a context vector represents the sentence that is currently being processed (see also the Appendix of Finn et al. (2017a)). Unlike CAVIA, this approach updates context parameters in the outer loop, i.e., it learns the initialisation of ξ. This coupling of the gradient updates leads to a less flexible meta-update and is not as robust to the inner loop learning rate like CAVIA, as we show empirically in 5.2.3.

Silver et al. (2008) proposed context features as a component of inductive transfer, using a predefined one-hot encoded task-specifying context as input to the network. They show that this works better than learning a shared feature extractor and having separate heads for all tasks. In CAVIA, we instead *learn* this contextual input from data of a new task. Such context features can also be learned by a separate embedding network as in, e.g., Oreshkin et al. (2018) and Garnelo et al. (2018a), who use the task's training set to condition the prediction network. CAVIA instead learns the context parameters via backpropagation through the same network used to solve the task.

Several methods learn to produce network weights from task-specific embeddings or labelled datapoints (Gordon et al., 2019; Rusu et al., 2019), which then operate on the task-specific inputs. By contrast, we learn an embedding that modulates a fixed network, and is independent of the task-specific inputs during the forward pass. Specific to few-shot image classification, metric-based approaches learn to relate (embeddings of) labelled images and new instances of the same classes (Snell et al., 2017; Sung et al., 2018). By contrast, CAVIA can be used for regression and RL as well. Other Meta-Learning methods are also motivated by the practical difficulties of learning in high-dimensional parameter spaces, and the relative ease of fast adaptation in lower dimensional space (e.g., Sæmundsson et al., 2018; Zhou et al., 2018).

In the context of Reinforcement Learning, Gupta et al. (2018) condition the policy on a latent random variable trained similarly to CAVIA, together with the reparametrisation trick (although they do not explicitly interpret these parameters as task embeddings). This latent variable is sampled once per episode, and thus allows for structured exploration. Unlike CAVIA, they adapt the entire network at test time, which can be prone to overfitting.

CAVIA is conceptually related to embedding-based approaches for fast adaptation such as *conditional neural processes* (CNPs) (Garnelo et al., 2018a) and *meta-learning with latent embedding optimisation* (LEO) (Rusu et al., 2019). These share the benefit of learning a low-dimensional representation of the task, which has

the potential to lead to greater interpretability compared to MAML. In contrast to existing methods, CAVIA uses the same network to learn the embedding (during a backward pass) and make predictions (during a forward pass). Therefore CAVIA has fewer parameters to train, but must compute higher-order gradients during training.

Since the publication of this work, several works have investigated the relationship between "fast adaptation" versus "feature extraction" in MAML (Raghu et al., 2020; Oh et al., 2021; Collins et al., 2022). The question raised in these works is whether MAML mainly learns a feature extractor (and then only adapts the last layer of the network at test time, but leaves those features mostly untouched), or if it learns to adapt quickly (defined as also changing the first few layers of the network, i.e., the feature extractor). Raghu et al. (2020) find that MAML indeed seems to mainly update the last layer when adapting to test time. They also find that when only updating the last layer of MAML (and freezing all earlier layers), leads to the same or better performance – essentially, here the "task solving" is done by the first layers, and the "task inference" is done by the last layer. This confirms our hypothesis that many Fast Adaptation tasks, like few-shot classification, mostly require task inference at test time, and that separating this from task solving is beneficial. The split can be implemented in different ways, although we argue that using a context vector has several benefits, like better interpretability, and usefulness for downstream tasks. It also proves particularly useful in the following chapters where we focus on the Reinforcement Learning problem, and introduce Bayesian reasoning over the context parameters in order for the agent to take into account its task uncertainty for exploration.

5.5 Discussion

In this chapter, we introduced *Fast **C**ontext **A**daptation **via** Meta-Learning* (CAVIA): a Meta-Learning method that separates a model into network parameters θ, and an additional input called the context vector ξ serving as a task representation. CAVIA explicitly optimises the task-independent parameters θ across tasks, such that adapting the task-specific parameters ξ at test time allows for Fast Adaptation.

Conclusion We found that explicitly separating task inference and solving using fast context adaptation has several advantages.

- Adapting a small number of input parameters (instead of the entire network) is sufficient to yield performance equivalent to or better than MAML on regression, classification, and RL problems. In fact, it prevents meta-overfitting (Mishra et al., 2018) since it only updates a few parameters at test time, as opposed to the entire network, and is robust to the inner loop learning rate. This allowed us to scale up the network size and improve performance significantly in on the Mini-Imagenet task (Table 5.3).

- CAVIA is easier to parallelise and implement compared to MAML given current auto-differentiation tools: computing the context parameters can be easily parallelised in the inner loop without requiring parameter copies, and we do not need to manually access and perform operations on the network weights and biases to set up computation graphs.

- An embedding of the task emerges in the context vector solely via backpropagation. We confirm empirically that the learned context parameters indeed match the latent task structure (Figures 5.3 and 5.8b).

Open Questions In our experiments, we focused on a wide variety of domains from Supervised Learning and Reinforcement Learning. An interesting open question is how CAVIA can be tailored more to individual settings. For example, in N-way classification, we could learn one context vector per class, instead of a joint one.

Another open questions is how to adapt when more generalisation (beyond task identification) is required at test time. Mendonca et al. (2020) learn an environment model with the same context-based approach as CAVIA, and if given an out-of-distribution task at test time, first only adapt the context ξ of the environment model, and then further continue updating θ until the model performs well. This allows for efficient adaptation when there is a distribution shift, but assumes knowledge of whether this shift occurs.

Next Steps In our RL experiments, we have so far ignored the role of *exploration*. The initial policy π_θ is responsible for collecting the data with which the gradient update is performed. Technically, Equation (5.2) in MAML and Equation (5.9) in CAVIA optimise for this. In practice however, computing the correct higher-order gradients in the RL setting is challenging and an actively researched problem (Al-Shedivat et al., 2018; Stadie et al., 2018; Rothfuss et al., 2019; Liu et al., 2021a; Vuorio et al., 2021).

Another drawback of using gradient-based adaptation is that this makes *Online Adaptation* difficult. Recall from Section 3.2.2 that in the Online Adaptation setting, we want the agent to perform well from the very first timestep it starts interacting with the environment. If a gradient-update is only done after a handful of rollouts (which is often necessary to collect a batch that is large enough to perform a sensible update), adaptation happens too slowly. Lastly, we cannot do Bayesian reasoning easily – something that is required for optimal adaptation as we have seen in Section 2.2.

In the next chapter, we therefore introduce Bayesian reasoning over the context parameters, and compute them using RNNs which allows us to update the context after every environment step.

6

Deep Bayes-Adaptive RL via Meta-Learning

Contents

For the RL experiments in the previous chapter, we assumed that the agent gets to explore freely for some time, before being evaluated. But we might not always have this luxury: in many situations we instead want the agent to perform well from the first time that it starts interacting with its environment. In this Online

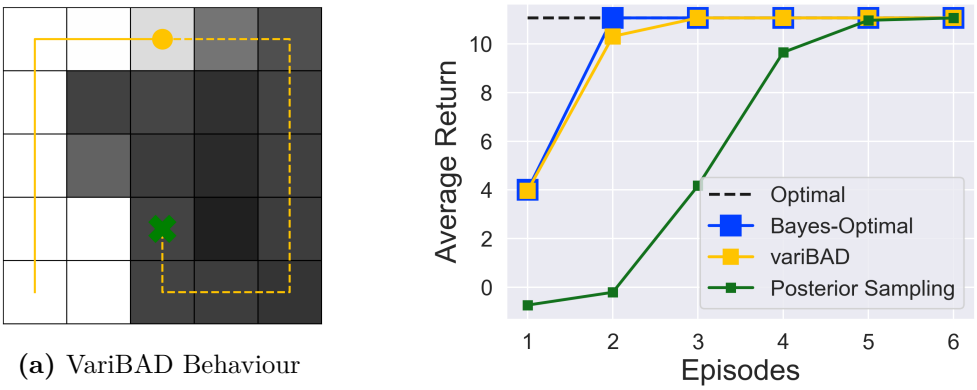
Adaptation setting (see Sec 3.2.2), the agent has to carefully balance exploration and exploitation (see Sec 2.1.6). In Section 2.2 we introduced Bayes-Adaptive MDPs (Duff et al., 2002) and how they allow us to, in principle, compute policies that optimally trade off exploration and exploitation during learning. Unfortunately, doing so is intractable for all but the smallest tasks. In this chapter, we combine ideas from Bayesian RL, approximate variational inference, and Meta-Learning to tackle this challenge. Like in the previous chapter, we use a context vector to summarise the task, but we now introduce Bayesian reasoning over this vector. This allows the agent to take task uncertainty into account for exploration.

More specifically, we propose ***vari**ational **B**ayes-**A**daptive **D**eep RL* (variBAD), a way to meta-learn to perform approximate inference on an unknown task, and incorporate task uncertainty directly during action selection. Given a distribution over MDPs $p(M)$, we represent a single MDP M using a learned, low-dimensional stochastic latent variable m and simultaneously meta-train:

1. A variational auto-encoder that can infer the posterior distribution over m in a new task, given the agent's experience, while interacting with the environment.

2. A policy that conditions on this posterior belief – a probabilistic context – over MDP embeddings, and thus learns how to trade off exploration and exploitation when selecting actions *under task uncertainty*.

We first evaluate our approach on two toy domains, the Gridworld from Figure 2.1 in Section 2.2.2, and a 2D Navigation task. These help us illustrate how variBAD maximises expected online return using the meta-learned approximate belief. A preview showing that variBAD closely matches the hard-coded Bayes-optimal agent from the Gridworld example is shown in Figure 6.1. We further evaluate variBAD on the widely used MuJoCo Meta-RL benchmarks, and show that variBAD achieves higher returns during learning compared to existing methods. Lastly, on the recently proposed challenging Meta-World ML1 benchmark, variBAD achieves state of the art performance with a large margin compared to existing methods, fully solving



(a) VariBAD Behaviour

(b) Performance Comparison

Figure 6.1: VariBAD's performance on the Gridworld from Section 2.2.2. (a) Behaviour of variBAD at meta-test time. The grey background visualises the current approximate belief (darker = higher probability of containing the goal). (b) VariBAD closely matches Bayes-optimal behaviour, and significantly outperforms posterior sampling.

two out of the three ML1 benchmark tasks for the first time. As such, variBAD opens a path to tractable approximate Bayes-optimal exploration for Deep RL.

6.1 Bayes-Adaptive Deep RL

The Online Return objective from Equation (3.2), Section 3.2.2, 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 (see Section 2.2.1) provides a principled way of formulating Bayes-optimal behaviour. However, solving BAMDPs is hopelessly intractable for most interesting problems. The main challenges are as follows.

- We often do not know how to parameterise the reward and transition model.
- The belief update (computing the posterior $p(R, T|\tau_{:t})$) is often intractable.
- Even with the correct posterior, planning in belief space is typically intractable.

In the following, we propose a method that simultaneously meta-learns the reward and transition functions, how to perform inference in an unknown MDP, and how to use the belief to maximise expected online return. Since the Bayes-Adaptive

policy is learned end-to-end with the inference framework, no planning is necessary at test time. We make minimal assumptions, resulting in a highly flexible and scalable approach to Bayes-adaptive Deep RL.

This section is organised as follows. 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.

6.1.1 Task Contexts for BAMDPs

In the typical Meta-RL 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

$$R_i(r_{t+1}|s_t, a_t, s_{t+1}) \approx R'(r_{t+1}|s_t, a_t, s_{t+1}; m_i), \tag{6.1}$$

$$T_i(s_{t+1}|s_t, a_t) \approx T'(s_{t+1}|s_t, a_t; m_i), \tag{6.2}$$

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 timestep t collected in M_i,

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

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

Recall that our goal is to *learn a belief 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.

6.1.2 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 $p_\psi(\tau_{:H^+}|a_{:H^+-1})$, parameterised by ψ, 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 timestep t, our model learning objective is thus to maximise

$$\mathbb{E}_{\rho(M,\tau_{:H^+})}\left[\log p_\psi(\tau_{:H^+}|a_{:H^+-1})\right], \tag{6.3}$$

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 Equation (6.3) directly, 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:

$$\begin{aligned}
\mathbb{E}_{\rho(M,\tau_{:H})}\left[\log p_\psi(\tau_{:H})\right] &= \mathbb{E}_\rho\left[\log \int p_\psi(\tau_{:H}, m)\frac{q_\phi(m|\tau_{:t})}{q_\phi(m|\tau_{:t})}dm\right] \\
&= \mathbb{E}_\rho\left[\log \mathbb{E}_{q_\phi(m|\tau_{:t})}\left[\frac{p_\psi(\tau_{:H}, m)}{q_\phi(m|\tau_{:t})}\right]\right] \\
&\geq \mathbb{E}_{\rho,\ q_\phi(m|\tau_{:t})}\left[\log \frac{p_\psi(\tau_{:H}, m)}{q_\phi(m|\tau_{:t})}\right] \\
&= \mathbb{E}_{\rho,\ q_\phi(m|\tau_{:t})}\left[\log p_\psi(\tau_{:H}|m) + \log p_\psi(m) - \log q_\phi(m|\tau_{:t})\right] \\
&= \mathbb{E}_\rho\left[\mathbb{E}_{q_\phi(m|\tau_{:t})}\left[\log p_\psi(\tau_{:H}|m)\right] - KL(q_\phi(m|\tau_{:t})||p_\psi(m))\right] \\
&= ELBO_t.
\end{aligned} \tag{6.4}$$

The term $\mathbb{E}_q[\log p(\tau_{:H^+}|m)]$ is referred to as the reconstruction loss, and $p_\psi(\tau_{:t}|m)$ as the decoder. The term $KL(q(m|\tau_{:t})||p_\psi(m))$ is the KL-divergence between our variational posterior q_ϕ and the prior over the embeddings $p_\psi(m)$.

Instead of using the same fixed prior for each timestep and ELBO_t, we set the prior to our previous posterior, $q_\phi(m|\tau_{:t-1})$, with initial prior $q_\phi(m) = \mathcal{N}(\mathbf{0}, \mathbb{I})$. We thus have $KL(q_\phi(m|\tau_{:t})||q_\phi(m|\tau_{t-1}))$. This is akin to a Bayesian filtering update, and works better empirically, as we confirm in Section 6.4.3. We take the gradient w.r.t. the entire term (i.e., through the current and previous approximate posterior).

The reconstruction term in the ELBO (6.4) is different than in the standard VAE (Sec 2.1.3 Kingma et al., 2014)). At timestep t, we encode the past trajectory $\tau_{:t}$ to get the current posterior $q_\phi(m|\tau_{:t})$. We then decode the *entire* trajectory $\tau_{:H^+}$ *including the future*, i.e., model $\mathbb{E}_{q_\phi}[p_\psi(\tau_{:H^+}|m)]$, which is possible because we have access to this information during meta-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. We confirm this empirically in Section 6.3. The reconstruction term $\log p_\psi(\tau_{:H^+}|m)$ factorises as

$$\begin{aligned}
&\log\, p_\psi(\tau_{:H^+}|m, a_{:H^+-1}) \\
&\quad = \log p_\psi((s_0, r_0, \ldots, s_{t-1}, r_{t-1}, s_t)|m, a_{:H^+-1}) \qquad (6.5)\\
&\quad = \log p_\psi(s_0|m) + \sum_{i=0}^{H^+-1} \left[\log p_\psi(s_{i+1}|s_i, a_i, m) + \log p_\psi(r_{i+1}|s_i, a_i, s_{i+1}, m)\right].
\end{aligned}$$

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

6.1.3 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 components:

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

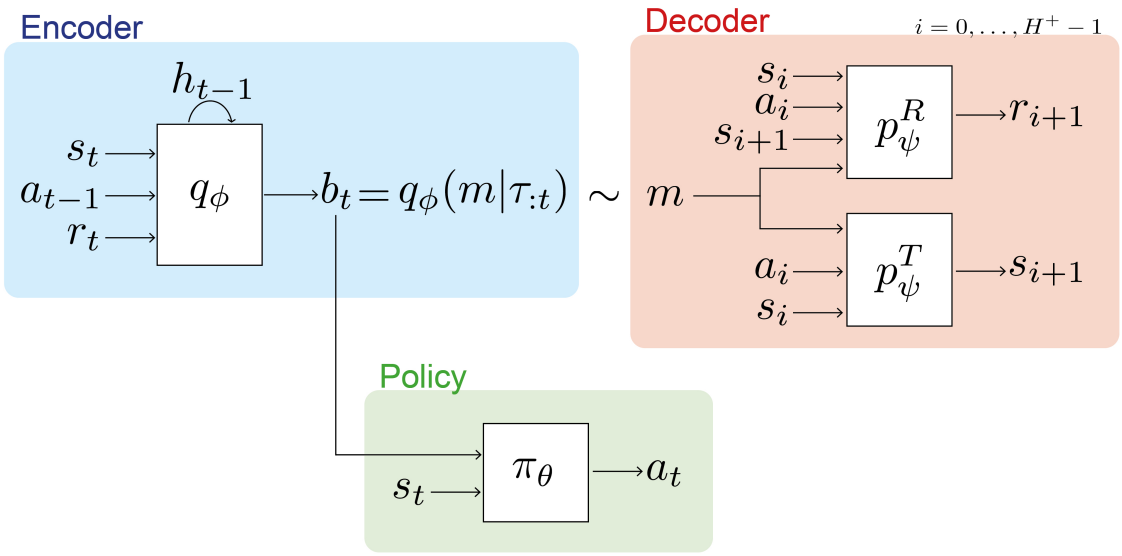


Figure 6.2: VariBAD architecture: A trajectory of states, actions and rewards is processed online using an RNN to produce the posterior over task embeddings, $q_\phi(m|\tau_{:t})$. The posterior is trained using a decoder which predicts past and future states and rewards from current states and actions. The policy conditions on the posterior in order to act in the environment and is trained using RL.

An overview of the network architecture is shown in Figure 6.2.

The policy $\pi_\theta(a_t|s_t, q_\phi(m|\tau_{:t}))$ is conditioned on the environment state s_t and the current approximate belief $b_t \equiv q_\phi(m|\tau_{:t})$ over m. This is similar to the formulation of BAMDPs introduced in Section 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/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

$$\mathcal{L}(\phi, \psi, \theta) = \mathbb{E}_{p(M)} \left[\mathcal{J}(\theta, \phi) + \lambda \sum_{t=0}^{H^+} ELBO_t(\phi, \psi) \right], \tag{6.6}$$

where $J(\theta, \phi)$ is the expected return as defined in Section 2.2.

6.1.4 Meta Training

During meta-training, we train the policy and the VAE using Equation (6.6).

In Equation (6.6), λ weights the supervised model learning objective against the RL loss. This is necessary since parameters ϕ are shared between the VAE and the policy. However, we found that backpropagating the RL loss through the encoder is typically unnecessary in practice. Not doing so speeds up training considerably, avoids the need to trade off these losses, and prevents interference between gradients of opposing losses. In ablation studies (see Appendix B) we found that backpropagating both losses through the encoder can marginally improve performance in some cases, but can be detrimental in others, and depends strongly on the relative weighting between the VAE and the RL loss. In our experiments, we therefore optimise the policy and VAE alternatingly with separate optimisers.

Meta-Training the VAE The VAE is trained by approximating expectations in Equation 6.6 using Monte Carlo samples, and using the reparameterisation trick (Kingma et al., 2014). For $t = 0$, we use the prior $q_\phi(m) = \mathcal{N}(\mathbf{0}, \mathbb{I})$. We encode past trajectories using a recurrent network, but other types of encoders could be considered like the ones used in Zaheer et al. (2017), Garnelo et al. (2018b), and Rakelly et al. (2019). The encoder outputs an approximate belief over the current task m, by predicting the mean and variance of a Gaussian distribution. The decoder is trained using samples from this approximate posterior, and by separately predicting the rewards and state transitions of the entire current trajectory (i.e., including future steps to which we have access during meta-training).

Equation (6.6) shows that the ELBO appears for all possible context lengths t. This way, variBAD can learn 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 computational efficiency if H^+ is large.

Meta-Training the Policy We meta-train the policy using proximal policy optimisation (Schulman et al., 2017, PPO), but other methods can in principle be used, including off-policy methods (see Dorfman et al. (2021)). The policy receives the approximate belief as an input in addition to the state, allowing it to meta-learn how to act approximately Bayes-optimal. This approximate belief, $q_\phi(m|\tau_{:t})$, is represented by two vectors: the mean and variance of a Gaussian distribution with diagonal covariance matrix. These vectors are predicted by the encoder, and concatenated with the environment state before being passed to the policy. In our experiments, the RL loss is not backpropagated through the encoder. We train the RL agent and the VAE using different data buffers: the policy is only trained with the most recent data since we use an on-policy algorithm in our experiments; and for the VAE we maintain a separate, larger buffer of observed trajectories.

Meta Testing At meta-test time, we roll out the policy in randomly sampled test tasks to evaluate performance, by performing forward passed through the encoder (to compute the approximate belief) and the policy (to choose how to act given the state and approximate belief). The belief is updated after every step by feeding in the new (state, action, reward) tuple, allowing the agent to adapt online. The decoder is not used at test time, and no gradient updates are performed on the encoder or policy network: the agent has learned to act approximately Bayes-optimal during meta-training.

6.2 Related Work

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 an explicit prior and belief update on the reward and transition functions, 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 these methods.

Meta-Learning Bayes-Adaptive Policies It has been shown theoretically (Ortega et al., 2019) and empirically (Mikulik et al., 2020) that meta-trained agents approximate Bayes-optimal agents on the given task distribution. A prominent model-free meta-RL approach to this problem is to use the dynamics of recurrent networks for fast adaptation (Wang et al., 2016; Duan et al., 2016). At every timestep, the network receives an auxiliary input consisting of the preceding action and reward. This allows learning within a task to happen online, entirely in the dynamics of the recurrent network, and no gradient adaptation is needed at meta-test time. If we remove the decoder (Fig 6.2) and the VAE objective (Eq (6.3)), 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 and rewards (which acts as an auxiliary loss to encode the task in latent space and deduce information about unseen states). While model-free Meta-Learning methods (Wang et al., 2016; Duan et al., 2016) can meta-learn approximately Bayes-optimal policies (Ortega et al., 2019; Mikulik et al., 2020), we show empirically that they do not scale as well in terms of performance as variBAD. In addition, variBAD provides explicit belief representations, which can be used in downstream tasks or for auxiliary tasks.

Closely related to our approach is the work of Humplik et al. (2019). Like variBAD, their algorithm conditions the policy on a meta-learned approximate belief. This approximate belief is learned using privileged information during meta-training such as a task description. In comparison, variBAD meta-learns to represent the belief in an unsupervised way, and does not rely on privileged task information.

Building on the conference version of this work (Zintgraf et al., 2020), Dorfman et al. (2021) propose a method to meta-learn Bayes-adaptive policies from offline data, and additionally demonstrate good results when using an off-policy variant of variBAD. They use variBAD with the off-policy algorithms DQN (Mnih et al., 2015) (for discrete control) and SAC (Haarnoja et al., 2018) (for continuous control), demonstrating that variBAD can be combined with different RL algorithms.

Other Meta Reinforcement Learning Settings Another popular approach to Meta-RL is to learn an initialisation of the model, such that at test time, only a few gradient steps are necessary to achieve good performance (Sec 4.1; Finn et al., 2017a; Nichol et al., 2018; Stadie et al., 2018; Rothfuss et al., 2019). These typically consist of a feed-forward policy and are therefore relatively lightweight. 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 to 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 separate exploration (before gradient adaptation) and exploitation (after gradient adaptation) at test time by design, making them less sample efficient.

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 method for Bayesian planning for tractable, sample-based approximately 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 for exploration are, e.g., certain reward bonuses (Kolter et al., 2009; Sorg et al., 2012) and methods based on optimism in the face of uncertainty (Kearns et al., 2002; Brafman et al., 2002). Non-Bayesian methods for exploration are often used in practice, such as other exploration bonuses (e.g., via state-visitation counts) or using uninformed sampling of actions (e.g., ϵ-greedy action selection). Such methods are prone to wasteful exploration that does not help maximise expected reward.

Contextual MDPs Related to BAMDPs are *contextual MDPs*, where the task description is referred to as a context, on which the environment dynamics and rewards depend (Hallak et al., 2015; Jiang et al., 2017; Dann et al., 2019; Modi et al., 2019). Research in this area is often concerned with developing tight bounds by putting assumptions on the context, such as having a small known number of contexts, or that there is a linear relationship between the contexts and dynamics/rewards. Similarly, the framework of hidden parameter (HiP-) MDPs assumes that there is a set of low-dimensional latent factors which define a family of related dynamical systems (with shared reward structure). The assumptions in this line of work are similar to the assumption we make in Equations (6.1) and (6.2) (Doshi-Velez et al., 2016; Killian et al., 2017; Yao et al., 2018). These methods however do not directly learn Bayes-optimal behaviour; instead, they allow for a longer training period in new environments to infer the latents and train the policy.

Skill and Task Embeddings Learning (variational) task or skill embeddings for meta or transfer reinforcement learning is used in a variety of approaches. Hausman et al. (2018) use approximate variational inference to learn an embedding space of skills (using a different lower bound than variBAD). At test time a new embedder is learned that interpolates between 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 are controlled by a higher-level policy, framed within a hierarchical RL setting. This embedding is learned using a VAE to encode state trajectories and decode states and actions. Similar to variBAD, Zhang et al. (2018) use learned dynamics and reward modules to learn a latent representation which the policy conditions. They 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 where the encoder is updated at test time using gradient descent, 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 in Equation (6.4) 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.

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., how to do a posterior update. Apart from (RL) methods mentioned, related work in this direction can be found, a.o., in Garnelo et al. (2018b), Gordon et al. (2019), and Choi et al. (2019). In contrast to these supervised settings, variBAD's inference procedure is tailored to Bayes-optimal RL, a sequential setting where the objective of inference is to model beliefs over MDPs.

Partially Observable Markov Decision Processes (POMDPs) Several deep learning approaches to model-free Reinforcement Learning (Igl et al., 2018) and model learning for planning (Tschiatschek et al., 2018) in partially observable Markov decision processes have recently been proposed and use approximate variational inference methods. VariBAD by contrast focuses on BAMDPs (Martin, 1967; Duff et al., 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 the one by Igl et al. (2018) which use filtering to track the changing hidden state. While we use the power of deep approximate variational inference, other approaches for BAMDPs often use more accurate but less scalable methods, e.g., Lee et al. (2019a) discretise the latent distribution and use Bayesian filtering for the posterior update.

6.3 Empirical Evaluation

In this section we first investigate the properties of variBAD on the didactic Gridworld domain from Figure 2.1. These results show that variBAD performs *structured* and *online* exploration as it infers the task. We then evaluate variBAD on several more challenging domains: a sparse 2D navigation task, a range of tasks from the MuJoCo benchmark, and the Meta-World ML1 benchmark. On these, variBAD achieves state of the art performance. In Section 6.4, we perform ablation studies to motivate our design choices, and test how robust variBAD is to the size of the latent space.

The two main baselines we consider are RL^2 (Duan et al., 2016; Wang et al., 2016) and PEARL (Rakelly et al., 2019). RL^2 can be seen as a model-free version of variBAD. As discussed in Section 4.2, RL^2 can learn to act approximately Bayes-optimal by performing task inference in the recurrent state of the RNN. RL^2 only consists of an encoder and a policy, and trains both end-to-end using an RL loss only. PEARL is akin to posterior sampling. Recall from Section 2.2.2 that posterior sampling is a sub-optimal exploration strategy in theory compared to Bayes-optimal exploration (see Figure 2.1), but has the advantage that computing the optimal policy for a single MDP (sample from the approximate posterior) is more tractable than doing so in a BAMDP.

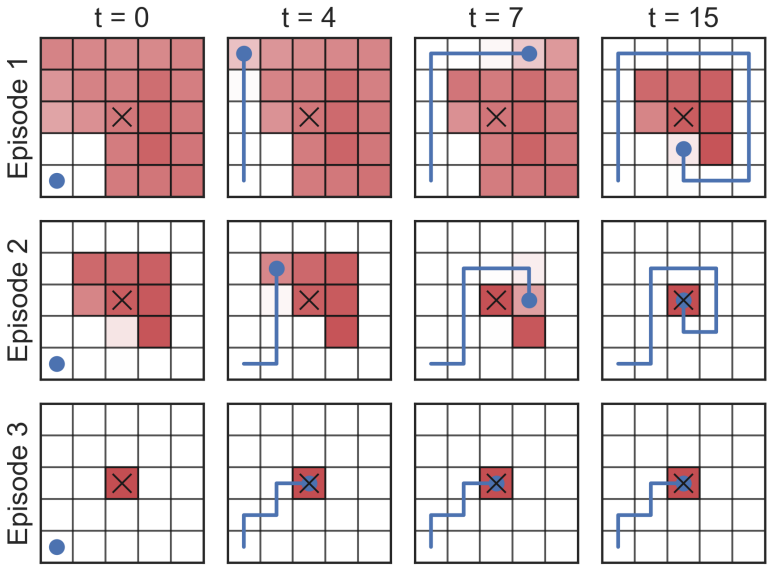
For experimental details, hyperparameters, and additional results, see Appendix B. Source code is available at `https://github.com/lmzintgraf/varibad`.

6.3.1 Gridworld

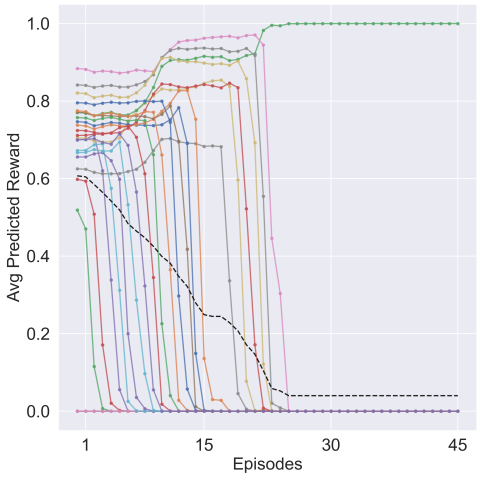
To gain insight into variBAD's properties, we start with the didactic Gridworld environment from Figure 2.1a. The task is to reach a goal in a 5×5 grid. The goal can be anywhere except around the starting cell, which is at the bottom left, and is selected uniformly at random. The goal is unobserved by the agent, inducing task uncertainty and necessitating exploration. Actions are: *up, right, down, left, stay*, and are executed deterministically. The horizon within the MDP is $H = 15$, and we set the horizon in the BAMDP to $H^+ = 4 \times H = 60$, i.e., we train our agent to maximise performance for 4 MDP episodes. After 15 steps the agent is reset to its starting position (but the goal stays the same). The agent gets a sparse reward signal: -0.1 on non-goal cells, and $+1$ on the goal cell (repeatedly if the agent stays on the goal). We use a latent dimensionality of 5 (see Sec 6.4.1 for how latent size affects performance). The Bayes-optimal strategy is to explore until the goal is found (Fig 2.1b), and stay at the goal or return to it when reset to the initial position.

Figure 6.3 illustrates how variBAD behaves at test time with deterministic actions (i.e., all exploration is done by the deterministic policy, not via sampling). In Figure 6.3a we see how the agent interacts with the environment over the course of three episodes (with a fixed goal), with the red background visualising the approximate posterior belief, 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 high expected rewards for unvisited cells. It explores the remaining cells until it finds the goal, at which point its posterior collapses to the correct task. As the agent gathers more data, more and more cells are excluded ($p(rew{=}1) = 0$, white cells), until eventually the agent finds the goal.

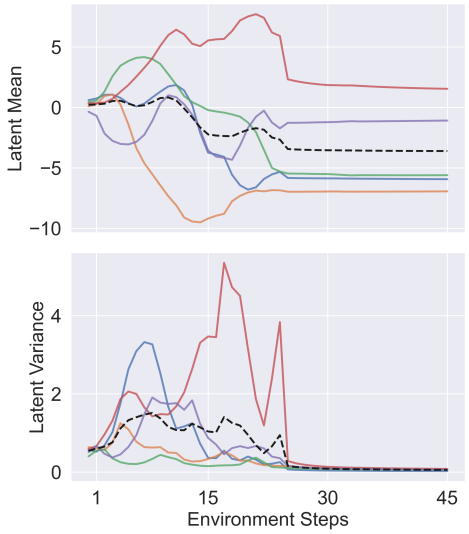
Figure 6.3b show the reward predictions: each line represents a grid cell and its value the probability of receiving a reward at that cell. As the agent gathers more data, more cells are excluded ($p(rew{=}1) = 0$). When the agent finds the goal, the predictions correspond to the true reward function. Figure 6.3c visualises the latent space. Once the agent finds the goal, the posterior concentrates: the variance drops close to zero, and the mean converges to a fixed value.



(a) Example Rollout



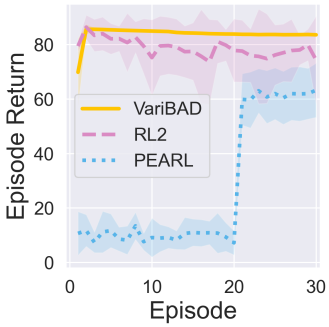
(b) Reward Predictions



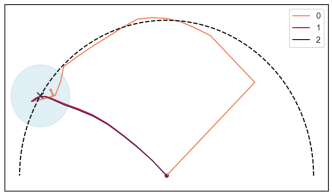
(c) Latent Space

Figure 6.3: Behaviour of variBAD in the Gridworld environment. **(a)** Hand-picked but representative example test rollout. The red 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 5-dimensional latent space; each line is one latent dimension, the black line is the average.

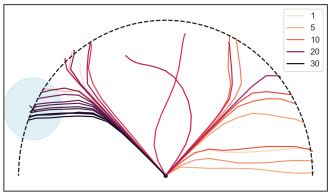
The behaviour of variBAD closely matches that of the Bayes-optimal policy (shown in Figure 2.1). Our results on this Gridworld indicate that variBAD is an effective way to approximate Bayes-optimal control, and has the additional benefit of giving insight into the task belief of the policy.



(a) Meta-Test Performance



(b) VariBAD example rollout



(c) PEARL example rollout

Figure 6.4: Meta-test performance on the Sparse 2D Navigation environment. (a): Performance at meta-test time (averaged over the task distribution). (b) and (c): Example rollouts of meta-trained variBAD (b) and PEARL (c) agents. PEARL does not optimize for optimal exploration, and thus it requires many more episodes to find the goal. On the other hand, variBAD optimises for optimal exploration, efficiently covering possible goal locations, and is able to quickly find the goal.

6.3.2 Sparse 2D Navigation

We evaluate on a Point Robot 2D navigation task used by Gupta et al. (2018), Rakelly et al. (2019), and Humplik et al. (2019), to further illustrate how variBAD performs online adaptation. The agent must navigate to an unknown goal sampled along the border of a semicircle of radius 1.0, and receives a reward relative to its proximity to the goal when it is within a goal radius of 0.2. Since this is a sparse reward environment, the Bayes-optimal exploration strategy includes walking along the semi-circle until the goal is found.

Figure 6.4 shows the average performance of PEARL, RL^2, and variBAD at test time, when rolling out the agent for 30 episodes in a single task. VariBAD adapts much faster to the task compared to PEARL. RL^2 also adapts rapidly but is less stable compared to variBAD when rolled out for a much longer time than during training: both variBAD and RL^2 were trained to perform well over three consecutive episodes.

To shed light on the performance difference between variBAD and PEARL, Figures 6.4b and 6.4c visualise representative example rollouts for meta-trained variBAD and PEARL agents. We picked examples where the target goals are at the end of the semi-circle, which we found are most difficult for the agent.

PEARL performs posterior sampling for exploration which means it is restricted to a fixed hypothesised goal position during each rollout. On the other hand, variBAD (Figure 6.4b) strategically explores the space within an episode to find the goal, which is more efficient. Once the goal is found, both variBAD and PEARL are able to quickly return to it.

The two toy experiments, Gridworld and PointRobot, illustrate how variBAD makes decisions: it adapts to the task *online* while updating the approximate belief, which allows it to rapidly adapt to new tasks. In the following sections, we test variBAD on more challenging meta-RL benchmarks, MuJoCo and Meta-World ML1.

6.3.3 MuJoCo Continuous Control Meta-Learning Tasks

We show that variBAD can scale to more complex Meta-Learning settings by employing it on MuJoCo (Todorov et al., 2012b) locomotion tasks commonly used in the meta-RL literature.[1] We consider the Ant-Dir, HalfCheetahDir, and Humanoid environments, where the agent has to run either forwards or backwards (i.e., there are only two tasks); the HalfCheetahVel environment where the agent has to run at different velocities; the Ant-Goal environment where the agent has to navigate to an initially unknown goal position; and the Walker environment where the system parameters are randomised. The rewards in these environments are dense, so that in principle the agent only needs a few exploratory actions to infer the task by observing the rewards it receives.

VariBAD and RL^2 were trained to maximise performance within two episodes (mainly so that we can roll them out for multiple episodes at test time; we are primarily interested in their adaptation behaviour within the first episode). PEARL (Rakelly et al., 2019), was trained using the open sourced code[2], to maximise performance after 3-5 episodes, depending on the environment. E-MAML (Stadie et al., 2018) and ProMP (Rothfuss et al., 2019) were trained using the open sourced

[1]The MuJoCo environments are taken from `https://github.com/katerakelly/oyster` .

[2]The implementation which we used for our PEARL experiments was published by Rakelly et al. (2019) and can be found at `https://github.com/katerakelly/oyster` .

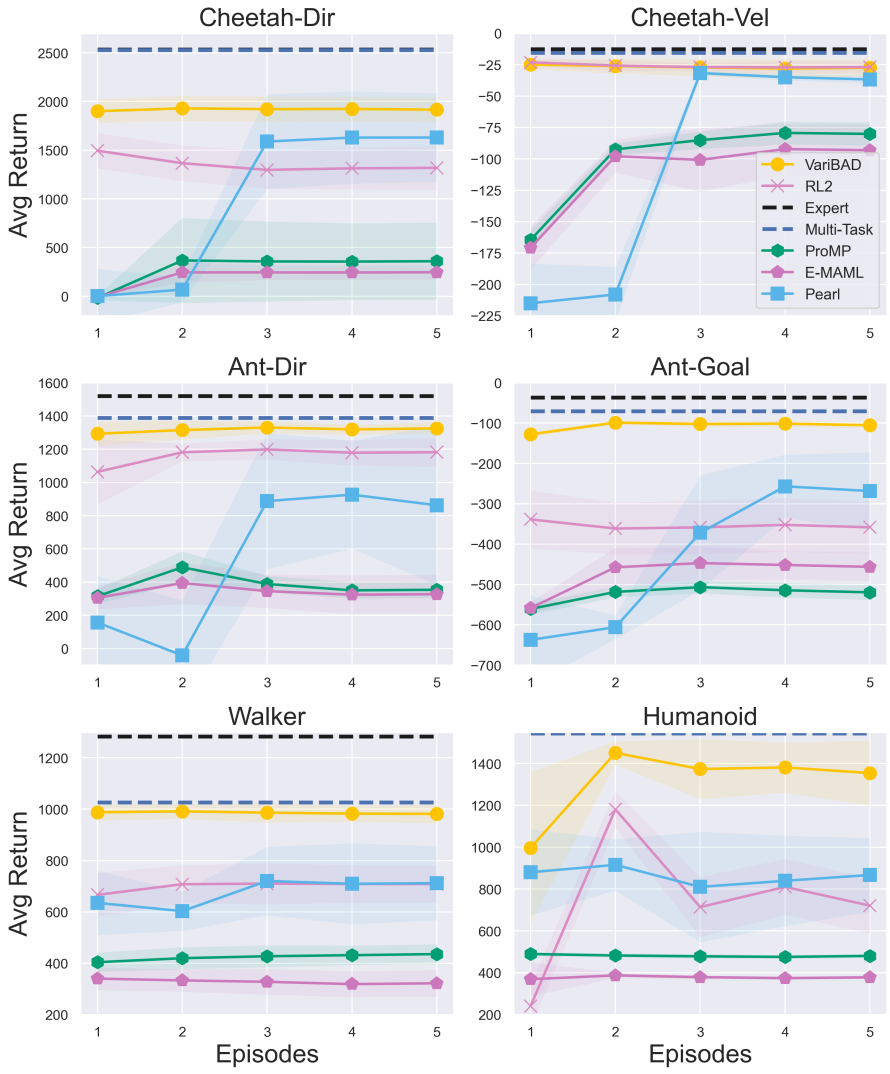


Figure 6.5: Average test performance on six different MuJoCo environments, trained separately with 10 seeds per MuJoCo environment per method. The meta-trained policies are rolled out for 5 episodes to show how they adapt to the task. Values shown are averages across tasks (95% confidence intervals shaded). Being an online adaptation method, variBAD adapts within the first episode. It outperforms other methods, even when these are given longer than one episode to adapt, and even though the first episode includes exploratory actions. RL^2 is also an online adaptation method and can adapt to the task within the first episode. On most environments, variBAD outperforms RL^2 (Wang et al., 2016; Duan et al., 2016) significantly. The other methods, PEARL (Rakelly et al., 2019), E-MAML (Stadie et al., 2018), and ProMP (Rothfuss et al., 2019), need at least one episode to adapt by design.

code by Rothfuss et al. (2019)[3], to maximise performance after 1 gradient step on 10-20 rollouts. To get an approximate upper bound on performance, we train a multi-task agent which conditions on a task descriptor, and an ensemble of expert agents (one per task) whose performances are averaged, using PPO.

Figure 6.5 shows the average performance at test time across different tasks. While we show the return of the agent during the first five rollouts, we emphasise that our primary interest lies in the agent's performance during the first episode, *while learning* about the environment, which tells us how well the agent can trade off exploration and exploitation.

Only variBAD and RL^2 are able to adapt to the task at hand within a single episode. VariBAD outperforms RL^2 in all environments except HalfCheetahVel where they are on par, and is close to the multi-task agent's performance in several environments. We generally found that learning with RL^2 is slower and less stable (see learning curves and runtime comparisons in Appendix B.2). We hypothesise that this is because the reinforcement learning loss is backpropagated through an RNN. In variBAD on the other hand, we train the encoder RNN with a supervised loss only. Even though the first rollout includes exploratory steps, this matches the optimal multitask policy (which is conditioned on the true task description) up to a small margin.

The 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 the first episode. 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. We evaluated E-MAML and ProMP by performing gradient steps after every episode; however, they are typically updated after collecting data for 20 episodes.

Method	Episode	Reach	Push	Pick-Place
MAML*	10	48	74	12
PEARL*	10	38	71	28
RL^{2*}	10	45	87	24
variBAD	**1**	**100**	**100**	29 (6/20 seeds)
variBAD	2	100	100	29

Table 6.1: Meta-test success rates on the ML1 Meta-World benchmark (v1). *Results taken from Yu et al. (2019). VariBAD was trained to maximise the expected return of 2 episodes (20 seeds per setting). The first episodes often *include exploratory actions*, yet variBAD has higher success rate in episode 1 than existing methods. For the Pick-Place results, in brackets we report the number of seeds that learned something.

6.3.4 Meta-World

Finally, we evaluate variBAD on the challenging Meta-World ML1 benchmark (v1; Yu et al., 2019), which has emerged as a key challenge for the Meta-Learning community. In the Meta-World environment, a robot arm has to perform different tasks like pushing objects to an (initially unknown) target location. There are three variants of the ML1 benchmark: Reach (the robot has to reach different goal positions), Push (the robot arm has to push objects to different goal positions), and Pick-Place (the robot arm has to pick up an object, and place it near a target goal).

Table 6.1 shows the results for variBAD and several baselines on the ML1 benchmark. VariBAD achieves state of the art results. On Reach and Push, variBAD outperforms the previous state of the art results by a significant margin, and is the first to fully solve these tasks. On the harder task Pick & Place, variBAD performs on par with PEARL. Though the benchmark allows for up to 10 episodes for adaptation, we train variBAD to optimise expected online return within two episodes. As these results show, variBAD can adapt rapidly even within the first episode, which includes exploratory actions. On the Pick-Place task, variBAD either learns to solve the task (6 seeds), or it does not meta-learn at all (14 seeds).

[3]The implementation which we used for our E-MAML and ProMP experiments was published by Rothfuss et al. (2019) and can be found at `https://github.com/jonasrothfuss/ProMP` .

6.4 Empirical Analysis

In this section, we examine how variBAD's performance varies with latent size. We further study the impact of different VAE loss formulations on the performance and approximate beliefs to shed light on our design choices.

6.4.1 Belief Dimensionality

In this section, we examine how the size of the latent belief impacts performance in Figure 6.6, for the Gridworld and Ant-Goal tasks. As expected, if the latent dimensionality is too small, then not all relevant information can be retained and the policy is unable to adapt to different tasks. This only happens when using a dimensionality of size 1 or 2, and any choice larger than this leads to decent performance. Interestingly, very large parameterisations (1000 for Gridworld and 300 for AntGoal which was the maximum we could fit into the memory of a single GPU) have a comparatively minute impact despite artificially increasing the size of the state space on which the policy acts. In practice this means that as long as we do not underparameterise the latent dimension, we achieve good performance.

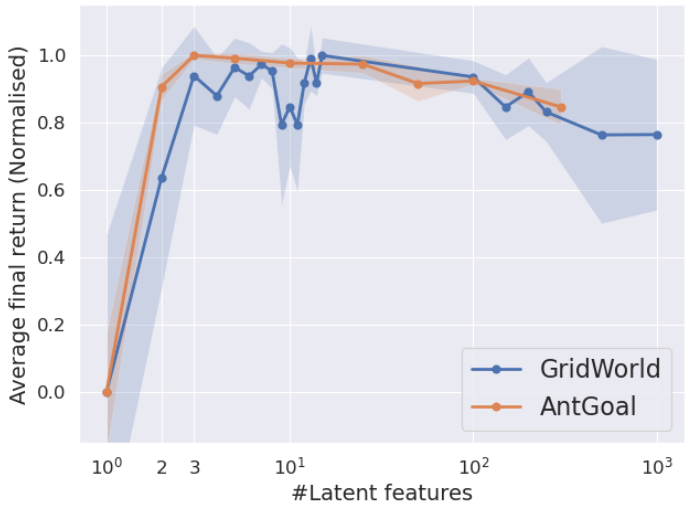


Figure 6.6: Normalised performance for different number of latent features, for the Gridworld and Ant environment. As long as the latent is not underparameterised, variBAD achieves good performance. Only when vastly overparameterising the latent we see a small performance decrease.

6.4.2 Modelling Horizon

When formulating our objective in Equation (6.3), we argue that at any time t in its trajectory, an agent should be able to model entire trajectories. In Equation (6.5) we see that this amounts to reconstructing transitions and rewards from the past ($\leq t$) and the future ($> t$). The intuition is that the VAE has to represent the task information in its belief, which encompasses a sufficient statistic about transitions the agent has already observed, and a belief about transitions it can observe in the future. To analyse this choice empirically, we test how different "modelling horizons" affect performance and the learned beliefs: modelling only the past ($\leq t$), only the future ($> t$), or only one step into the future ($t + 1$).

Figure 6.7 shows the resulting learning curves for the Gridworld, Mujoco AntGoal, and PointRobot tasks. The traditional VAE approach of reconstructing only the embedded part of the trajectory (past) tends to produce latent codes not sufficiently informative to predict future transitions and therefore leads to suboptimal performance. Similarly, targeting only the subsequent transition (next) does not encode sufficient information to reliably inform the policy. While decoding only the entire remainder of the observed training trajectories (future) performs well, we find that this has an undesired effect on the learned beliefs, discussed below.

Figure 6.8 visualises the approximate belief for different decoding targets in the Gridworld environment. We do so by plotting the reward predictions of the decoder from the VAE latent for each cell in the grid. Decoding only the past or only the next step leads to embeddings that predict no or only spurious rewards until the goal has been found (Figures 6.8b and 6.8d). Only decoding the future enables learning about the rewards prior to actually encountering them, but leads to spurious predictions for visited states (Figure 6.8c): predictions of non-zero rewards at visited non-goal states are not penalised, as these are unlikely to be revisited. These artefacts are cleared up by decoding full trajectories (Figure 6.8a), which is the default setting we chose for the variBAD objective.

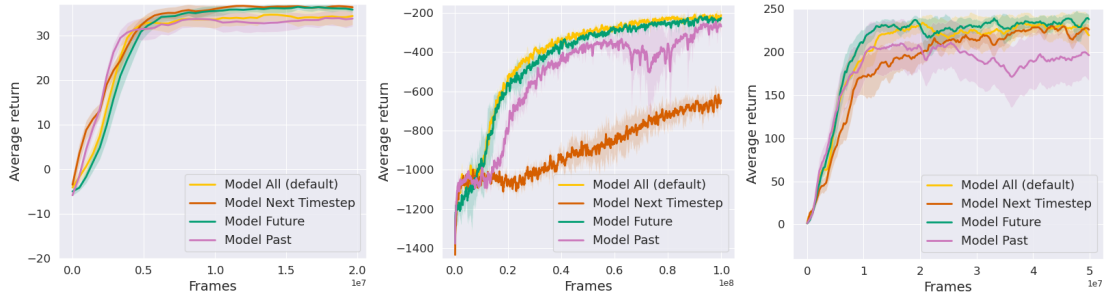
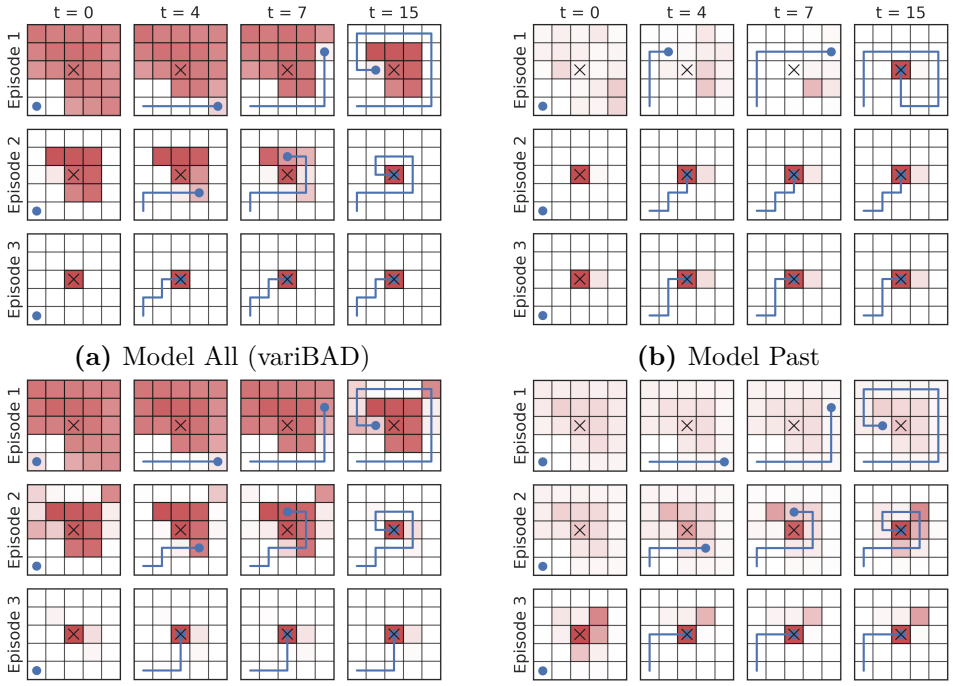


Figure 6.7: Performance of VAE decoding targets for (from left): Gridworld, AntGoal, and Pointrobot (15/5/20 seeds). VariBAD's default settings (modelling the past and future) perform well. Other choices either underperform (modelling only the past or only the next step), or have undesired effects on the beliefs (Fig 6.8c).



(a) Model All (variBAD) **(b)** Model Past

(c) Model Future **(d)** Model Next Step

Figure 6.8: Belief visualisation in the Gridworld for different decoding targets. Compared to the default variBAD objective which models the past *and* future (a), the approximate belief is less accurate for other decoding targets. In (b) the agent only decodes the past. It assigns low probability to all cells, when the goal is not found yet. This minimises the loss: for seen cells it correctly predicts that they do not contain the goal. For unseen cells it incorrectly predicts the same, but this is not penalised in the loss. In (c) the agent only decodes the future. For unseen cells it correctly predicts that they could contain the goal. For seen cells the loss does not incentivise it to predict the correct thing and predictions are noisy. We believe some predictions are zero since there is a chance of re-visiting states. In (d) the agent only predicts the immediate-next step, and assigns non-zero probability to these, when it has not found the goal. All methods (a)-(d) learn to correctly predict the goal position with high certainty once they have found it.

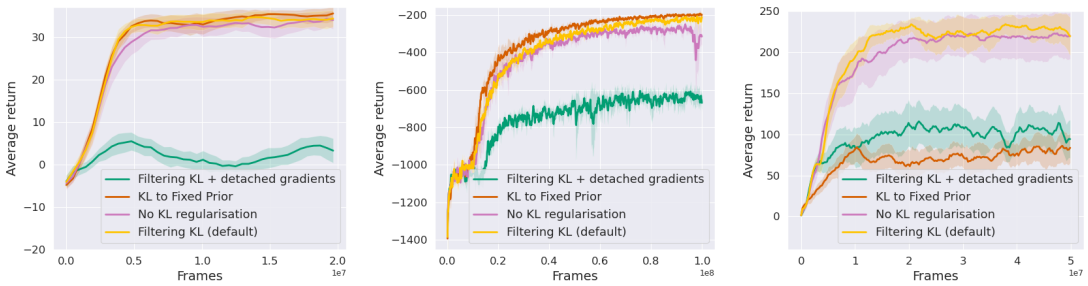
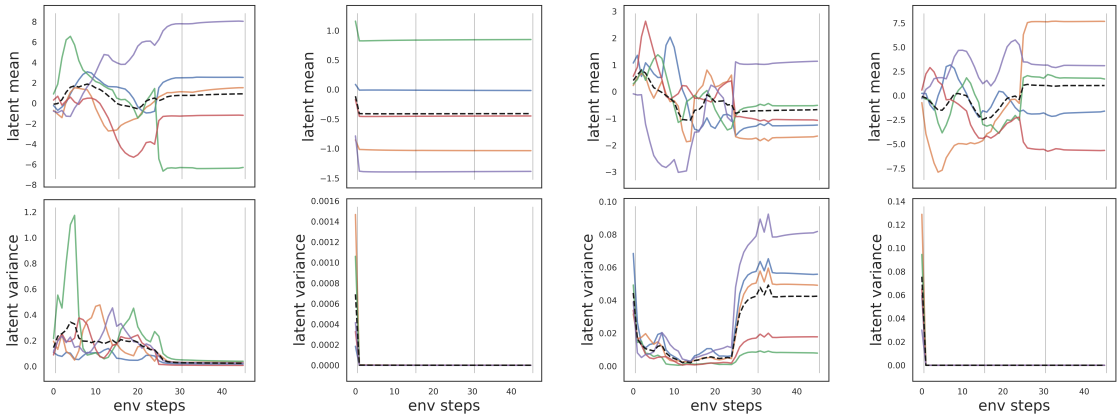


Figure 6.9: Performance of different KL regularisations in the VAE. Domains from left to right are Gridworld (15 seeds), Ant-Goal (5 seeds) and Pointrobot (20 seeds). Using the filtering KL objective performs well consistently across these environments. Using the KL to a fixed prior works well except in the PointRobot environment. We hypothesise this is due to the sparsity of the environment, and the fact that "excluding" certain regions without rewards becomes more challenging when the prior is fixed.

6.4.3 KL Regularisation

The second component of the ELBO, the KL term between the posterior and prior, acts as a regulariser and prior for the latent codes. In the variBAD objective, the prior is set to the previous approximate posterior, i.e., the KL term is defined as $KL(q_\phi(m|\tau_{:t})||q_\phi(m|\tau_{t-1}))$, which is akin to Bayesian filtering. The gradient is taken w.r.t. both terms in the KL. In this subsection, we empirically motivate this choice, by comparing to using a fixed standard Normal distribution as a prior, $\mathcal{N}(\mathbf{0}, \mathbb{I})$, across all times steps ("fixed prior"), detaching the gradient of the approximate previous posterior, or not doing KL regularisation.

The learning curves are shown in Figure 6.9 for the Gridworld, AntGoal, and Pointrobot environment, and visualisations of the learned approximate beliefs in the Gridworld are shown in Figure 6.10. Using a fixed prior is not sufficiently flexible to allow the learning of good latent codes in all cases: the performance is significantly worse in PointRobot (Fig 6.9, right), and when rolling out the meta-learned policy the variance increases sharply after the agent finds the goal in the Gridworld (Fig 6.10c). This is undesirable—the variance should collapse once the agent is certain about the task. By forcing the VAE to adhere to a fixed prior we artificially increase uncertainty as the embedding is encouraged to regress towards the prior even once the goal has been identified.



(a) Filtering KL **(b)** Detach Gradient **(c)** Fixed Prior **(d)** No KL

Figure 6.10: Visualisation of latent estimates for different decoding targets, for a representative Gridworld rollout. The agent finds the goal after ~ 25 steps. When using a filtering KL (variBAD's default setting), the variance collapses once the goal is found (a). When using a fixed Gaussian prior (c), the posterior behaves unexpectedly: the variance *increases* once the goal is found. Detaching the gradient of the previous belief (b) or no using KL regularisation (d) sets the variance to zero.

When training with the default variBAD objective we get reasonable variance estimates that decay with progressing exploration and collapse when the goal is found (Fig 6.10a). Using no KL regularisation at all slightly reduces performance (Fig 6.9) and a total collapse in variance (Fig 6.10c). A lack of regularisation allows the VAE to encode observed trajectories as point masses with no variance, indicating it overfits by producing a unique code for each of them. Detaching the gradient on the previous approximate posterior significantly harms performance in all cases (Fig 6.9) and yields an inflexible latent space (Fig 6.10b).

6.5 Discussion

Conclusion We presented variBAD, a novel Meta-RL method to learn approximately Bayes-optimal behaviour, which uses Meta-Learning to exploit knowledge obtained in related tasks and perform approximate inference in unknown environments. In a didactic Gridworld environment, our agent closely matches Bayes-optimal behaviour, and in more challenging MuJoCo tasks, variBAD outperforms existing methods in terms of achieved reward during a single episode. On the recently proposed Meta-World ML1 benchmark, variBAD outperforms existing

methods by a large margin and fully solves two out of the three benchmark tasks for the first time. In summary, we believe variBAD opens a path to tractable approximate Bayes-optimal exploration for Deep Reinforcement Learning.

Open Questions There are several interesting directions of future work based on variBAD. For example, we currently do not use the decoder at test time, but it could be used for online planning, or to get a sense for how wrong the predictions are (which might indicate we are out of distribution, and further training is necessary). Another exciting direction for future research is considering settings where the training and test distribution of environments are different. Generalising to out-of-distribution tasks poses additional challenges and in particular for variBAD two problems are likely to arise: the inference procedure will be wrong (the prior and/or posterior update) and the policy will not be able to interpret a changed posterior. In this case, further training of both the encoder/decoder might be necessary, together with updates to the policy and/or explicit planning. We discuss these directions in more detail with some empirical examples in the concluding part of this thesis, Chapter 9.

In the next chapter, we use the lessons learned in variBAD to look a different set of problems: Multi-Agent Reinforcement Learning. Here, the "task" does not differ in the reward and transition function, but in the other agents that are present in the environment. This can be more challenging because the agent's own actions influence the other agents' future actions, meaning it has to *track* the other agents' *current* internal state as well. This means we are in a more general POMDP, and we have to model the other agents differently.

7

Deep Interactive Bayesian RL via Meta-Learning

Contents

In the previous chapters, we considered Fast Adaptation settings where an agent is faced with a new environment or task, and has to rapidly adapt to this. Throughout, we assumed that the agent is alone in its environment, i.e., no other artificial or human agents are present. In most real world settings however, there will likely also be other artificial or human agents in the environment. When multiple agents interact in the same environment, they influence each other through their actions:

either directly by cooperating, communicating, or competing; or indirectly by affecting the state of the world. For example, playing a game of soccer with a new team requires learning about each player's role and coordinating actions; driving a car through busy traffic requires anticipating other drivers' moves and reacting to these appropriately; and successfully teaching a complex subject to a student requires adjusting the teaching method to their learning style.

A desirable capability of artificial agents is therefore the ability to adapt to other (human or artificial) in an ad-hoc way. Contrary to our previous set-up, we are now in an *interactive* setting, where our agent's actions influence how the other agents react. This makes belief modelling more challenging. The framework for computing Bayes-optimal such agents in a multi-agent setting is called *Interactive Bayesian Reinforcement Learning* (IBRL; Chalkiadakis et al., 2003; Hoang et al., 2013). This approach requires maintaining a belief over the other agents' strategies, and computing the optimal action given that belief. Similar to the BAMDP framework (see Section 2.2.1), the IBRL framework is incredibly powerful, since its solution – a Bayes-optimal policy – yields agents that adapt in the best possible way to other agents. Unfortunately, computing this solution is intractable for most interesting problems, and existing approximation methods are restricted to small environments and simple agent models (Chalkiadakis, 2007; Hoang, 2014).

In this chapter, we argue that the IBRL framework is a useful proxy for learning adaptive policies, and propose ***Me****ta* ***L****earning* ***I****nteractive* ***B****ayesian* ***A****gents* (MeLIBA). We leverage recent advances in agent modelling, approximate variational inference, and Meta-Learning, to compute approximately Bayes-optimal agents for multi-agent settings in a general and tractable way.

Specifically, our model consists of two parts that are jointly meta-trained on a given prior distribution over other agents: (1) a belief inference network (with a novel architecture), and (2) a policy that conditions on the approximate belief (as before). There is a rich history of modelling agents in the literature (Hula et al., 2015; Oliehoek et al., 2016; Albrecht et al., 2018, a.o.), and we build on ideas from Rabinowitz et al. (2018) who use a hierarchical latent structure that separates

permanent and temporal components of agent models. To maintain *beliefs* over these, we use a VAE (Sec 2.1.3; Kingma et al., 2014) for sequential data (Chung et al., 2015), combined with a hierarchical latent structure (Zhao et al., 2017). The policy conditions on the latent variables of this VAE, allowing it to adapt to others online.

Empirically, we demonstrate that explicitly learning and conditioning on an approximate belief over other agents' strategies can substantially improve performance in multi-agent settings, compared to relying on samples, or using a model-free policy with access to memory. Our results indicate that factoring our model according to environment and the other agents' structure is beneficial when learning to interact with different agents, and that we can successfully learn adaptive policies.

7.1 Background

This section presents the problem setting, main assumptions, and IBRL framework.

7.1.1 Environment

We consider a Markov game (Shapley, 1953; Van Der Wal, 1981) with N agents, defined as a tuple $M = (N, \mathcal{S}, \mathcal{U}, R, T, H)$, where $\mathcal{U} = \mathcal{U}_1 \times \ldots, \times \mathcal{U}_N$ is the joint action space, $\mathcal{S}$ the shared state space, $R(r_{t+1}|s_t, u_t, s_{t+1})$ a reward function where $r \in \mathbb{R}^N$ and $u_t = (u_1, \ldots, u_N)$ are the rewards and joint actions, and $T(s_{t+1}|s_t, u_t)$ are the state transition probabilities (for brevity this includes the initial state distribution $T_0(s_0)$). At each timestep t in state $s_t \in \mathcal{S}$, each agent i takes an action $u_t^{(i)} \in \mathcal{U}$. The actions of all agents are executed simultaneously, and each agent receives its own reward.

Each agent i chooses actions according to its policy $a^{(i)} : \mathcal{S} \times \mathcal{U} \times \mathbb{R} \times \ldots \times \mathcal{S} \to \mathcal{U}_i$ which maps interaction histories to action probabilities, $a^{(i)}(u_t|\tau_{:t}^{(i)})$. Here we allow an agent to condition its actions not only on the current state, but on the history $\tau_{:t}^{(i)} = (s_0, u_0, r_1^{(i)}, \ldots, s_t)$ with full observability of states and other agents' actions, and its private reward.

7.1.2 Objective

We consider the task of learning a policy $a^{(1)}$ for agent 1, with no control over the other agents' policies $a^{(-1)} = a^{(2)}, \ldots, a^{(N)}$. Instead, we have a prior distribution over those policies $P_A^{N-1}(a^{(-1)}) = \prod_{i=2}^{N} P_A(a^{(i)})$. The objective of our agent is to perform well in expectation when faced with a random sample of policies from this distribution, within a single game:

$$\max_{a^{(1)}} \quad \mathbb{E}_{P_A^{N-1}} \left[\mathbb{E}_{P_R^a, P_T^a, a} \left[\sum_{t=0}^{H^+-1} r_{t+1}^{(1)} \right] \right]. \tag{7.1}$$

Since the other agents' policies are unknown to our agent, it has to simultaneously learn about the other agents, while adapting its behaviour accordingly in order to maximise its own online return within the given horizon H^+.[1]

7.1.3 Interactive Bayesian RL

An agent that maximises Equation (7.1) acts optimally under uncertainty and is called *Bayes-optimal* (see Section 2.2.1), assuming we treat the distribution P_A^{N-1} over other agents as our epistemic belief about the world. It gathers information about the other agents *if and only if* this helps accumulate more rewards in expectation in the future *within* the given horizon H^+, and adapts its strategy conditioned on its belief about the other agents.

In principle, a Bayes-optimal policy can be computed using the framework of Interactive Bayesian Reinforcement Learning (IBRL, Chalkiadakis et al., 2003; Hoang et al., 2013). Given a prior belief over the other agents' strategies P_A^{N-1}, the objective in IBRL is to maximise the expected return *under uncertainty*, shown in Equation (7.1). To this end, our agent maintains a belief about the other agents' strategies, the posterior distribution $p(a^{(-1)}|\tau_{:t})$, where $\tau_{:t} = (s_0, u_0, r_1^{(1)}, \ldots, s_t)$ is the agent's experience until the current timestep t. This posterior is updated deterministically at every timestep following Bayes' rule.

[1] We do not make assumptions about the other agents' rewards, but assume access to a reward $r^{(1)}$ that leads to the desired behaviour if it acts self-interestedly. This applies to cooperative, competitive, and mixed settings.

The agent's actions are now conditioned on hyper-states (Duff et al., 2002) $s^+ = (s_t, b_t)$: the environment states, together with the current belief. We slightly abuse notation and write $b_t = b(\tau_{:t}) = p(a^{(-1)}|\tau_{:t})$ for the current belief, and condition the policy on this belief, denoted

$$a^{(1)}(u_t|s_t, b_t) \qquad \text{or} \qquad a^{(1)}(u_t|s^+). \tag{7.2}$$

We assume a parameterised posterior where the belief b_t is fully characterised by the distribution's parameters.

The transition and reward function for hyper-states are

$$T(s^+_{t+1}|s^+_t, u_t) = \mathbb{E}_{b_t}\left[T(s_{t+1}|s_t, u_t)\right] \delta(b_{t+1}{=}p(a^{(-1)}|\tau_{:t+1})) \tag{7.3}$$

where δ is the Dirac-delta function, and

$$R(r_{t+1}|s^+_t, u_t, s^+_{t+1}) = \mathbb{E}_{b_t}\left[R(r_{t+1}|s_t, u_t, s_{t+1})\right]. \tag{7.4}$$

The policy that maximises the expected return in the resulting belief MDP (Kaelbling et al., 1998) is the policy that optimally adapts to other agents given some prior belief about their strategies. Unfortunately, computing this solution analytically is intractable for all but the smallest tasks, and even existing approximation methods are restricted to small environments.

Prior Beliefs over Other Agents We assume that the support of the prior distribution P_A^{N-1} is over agents of the general form $a^{(i)}(u_t|\tau^{(i)}_{:t})$ described above. These agents can be non-stationary in that they can adapt their behaviour depending on the interaction history within a *single* game (which may consist of several rounds), up until the horizon H^+. For example, an agent might decide whether to cooperate or not based on how often the other agents cooperated. Any general solution therefore has to support these general types of other agents. In our approach, we use the prior distribution P_A^{N-1} for sampling the other agents' policies i.i.d. during meta-training (see Section 7.2).

7.2 Meta-Learning Interactive Bayesian Agents

In this section we introduce ***Me**ta **L**earning **I**nteractive **B**ayesian **A**gents* (MeLIBA), a method for meta-learning approximately Bayes-optimal agents that adapt to other agents. To this end, we bring together the theoretical motivation from Interactive Bayesian RL with modern methods from Meta-Learning, agent modelling, and belief inference. In particular, we propose a hierarchical sequential VAE to model beliefs over other agents, and jointly train this inference network with a policy that conditions on environment states and approximate beliefs.

7.2.1 Modelling Other Agents

We model an agent by its own permanent latent variable (m, also called *agent character*) and a temporal latent variable (m_t, also called *mental state*):

$$a^{(i)}(u_t|\tau_{:t}) \equiv a(u_t|s_t, m^{(i)}, m_t^{(i)}). \tag{7.5}$$

The character m does not change throughout the agent's lifetime. The mental state m_t can change in response to new observations at every timestep and allows us to model agents with non-stationary policies, i.e., policies conditioned on the interaction history. For example, the mental state of an agent can represent counts of how often other agents have cooperated, or capture the other agents' belief over other agents (and their beliefs, and so on).

This can be viewed as a probabilistic extension of an idea by Rabinowitz et al. (2018), who propose this split to model other agents and coined the terms agent character and mental state. Compared to Rabinowitz et al. (2018) who model agents in a purely observational setting, we want to learn to *interact* with other agents, and therefore need to maintain beliefs over the other agents. That is, we want to infer the posterior $p(m, m_t|\tau_{:t})$ given the agent's experience $\tau_{:t}$ up until the current timestep, where $m = (m^{(2)}, \dots, m^{(N)})$ and $m_t = (m_t^{(2)}, \dots, m_t^{(N)})$.

This is different than our belief model in the previous chapter, where we assumed one latent variable m that was fixed and described the unknown task. To accommodate this, we use a hierarchical sequential VAE, described next.

7.2.2 Approximate Belief Inference

Our agent's objective is to maximise expected future return (Eq (7.1)), the expected return given its current belief over the other agents' policies. Our agent therefore must predict other agents' future behaviour. We use this fact as an inductive bias to our model during meta-training, by learning to predict the future actions $p(u^{(i)}_{t:H}|s_t)$ of each of the other agents i at every timestep t.

Optimising $p(u_{t:H}|s_t)$ is intractable. Instead, we can optimise an evidence lower bound (ELBO):

$$\log\ p(u_{t:H}|s_t) \geq\ \mathbb{E}_{q(m,m_t|\tau_{:t})}\left[\log g^{\text{act}}\right] \tag{7.6}$$

$$-\ KL(q(m,m_t|\tau_{:t})||q(m,m_t|\tau_{:t-1})) \tag{7.7}$$

$$=\ ELBO_t, \tag{7.8}$$

where

$$g^{\text{act}} = \mathbb{E}_{p(u^{(-i)}_{t:H-1},s_{t+1:H}|u_t,s_t,m,m_t)}\left[\prod_{k=t+1}^{H-1} p(u_k|s_k,m,m_k)\right]. \tag{7.9}$$

Our model is a special type of VAE (Section 2.1.3; Kingma et al., 2014), and combines a sequential VAE (Chung et al., 2015) with a hierarchical latent structure (Zhao et al., 2017). We provide an intuitive explanation of this objective here, and the full derivation is given in Appendix C.1.

Line (7.6) (RHS) / Eq (7.9): At timestep t, given the current posterior $q(m,m_t|\tau_{:t})$, we predict the other agents' future actions $p(u_k|s_k,m,m_k)$ for all future timesteps $k=t+1,\ldots,H$. Since we assume independence between agents, this term factors across the other agents. During meta-training, we have access to these future actions from the collected rollouts. Future actions depend on the mental state $m^{(i)}_k$ for each other agent i, which changes at each timestep k in the future. Therefore, the model we maintain of the other agent includes a latent variable m_k that evolves over time, for which the encoder outputs only the current mental state, m_t. We model this using a recurrent architecture.

Line (7.7): We use a hierarchical latent structure where the agent's temporal state m_t can depend on its permanent type m. We use a *single* latent variable model for this, i.e., we do not factor the posterior distribution. This follows insights by Zhao et al. (2017) who show (for non-sequential VAEs) that learning hierarchical features can be obtained by having a single latent variable model, but a hierarchical structure in the network architecture. Intuitively, this means that we generate m from early layers in the encoder and m_t from deeper layers. Similarly, the decoder has a reverse hierarchical structure (see Fig 7.1). Following the variBAD approach from the previous chapter, we choose the prior (right-hand inside the KL) to be the previous posterior, akin to a filtering-type Bayesian update. This incentivises the posterior distribution to change slowly over time as the agent collects more data.

In practice, we represent the posterior using a diagonal Gaussian distribution $\mathcal{N}(\mu_t, \sigma_t)$, where $\mu_t = (\mu_t^{m,(2)}, \mu_t^{m_t,(2)}, \dots, \mu_t^{m,(N)}, \mu_t^{m_t,(N)})$ is the mean and where $\sigma_t = (\sigma_t^{m,(2)}, \sigma_t^{m_t,(2)}, \dots, \sigma_t^{m,(N)}, \sigma_t^{m_t,(N)})$ is the diagonal of the covariance matrix (which is zero on the off-diagnoals). For the prior for the first timestep we set $q(m, m_t) = \mathcal{N}(\mathbf{0}, \mathbb{I})$.

7.2.3 Meta-Learning Bayes-Adaptive Policies

Given the approximate posterior $q(m, m_t|\tau_{:t})$, we want to learn an approximately Bayes-optimal policy. To this end, we condition our policy not only on the environment state, but also on this approximate belief over the other agents' policies (see Equation (7.2)). This enables approximately Bayes-optimal behaviour: the policy can take into account its uncertainty over the other agents' policies when choosing actions, and use it to trade off exploration and exploitation. In practice, we approximate q using a Gaussian distribution $\mathcal{N}(\mu, \sigma)$. The approximate posterior is fully characterised by the mean μ_t and variance σ_t. The policy is then trained using standard RL methods by conditioning on environment states s_t and beliefs $b_t = (\mu_t, \sigma_t)$, $a^{(i)}(u_t|s_t, \mu_t, \sigma_t)$.

We use deep neural networks to represent the individual model parts as follows.

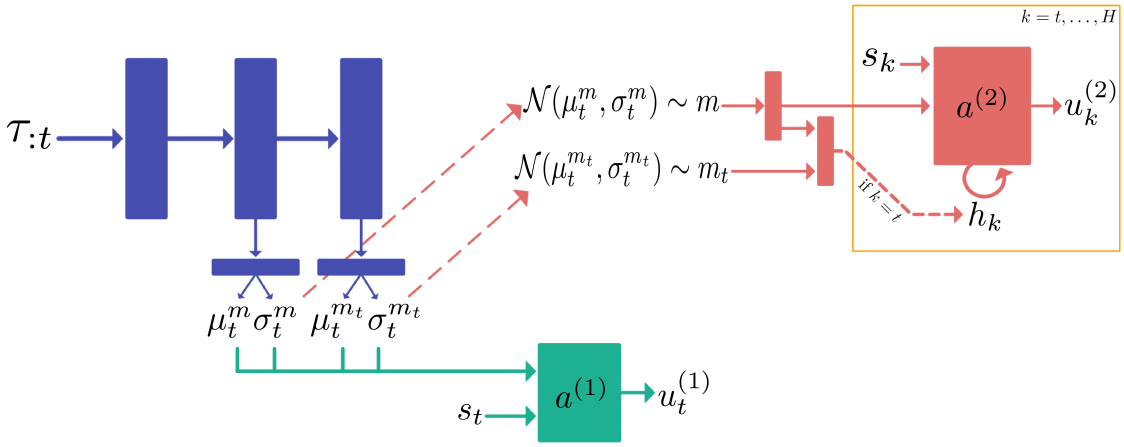


Figure 7.1: MeLIBA Network Architecture.

- **An encoder** $b_\phi(\tau_{:t}) = (\mu, \sigma)$ parameterised by ϕ, where the outputs fully characterise the approximate posterior $q(m, m_t|\tau_{:}t) = \mathcal{N}(m, m_t|\mu, \sigma\mathbb{I})$. The encoder has a hierarchical structure (see Fig 7.1). Each agent's latent variables (m, m_t) are modelled using a diagonal Gaussian of size $M + M_t$, such that $\mu^{m,(i)}, \sigma^{m,(i)} \in \mathbb{R}^M$ and $\mu^{m_t,(i)}, \sigma^{m_t,(i)} \in \mathbb{R}^{M_t}$.

- **A recurrent action decoder** to predict the other agents' future actions, $a_\psi^{(i)}(u_t|s_t, m^{(i)}, m_t^{(i)})$, parameterised by ψ. Since there are several other agents, the encoder outputs N distribution parameters, $\mu = (\mu^{(2)}, \ldots, \mu^{(N)})$ and $\sigma = (\sigma^{(2)}, \ldots, \sigma^{(N)})$, which are independently fed into the same decoder to compute the reconstruction loss.

- **A policy** $a_\theta^{(1)}(u_t|s_t, b_t)$, parameterised by θ and dependent on ϕ through the encoder (shorthand $a_{\theta,\phi}^{(1)}$).

Fig 7.1 shows the network architecture (assuming *one* other agent), with the colours above corresponding to the colours in the figure.

The overall objective of our agent is to maximise

$$\mathcal{L}(\phi, \psi, \theta) = \mathbb{E}_{P_A^{N-1}}\left[\mathcal{J}(\phi, \theta) + \lambda \sum_{t=0}^{H-1} ELBO_t(\phi, \psi)\right], \tag{7.10}$$

where $\mathcal{J}(\phi, \theta) = \mathbb{E}_{P_R^a, P_T^a, a_{\theta,\phi}^{(1)}}\left[\sum_{t=0}^{T} r_t\right]$. Like before (Chapter 6), we train the policy using PPO, and do not backpropagate the RL-loss through the encoder. For experiment details and hyperparameters see Appendix C.

7.3 Related Work

Adapting to unknown other agents is crucial for many applications where agents can be deployed, and their limited ability to do so is a critical limiting factor for real world applications. For example, in ad-hoc teamwork (Stone et al., 2010), agents trained to coordinate with each other fail to do so when paired with unseen partners (Carroll et al., 2019; Canaan et al., 2020a). We argue that a desirable objective for adaptive agents is the expected *online* return, and therefore we should aim to learn approximately Bayes-optimal behaviour.

To use the BAMDP framework from the previous chapter directly in a multi-agent setting, we could simply model other agents as part of the environment (Multiple Individual Learners; Tan, 1997) and maintain a belief over the reward and transition function. However, ignoring the known structure in the environment conflates several sources of stochasticity, making learning harder. By contrast, in the IBRL formulation (see Sec 7.1.3), we explicitly model beliefs over the other *agents*. In principle, this provides the right structure to solve the problem – however, computing the solution is generally intractable and existing work provides approximate solutions restricted to small environments or restrictive assumptions (Chalkiadakis, 2007; Hoang et al., 2013; Chalkiadakis et al., 2010; Hoang, 2014).

Related to IBRL, a large body of literature on Bayesian RL for multi agent settings and game theory uses beliefs over other agents, e.g., to compute best responses, maximise value gain, or learn models for planning (a.o., Carmel et al., 1990; Nachbar, 2005; Albrecht et al., 2016; Sadigh et al., 2016). What separates MeLIBA from some of these approaches is that we optimise for *Bayes*-optimal behaviour, and evaluate the online return (rather than episodic return).

A requirement for IBRL is the need to maintain a distribution over other agents, and to do so explicitly we need to model them. We outline the most relevant agent modelling literature here, and refer to Albrecht et al. (2018) and Hernandez-Leal et al. (2017) for comprehensive surveys. A popular way of modelling other agents is type-based modelling which assumes that the other agent has one of several (pre-defined or learned) types (Albrecht et al., 2017; Barrett et al., 2013; Stone et al.,

2010). In this work, we learn a latent variable model, in which the latent variables can be seen as a continuous representation of agent types learned in an unsupervised way. A central concept we use is separating behaviour shared by all other agents, and agent-specific permanent and temporal features. This is based on ideas from Rabinowitz et al. (2018), who refer to these as the agent's character (which is fixed throughout its life), and its mental state (which can change at any time and includes beliefs over other agents). Rabinowitz et al. (2018) consider a purely observational setting, whereas we are interested in learning Bayes-optimal behaviour in an *interactive* setting. This requires maintaining beliefs over the other agents' models, for which we combine hierarchical and sequential VAEs (Kingma et al., 2014; Chung et al., 2015; Zhao et al., 2017) to meta-train a belief inference model. Further, it requires training a policy that interacts with (and influences) other agents, and uses these beliefs when making decisions to maximise online return.

Closely related to our approach is the work of Papoudakis et al. (2020), who consider a similar problem setting to the one outlined in Sec 7.1. They focus on partially observable settings, and approximating global from local (agent-specific) information. To this end, they use a VAE to model other agents, and train an agent to use the latent variable to adapt. In contrast to our setting, they assume the other agents are Markov and only condition their actions on the current state – i.e., the other agents do not themselves adapt to the agents around them. Instead, we allow full observability but focus on the question of how to model other agents that are non-stationary. We show empirically that a different type of model is necessary if such non-stationary policies are allowed. Most importantly, Papoudakis et al. (2020) condition their policy on a *sample* from the approximate belief. However, this means they cannot learn a Bayes-optimal policy, since the agent cannot take into account its uncertainty about the other agents' policies. Like Papoudakis et al. (2020), many existing methods consider the other agents to be Markov, i.e., their policy depends only on the current state (He et al., 2016a; Carroll et al., 2019; He et al., 2016a). Our model in contrast can model agents that condition their actions on a history, and adapt to non-Markov agents.

Smith et al. (2020) propose Q-Mixing, which uses transfer learning to transfer Q-values to new mixture of opponents, and an opponent classifier to refine the mixing of Q-values to adapt to the new mix of opponents. In comparison, our method does not rely on having a discrete (small) number of other agents, but can also represent continuous distributions over other agents' policies.

Several methods have also been developed for adaptive agents on specific domains or making use of the structure of certain games, such as Avalon (Serrino et al., 2019), Hanabi (Foerster et al., 2019; Hu et al., 2019; Canaan et al., 2020b), or Poker (Bard et al., 2007; Southey et al., 2012). Our model on the other hand is general and can be applied to cooperative and competitive settings, and mixed and general sum games.

7.4 Empirical Evaluation

In this section we study MeLIBA and our modelling choices empirically. We first look at a small matrix game that allows us to analyse the hierarchical structure that emerges in the latent space. We compare MeLIBA to existing approaches for adaptive policies in complex multi-agent systems:

- RL^2 (Duan et al., 2016; Wang et al., 2016), the model-free Meta-Learning method introduced in Section 4.2. The architecture is similar to Figure 7.1 but with no decoder and no hierarchy in the encoder (no m_t). Unlike in MeLIBA, the RL loss is backpropagated through the encoder.

- LIOM (Papoudakis et al., 2020). The main differences to MeLIBA are that the policy receives a *sample* from the approximate posterior, the encoder is not hierarchical (no m_t), and the decoder is not recurrent.

- An *average* policy that cannot adapt but learns a policy that is good on average across all other agents.

MeLIBA and all baselines are based on PPO (for further implementation details see Appendix C). We also study our architecture choices quantitatively and qualitatively by ablating the hierarchical and the sequential structure.

7.4.1 Game of Chicken

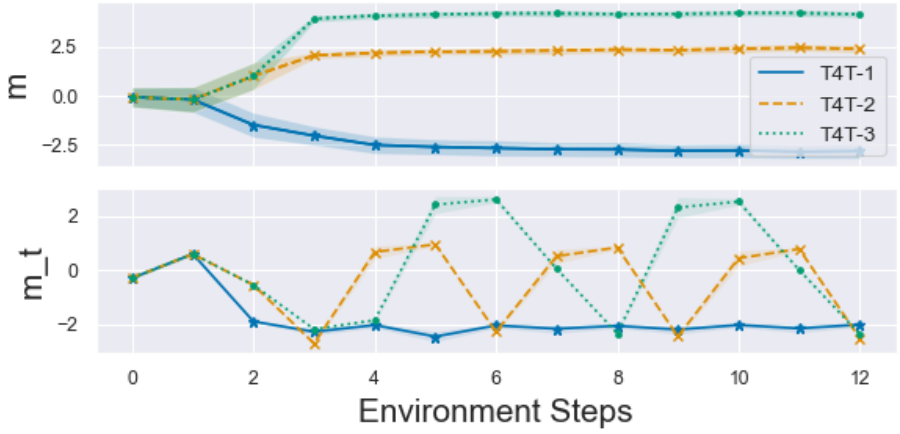
We first consider a simple matrix game, which allows us to verify that MeLIBA works as expected, and analyse its learned latent structure.

Environment The Game of Chicken (Bergstrom et al., 1998) is a 2-player competitive game (see Figure 7.2a). Imagine two cars driving towards each other: if nobody swerves, they crash and get a penalty (-1); if they both swerve they get a medium reward (1); and if only one of them swerves they are "the chicken" and get a low reward (0) and the other player gets a high reward (8). We hand-code three Tit-4-Tat agents, a strategy commonly used in game theory (Heap et al., 2004) which swerve if the opponent swerved once / twice / three times in a row (T4T-1/2/3). We randomly sample an agent to play with for 13 repetitions, which is long enough that the optimal strategy requires inferring and remembering the opponent's strategy, and short enough to analyse game play. The Bayes-optimal strategy is to swerve until the other agent also swerves and thereby reveals which T4T strategy it is using, after which it can be exploited by swerving just until the other agent will swerve, and then going straight to get a payoff of 8.

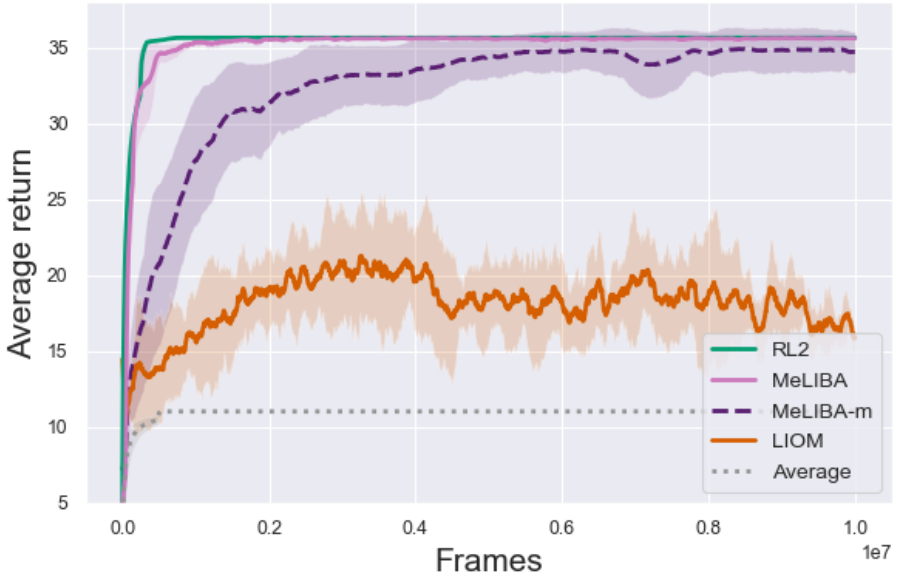
Latent Visualisation For MeLIBA we use a latent dimensionality of 1 each for the permanent and temporal aspects, $m \in \mathbb{R}$ and $m_t \in \mathbb{R}$. Figure 7.2b shows the latent variables when rolling out the meta-learned policy. The top is the mean (solid) and standard deviation (shaded) of the latent variable m. The separation between agent types happens after just 2 timesteps, which is how long it takes before the other agent starts swerving (given that MeLIBA learned to always swerve at the beginning). As expected, the standard deviation of the learned latent belief is high at first, and declines as the agent gathers more information about the other agent. The bottom of Figure 7.2b shows the same visualisation for the temporal latent variable, m_t. This shows that the model learned to count the number of swerves. Figure C.1 in Appendix C.2 shows that the latent variables of other architectures (no hierarchy, or only permanent/temporal latents) do not give the same desired structure.

	Swerve	Straight
Swerve	(1, 1)	(0, 8)
Straight	(8, 0)	(-1, -1)

(a) Payoff Matrix



(b) Latent variables during test rollout.



(c) Learning Curves

Figure 7.2: Results for the Game of Chicken. (a) The payoff matrix describing the Game of Chicken. **(b)** The latent variables of MeLIBA during three separate rollouts at test time, when playing against the different types of other agents T4T-1/2/3. Each colour represents a different game against one of these agents. The top shows the permanent latent variable m, which clearly separates the different agent types. The bottom the temporal latent variable m_t which keeps track of how often the agents have cooperated. **(c)** The learning curves for MeLIBA and ablations/baselines. Both RL^2 and MeLIBA learn the Bayes-optimal behaviour quickly. Using only a permanent latent state (MeLIBA-m) leads to a performance drop because the other agent is not modelled appropriately. LIOM, which conditions the policy on *samples* of the latent variable, underperforms in this environment, possibly because of the small latent dimension combined with the noise introduced via sampling.

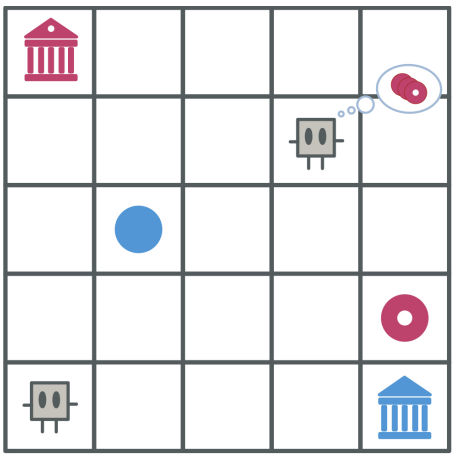
Performance Comparison Figure 7.2c shows the performance of MeLIBA compared to several baselines. RL^2 (with a 128-dimensional hidden state where MeLIBA has a 2D bottleneck) learns to solve the task quickly, which is unsurprising given the simplicity of the game. LIOM however performs poorly on this task (with similar architecture as MeLIBA). We believe this is because the latent space is small, and small perturbations (caused by the sampling) make PPO unstable. We found that backpropagating the RL loss through the encoder, or increasing the latent dimension, helped to some extent (see Figure C.2 in Appendix C). But even then, LIOM does not learn to solve the problem, which can be explained by LIOM not having the appropriate agent model (i.e., decoder) for opponents that adapt themselves. We test this hypothesis by testing MeLIBA with *only* the fixed latent m and a feed-forward decoder (denoted MeLIBA-m). As Figure 7.2c shows, MeLIBA-m cannot solve the task given the wrong model for the other agent.

7.4.2 Treasure Hunt

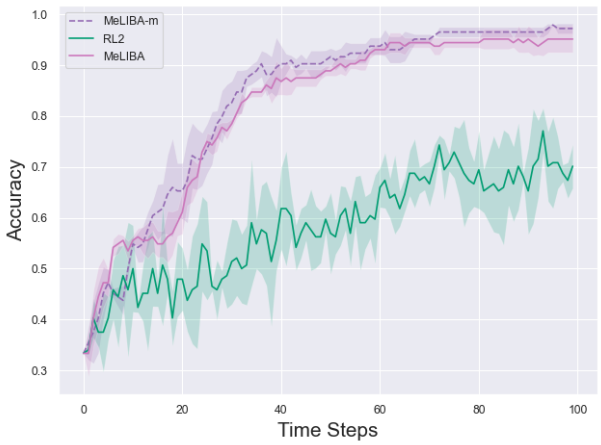
In this section we focus on whether predictive structure improves over model-free approaches. A model-free approach, where the policy is a recurrent network conditioned on its trajectory as in RL^2, can in principle learn the Bayes-optimal strategy (Ortega et al., 2019). However in practice we expect it to be difficult to learn to both infer the other agents' type, and use this information in the right way. In most settings studied in the single-agent setting, there is a close mapping between tasks and reward, for example if the agent gets rewarded when it reaches a goal, or if the reward is a dense signal of which direction the agent has to walk, and that the right inductive biases can help. In some multi-agent settings however, the mapping from task (i.e., what the other agent's strategies are) and the reward which the agent observes in the environment is more complex, since the reward function depends on the other agent's actions, which in turn can depend on complex history-dependent strategies. We therefore hypothesise that meta-learning to perform inference of the other agent's strategies is useful for maximising expected online return.

MeLIBA	MeLIBA-m	LIOM	RL^2	Average
10.1 (0.1)	**10.0** (0.2)	**10.2** (0.05)	7.7 (0.3)	1.3 (0.9)

(a) Meta-Test Performance



(b) Environment (5x5)



(c) Agent Type Prediction Accuracy

Figure 7.3: Treasure Hunt Game.

To this end, we use a gridworld version of the Treasure Hunt by Iqbal et al. (2018), which we designed to be able to easily control the other agents' strategy. It is a collaborative game with two agents who have to collect coloured coins and bring them to banks (see Fig 7.3b). Agents get a small bonus for collecting coins (0.1), a large bonus for dropping them at the correctly coloured bank (1), and a penalty for dropping them at the wrong bank (-1). Coins re-spawn at random locations after being dropped at a bank. We hard-code 3 other agents: two that only collect coins of one colour (unless the agent accidentally picks up a different-coloured coin, in which case it brings the coin to the correct bank), and an agent which alternates between colours. We use a 10×10 grid with horizon 100.

If no coin of the preferred colour is available, it does nothing and waits. To maximise return it is therefore beneficial to identify which colour the other agent prefers and focus on the other coins. I.e., the agent has to learn to infer the other agent's type, learn how this translates to what the other agent will do, and use this information to adapt accordingly and focus on the coins of a different colour. The results are shown in Table 7.3c (averages across 3 seeds and 95% confidence intervals in brackets; learning curves are in Appendix C, Figure C.3).

Comparison to Models with Permanent Latent Variables We first consider the comparison between MeLIBA and the two models MeLIBA-m and LIOM, which use a permanent latent variable only. The results show that there is no significant difference between these methods. At first glance this seems surprising given that one of the other agents switches between coin colours. However in this case, the VAE loss is lowest if the latent variable represents the agent type and the last coin collected, in which case the reconstruction term can be correct until the next coin is collected (and wrong only for terms further into the future). So despite the model mismatch, the information that is most relevant for the policy can still be captured in the latent variable. This suggests that in practice, if we know upfront that the other agent's actions do not depend on the history, we can use the simpler model MeLIBA-m. If this is not known, we do not lose out on performance if we use the full MeLIBA model. LIOM performs as well as MeLIBA-m in this setting, which indicates that sampling does not have any significant effect on performance.

Comparison to model-free methods RL^2 learns to adapt to some extent and performs better than the average policy; however, it is significantly outperformed by MeLIBA. This failure of RL^2 to learn the task compared to MeLIBA requires us to take a closer look at what both models do, in order to understand where this difference comes from. To this end, we train a logistic regression classifier (see Appendix C.2) to predict the other agent's type from the latent states of the models: for MeLIBA that is m and m_t, and for RL^2 that is the hidden state of the RNN. The results are shown in Figure 7.3c. For RL^2, we see that the other agent's strategy is represented to some extent in the hidden state of the recurrent model, and that the accuracy goes up as the policy gathers more data. Given that RL^2 fails to outperform the feed-forward policy however, it might not have learned to use the information about the other agent's strategy to adapt. In contrast, we can accurately predict the other agent's type from the latent variables in MeLIBA, confirming that predicting future actions of the other agent is a useful auxiliary task for learning to infer agent type in this environment.

7.5 Discussion

Conclusion We introduced MeLIBA, a general method for meta-learning approximately Bayes-optimal policies for a given distribution of other agents that can scale to complex multi-agent environments. We showed that MeLIBA's model of other agents learns a hierarchical latent representation of other agent types, separating the permanent and temporal internal states. On several environments we demonstrated that learning this latent representation by predicting future actions of other agents, and conditioning the policy on it, allows an agent to adapt to others. We conclude that maintaining explicit beliefs over other agents helps compared to model-free approaches, especially when there is not a tight coupling between other agent type and rewards.

Future Work In order to learn more informative priors, the prior distribution could be learned from (human) expert data using imitation learning techniques (Song et al., 2018), or by using existing multi-agent algorithms to pre-train the other agents (Canaan et al., 2020b). Interesting settings include humans playing video games together, where we might be interested in learning agents that can cooperate with humans in an ad-hoc setting; or cars driving on the highway, to later train self-driving cars in simulation to navigate in traffic before deploying them in the real world.

If such data is not available, another interesting direction for future work is to train all agents simultaneously, with the goal of generalising well to new agents that were not seen during training. This is a difficult problem, since sufficient diversity has to be maintained and agents can overfit to the agents they have been trained with (Carroll et al., 2019; Canaan et al., 2020a). In addition, this introduces a different type of non-stationarity in the other agents which must be addressed, since all agents now also evolve in-between episodes. In our current setting we assume the other agents can adapt *within* a single episode, but are reset at the beginning of each episode.

Even with this assumption, computing a Bayes-optimal policy is challenging. If we additionally want to model other agents that learn over time (not just within

a single episode), one way forward is to either assume their learning algorithm is known and use the interactions with them to do inference on how their character changes over time, or add a third level to our hierarchy (on top of the agents' permanent type m and its temporal type m_t that allows it to adapt within an episode), e.g., m_l, that models their latent *learning* algorithm. Our agent would then have to reason about how its own actions and learning behaviour influences the other agents' learning behaviour across episode/game boundaries. This is an incredibly challenging problem that is beyond the scope of this work.

In variBAD (Chapter 6), we modelled both the agent's past and future environment interactions (Sec 6.1.2). We found that this produces the best qualitative belief representations, and performs best (on par with modelling only the future). By contrast, in MeLIBA, we only model the future (Sec 7.2.2). We make this compromise for MeLIBA because we have a more complex latent variable model where one latent variable can change at every timestep (m_t), which is more difficult to model, and because we know from our variBAD experiments that modelling only the future performs well (see Sec 6.4.2). Ideally, we want to model both the past and future also in MeLIBA, which requires either having two decoders (one RNN to decode the past, and one RNN to decode the future), or a different decoder architecture. We leave this to future work.

Other interesting future directions are to, e.g., not require access of the other agent's observations at test time (Papoudakis et al., 2020), or include reward or state uncertainty (Gmytrasiewicz et al., 2005; Ng et al., 2012; Oliehoek et al., 2014).

Next Steps In the next chapter, we unveil a general challenge in Meta-RL: that of exploration *during* meta-training. We show how existing methods fail due to bad meta-exploration when rewards are very sparse, and propose a way to overcome this using novelty bonuses on hyper-states (the combination of environment states and beliefs).

8

Overcoming the Meta-Exploration Problem

Contents

In the previous chapters we saw that Meta-Learning can be used to learn policies that adapt fast to new tasks. This "learning to learn" requires a feedback signal on the meta-level about the agent's *learning performance*, to quantify how well the model performs after a learning episode. In Section 3.2 we introduced two objectives for measuring learning performance in the Fast Adaptation setting. If, however, this is difficult to measure – e.g., because the agent has not made sufficient learning progress or does not receive a lot of signal due to reward sparsity – meta-learning can fail. In this case, the agent faces an exploration challenge at the meta-level: how to find a good meta-learning signal in the first place.

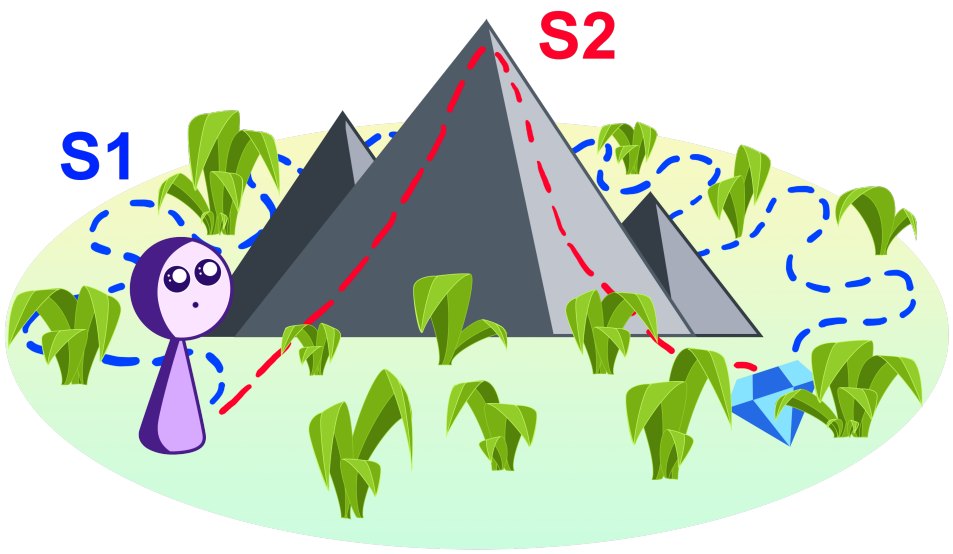


Figure 8.1: Illustration of the meta-exploration problem. We return to the example from our introduction to Meta-RL and learning to explore. As explained in Section 3.2, the superior exploration strategy is to climb up the mountain, see the treasure, and go there (s2), even though this incurs a large penalty in the short term for going up the mountain. However, most existing Meta-Learning methods fail to find this solution and learn strategy s1 instead, where the agent searches the grass for the treasure (Sec 8.4.1). The problem is that during meta-training, the agent does not sufficiently explore the space of task-exploration strategies. Going up the mountain incurs a penalty, so that the agent quickly avoids this without ever learning that this can lead to high returns in the long term. To overcome this, we need *meta-exploration* across task-exploration strategies.

8.1 The Meta-Exploration Problem

A weakness of existing Meta-RL methods is that they often rely on dense rewards or sufficiently small state spaces, such that even with naive exploration during meta-training the agent receives rewards that guide it towards good behaviour. However, we observe empirically (Sec 8.4) that if the rewards are sparse or if exploratory behaviour is penalised in the short term, existing methods can fail. This is problematic, since many real world applications have sparse rewards (e.g., a fail/success criterion), and crafting dense rewards is difficult, time-consuming, and prone to reward hacking (Abbeel et al., 2004; Hadfield-Menell et al., 2017). Hence, to make Meta-Learning practical for such settings, we need methods that can meta-learn even when rewards are sparse.

We distinguish between task-exploration and meta-exploration, as illustrated

in Figure 8.1. **Task-exploration** refers to the exploration behaviour we want to meta-learn: when in a new environment, the agent must explore to learn the task. We want this agent to be Bayes-optimal. **Meta-exploration** refers to the challenge of *exploring across tasks and adaptation behaviours* during meta-training. The agent has to (a) explore across individual tasks since the same state can have different values across tasks, and (b) learn about the shared structure between tasks to extract information about how to adapt, i.e., the agent must try out different task-exploration strategies during meta-training to find the Bayes-optimal one. Contrary to the Bayes-optimal task-exploration we want to meta-learn, we do *not* care about the rewards incurred during meta-exploration, but rather about efficiently gathering the data needed for Meta-Learning.

In this chapter, we propose ***Hyper**-State E**x**ploration* (HyperX), a novel method for meta-learning approximately Bayes-optimal exploration strategies when rewards are sparse. HyperX combines variBAD (Chapter 6) with two exploration bonuses for meta-training. The first is a novelty bonus on approximate hyper-states using random network distillation (Osband et al., 2018; Burda et al., 2019b) that encourages the agent to try out different task-exploration strategies, so that it can better find an approximately Bayes-optimal one. As this requires accurate task inference which we aim to meta-learn alongside the policy, the beliefs are inaccurate early in training and this bonus is not useful by itself. We therefore use a second exploration bonus to incentivise the agent to gather the data necessary to learn approximate belief inference. This bonus is computed using the discrepancy between the rewards and transitions that the belief model decoder predicts, and the ground-truth rewards and transitions the agent observes. This exploration bonus encourages the agent to visit states where the belief inference is incorrect and more data should be collected.

We show empirically that in environments without dense and informative rewards, current state of the art methods either fail to learn, or learn sub-optimal adaptation behaviour. In contrast, we show that HyperX can successfully meta-learn approximately Bayes-optimal strategies on these tasks.

8.2 Exploration in Approximate Hyper-State Space

Meta-learning good task-adaptation behaviour requires the agent to, during meta-training, gather the data necessary to learn good task-exploration strategies. If the environment rewards are sparse, they might however not provide enough signal for an agent to learn something if it follows naive exploration during meta-training. The agent needs to explore the state space sufficiently during meta-training, which is complicated by the fact that the same state can have different values across tasks. A good meta-exploration strategy also ensures that the agent tries out diverse task-exploration strategies which allow it to find an approximately Bayes-optimal one.

To address the meta-exploration problem, we propose HyperX (Hyper-State Exploration), a method to meta-learn approximately Bayes-optimal behaviour even when rewards are not dense. The two key ideas behind HyperX are:

1. We can incentivise the agent to try different task-exploration strategies during meta-training by rewarding novel *hyper*-states. By exploring the *joint* space of beliefs and states (i.e., hyper-states), the agent simultaneously (a) explores the state space, while distinguishing between visitation counts in different tasks due to changing beliefs, and (b) tries out different task-exploration strategies because these lead to different beliefs (even in the same state). We call this exploration bonus $r^{\text{hyper}}(s^+)$.

2. For the novelty bonus on hyper-states to be meaningful, the beliefs needs to be meaningful. However since the inference procedure is meta-learned alongside the policy, they do not capture task information early in training. We therefore additionally incentivise the agent to explore states where beliefs are inaccurate, by using the VAE reconstruction error (of the current rewards and transitions given the current belief) as a reward bonus, $r^{\text{error}}(s_t, r_t)$. Since the belief is conditioned on the history *including* the most recent reward r_t and state s_t, and the VAE is trained to predict rewards and states given beliefs, this bonus tends to zero over training.

In the following, we describe how to compute these bonuses.

Hyper-State Exploration To compute exploration bonuses on the hyper-states, we use random network distillation (see Appendix D.1) given its empirical successes in standard RL problems (Osband et al., 2017, 2018; Burda et al., 2019b) and theoretical justifications for deep networks (Pearce et al., 2020; Ciosek et al., 2020). To compute a reward bonus, a predictor network $f(s^+)$ is trained to predict the outputs of a fixed, randomly initialised prior network $g(s^+)$, on all hyper-states s^+ visited by the agent so far in meta-training. The mismatch between those predictions is low for frequently visited hyper-states and high for novel hyper-states. Formally we define the reward bonus for a hyper-state $s_t^+ = (s_t, b_t)$ as

$$r^{\text{hyper}}(s_t^+) = ||f(s_t^+) - g(s_t^+)||^2. \tag{8.1}$$

We train the predictor network f_ω alongside the policy and VAE.

Approximate Hyper-State Exploration To compute the hyper-state bonus above, we need an belief representation. This is provided by variBAD in the VAE latent. However at the beginning of meta-training, the beliefs do not sufficiently capture task information. If the policy does not explore and always gets a sparse reward which is uninformative w.r.t. the task, we fail to meta-learn to perform belief inference. The policy should therefore seek states where the VAE is not yet trained well. As a proxy for this, we use the VAE reconstruction error for the reward and states at the current timestep as a reward bonus:

$$r^{\text{error}}(r_t, s_t) = -\mathbb{E}_{q_\phi(m|\tau_{:t})}\Big[\log p_\psi(r_t|s_{t-1}, a_{t-1}, s_t, m) + \log p_\psi(s_t|s_{t-1}, a_{t-1}, m)\Big]. \tag{8.2}$$

Since r_t and s_t were observed in $\tau_{:t}$, the encoder q can in principle encode all necessary information for the decoder p to predict the current reward and state transition exactly. Early in training, these predictions are inaccurate in states where the rewards/transitions differ a lot across tasks. Therefore this exploration bonus incentivises the agent to visit states that provide crucial training data for the VAE. In practice, this reward bonus is computed using one Monte Carlo sample from q. If only one aspect (reward or transitions) change across tasks, variBAD only learns the respective decoder, and so we only use the respective reward bonus.

Algorithm 3 HyperX Pseudo-Code

Input: Distribution over MDPs $p(M)$
Initialise: Encoder q_ϕ, decoder p_ψ, policy π_θ, RND predictor network f_ω, buffer $\mathcal{B} = \{s_0, b_0\}$
for $k = 1, \dots, K$ **do**
  Sample environments $\mathbf{M} = \{M_i\}_{i=1}^{B}$ where $M_i \sim p$
  for $M_i \in \mathbf{M}$ **do**
    Reset s_0, h_0, b_0
    for $t = 0, \dots, T-1$ **do**
      Choose action: $a_t = \pi_\theta(s_t, b_t)$
      Step environment: $s_{t+1}, r_{t+1} = M_i.step(a_t)$
      Update belief: $b_{t+1} = q_\phi(s_{t+1}, a_t, r_{t+1}, h_t)$
      Compute exploration bonuses:
        $r^{\text{hyper}}(s^+_{t+1})$ using Eq (8.1)
        $r^{\text{error}}(r_{t+1}, s_{t+1})$ using Eq (8.2)
      Add data to buffer:
        $\mathcal{B}^p.add(s_{t+1}, b_{t+1}, a_t, r_{t+1}, r^{\text{hyper}}_{t+1}, r^{\text{error}}_{t+1})$
    end for
  end for
  Update VAE, policy, and RND predictor network:
  $(\phi, \psi) \leftarrow (\phi, \psi) + \alpha_{(\phi,\psi)} \nabla_{(\phi,\psi)} \sum_{t=0}^{H+} ELBO_t(\phi, \psi)$
  $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \hat{\mathcal{J}}(\theta)$ using Eq (8.3)
  $\omega \leftarrow \omega - \alpha_\omega \nabla_\omega \mathbb{E}_{s^+ \sim \mathcal{B}} [\, ||f_\omega(s^+) - g(s^+)||_2^2 \,]$
end for

Meta-Training Objective Putting these bonuses together, the new objective for the agent is

$$\hat{\mathcal{J}}^+(\theta) = \mathbb{E}_{b_0, T^+, \pi_\theta}\Bigg[\sum_{t=0}^{H^+-1} \gamma^t R^+(r_{t+1} | s^+_t, a_t, s^+_{t+1}) + \lambda_h r^{\text{hyper}}(s^+_{t+1}) + \lambda_e r^{\text{error}}(r_{t+1}, s_{t+1}) \Bigg]. \tag{8.3}$$

While in principle these bonuses tend towards zero during meta-training, we anneal their weights (λ_h, λ_e) over time. This prevents the policy to keep meta-exploring at meta-test time and ensure that it maximises only the expected online return. Algorithm 3 shows pseudo-code for HyperX. Implementation details are given in Appendix D.3. Source code is available at `https://github.com/lmzintgraf/hyperx`.

8.3 Related Work

Exploration Bonuses Deep RL has been successful on many tasks, and naive exploration via sampling from a stochastic policy is often sufficient if rewards are dense. For hard exploration tasks this performs poorly, and a variety of more sophisticated exploration methods have been proposed. Many of these reward novel states, often using count-based approaches to measure novelty (Strehl et al., 2008; Bellemare et al., 2016; Ostrovski et al., 2017; Tang et al., 2017). A prominent method is Random Network Distillation (RND) for state-space exploration in RL (Osband et al., 2017, 2018; Burda et al., 2019b; Ciosek et al., 2020). We use it for *hyper*-states in this paper. We further use an exploration bonus based on the VAE reconstruction error of rewards and transitions. Prediction errors of environment dynamics are used to explore the MDP state space, by, e.g., Achiam et al. (2016), Burda et al. (2019a), Pathak et al. (2017), Schmidhuber (1991), and Stadie et al. (2015).

Meta-Exploration To the best of our knowledge, all existing Meta-Learning methods for Online Adaptation rely on myopic exploration during meta-training. As we observe empirically (Sec 8.4), this can cause them to break down if rewards are too sparse. Two recent works also study the problem of exploration during meta-training, albeit for Few-Shot Adaptation. Still, similar considerations about meta-exploration apply. Zhang et al. (2021) propose MetaCURE, which meta-learns a separate exploration policy that is intrinsically motivated by an exploration bonus that rewards information gain. Liu et al. (2020) propose DREAM, where a separate exploration policy is trained to collect data from which a task embedding (pre-trained via supervision with privileged information) can be recovered. These methods can, in principle, still suffer from poor meta-exploration if rewards are so sparse that there is no signal to begin with, and information gain / task embedding recovery cannot be measured. We empirically compare to MetaCURE and find that this is indeed true (Sec 8.4.3).

If available, privileged information can be used during meta-training to guide exploration, such as expert trajectories (Dorfman et al., 2021), dense rewards for

meta-training but not testing (Rakelly et al., 2019), or ground-truth task IDs / descriptions (Liu et al., 2020; Kamienny et al., 2020). HyperX works well even if such information is not available.

Exploration in POMDPs Meta-exploration is related to exploration when learning in partially observable MDPs (POMDPs, Cassandra et al. (1994)), of which BAMDPs are a special case. This topic is mostly studied on small environments. Similar to our work, Cai et al. (2009) incentivise exploration in under-explored regions of belief space. However, they use two separate policies for exploration and exploitation and rely on Bayesian learning to update them, restricting this to small discrete state spaces. Several authors (Poupart et al., 2008; Ross et al., 2008; Doshi et al., 2008; Ross et al., 2011) explore model-based Bayesian reinforcement learning in partially observable domains. By relying on approximate value iteration to solve the planning problem, they are also restricted to small environments. To our knowledge, only Yordanov (2019) provides some initial results on a simple environment using Random Network Distillation. They propose various ways to deal with the non-stationarity of the latent embedding such as using a random *recurrent* network that aggregates past trajectories.

8.4 Empirical Evaluation

In this chapter, we present four experiments that illustrate how and why HyperX helps agents meta-learn good online adaptation strategies (Sec 8.4.1-8.4.3), and results on sparse MuJoCo Ant-Goal to demonstrate that HyperX scales well (Sec 8.4.4).

We also evaluated HyperX on standard Meta-RL benchmarks; the 2D navigation Pointrobot, Meta-World ML1 with sparse rewards, or the otherwise challenging dense AntGoal environment. However, we found that existing methods already perform well and there is no room for improvement via better exploration. We therefore refer these results to Appendix D.2.

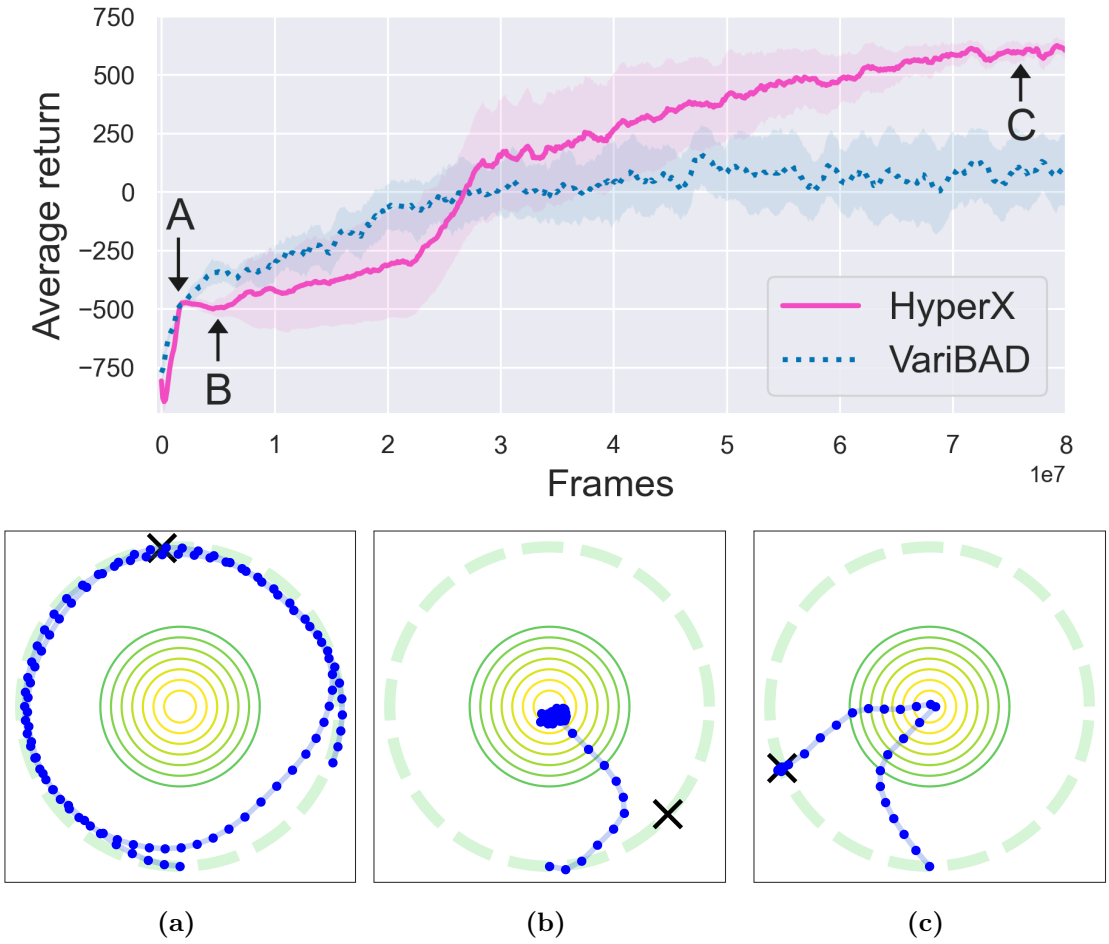


Figure 8.2: Treasure mountain results. Top: Learning curves for HyperX and variBAD (10 seeds, 95% confidence intervals shaded). Bottom: Behaviour of the HyperX agent at different stages of training. HyperX learns the superior task-exploration strategy of climbing the mountain to see the treasure, and going there directly after. VariBAD learns the inferior strategy of walking around the circle until finding the treasure (see rollouts in Appendix D.2).

8.4.1 Treasure Mountain

We consider our earlier example (Fig 8.1), where the agent's task is to find a treasure hidden in tall grass. There are two good task-exploration strategies: (s1) search the grass until the treasure is found, or (s2) climb the mountain, spot the treasure, and go there directly. The latter strategy has higher expected return, but is harder to meta-learn since (a) climbing the mountain is discouraged by negative rewards and (b) the agent must meta-learn to interpret and remember the treasure location it sees from the mountain.

We implement this as follows (details in Appendix D.3): the treasure can be anywhere along a circle. The agent gets a sparse reward when it reaches the treasure, and a time penalty otherwise. Within the circle is the mountain, represented by a smaller circle. Walking on it incurs a higher time penalty. The agent's observation is 4D: its x-y position, and the treasure's x-y coordinates, which are *only* visible from the mountain top. The agent starts at the bottom of the circle and has one rollout of 100 steps to find the treasure.

Figure 8.2 shows the learning curves of variBAD (which uses no exploration bonuses for meta-training) and HyperX, with HyperX performing significantly better. Figures 8.2a-8.2c show the behaviour of HyperX at different times during training. At the beginning (8.2a), it explores along the circle (but does not stop at the treasure and explores further) and its performance increases. Then, it discovers the mountain top: because the VAE reconstruction error r^{error} is high there, it initially just stays there (8.2b). Performance drops since the penalty for climbing the mountain is slightly higher than the time penalty the agent gets otherwise. Finally, at the end of training (8.2c) it learns the optimal strategy s2 (consistently across all 10 seeds). Inspection of the rollouts show that variBAD and other methods for online adaptation (RL^2 (Duan et al., 2016; Wang et al., 2016) and Belief Learning (Humplik et al., 2019)) always only meta-learn the inferior task-exploration strategy s1 (see Appendix D.2).

When using *only* r^{hyper}, the agent only learns the inferior strategy s1: early in training, hyper-states are meaningless and the agent stops exploring the mountain top. When using only r^{error}, the agent learns the superior strategy s2 around 70% of the time. For learning curves see Appendix D.2.

This experiment shows that HyperX tries out different task-exploration strategies during meta-training, and can therefore meta-learn a superior exploration strategy even when it is expensive in the short term, but pays off in the longer run.

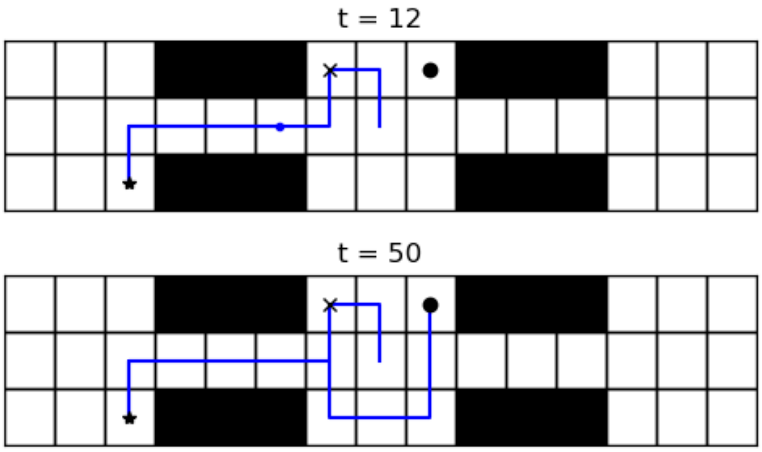


Figure 8.3: Multi-stage Gridworld. Goal 1 (×) unlocks goal 2 (∗), which unlocks goal 3 (•). Example behaviour of HyperX in blue.

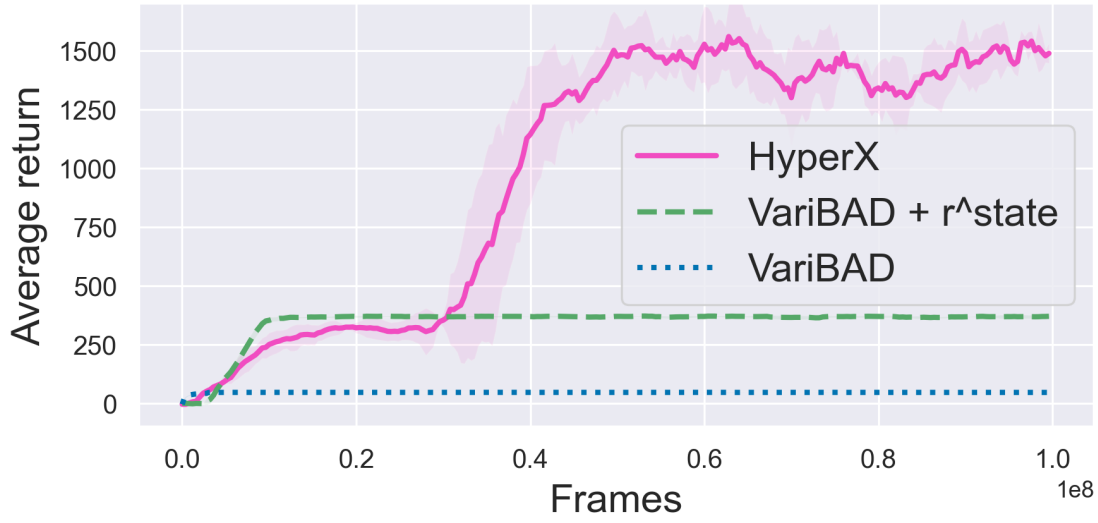


Figure 8.4: Multi-stage Gridworld learning curves. (3 seeds)

8.4.2 Multi-Stage Gridworld

Next, we consider a partially observable multi-stage Gridworld which illustrates how, without the appropriate exploration bonuses on hyper-states, existing methods can converge prematurely to a local optimum.

The Gridworld is illustrated in Figure 8.3: three rooms are connected by narrow corridors, and three (initially unknown) goals (G1-G3) are placed in corners of rooms: The goals provide increasing rewards, i.e. $r_1 = 1$, $r_2 = 10$ and $r_3 = 100$, but are only sequentially unlocked; G2 (r_2) is only available after G1 has been

reached; G3 (r_3) is only available after G2 has been reached. The environment is partially observable (Poupart et al., 2008; Cai et al., 2009) as the agent only observes its position in the environment and not which goals are unlocked. If the agent is not on an (available) goal it gets $r = -0.1$. G1 and G3 are always in the middle room, G2 always in an outer room on the same side as G1. The agent starts in the center of the middle room and has $H = 50$ steps. The best strategy is to search the first room for G1, then search the appropriate room for G2, and then return to the middle room to find G3.

Figure 8.4 compares variBAD, variBAD with state-novelty bonus, and HyperX. VariBAD learns to reach G1 and remains there, effectively receiving only r_1 at every timestep. VariBAD+$r(s)$ learns to find G2 and stay there, but fails to find G3. Only HyperX solves the problem (see behaviour in Figure 8.3). Methods which use a purely state-based exploration bonus such as variBAD+$r(s)$ are unable to find G3 in the middle room as those states s (not hyper-states (s, h)) appear already sufficiently explored. In contrast, a novelty bonus on the hyper-state $r(s, h)$ like in HyperX leads to a high novelty bonus in the middle room once G2 is found because the belief changes.

These results show that without the right exploration bonuses during meta-training, the agent can prematurely converge to a suboptimal solution. Additionally, we see that that HyperX can handle this degree of partial observability.

8.4.3 Sparse HalfCheetahDir

To demonstrate the effect of the different exploration bonuses, we consider the following example for which we can compute exact beliefs. The environment is based on the HalfCheetahDir MuJoCo environment, which we studied in Chapters 5 and 6, and which is commonly used in meta-RL (e.g., Finn et al., 2017a; Rakelly et al., 2019). In the dense version the agent is rewarded according to its (1D) velocity in the correct direction, and existing Meta-RL methods can learn good solutions (see Figure 6.5). We now consider a *sparse* version without resets: the agent only receives the dense reward once it walks sufficiently far away from its

starting position, outside an interval $[-5, 5]$ (and a control penalty otherwise), and has 200 environment steps to adapt. This makes it much more difficult to find the optimal adaptation strategy, which is to walk far enough in one direction to infer the task, and turn around in case the direction was wrong.

Without dense rewards, existing Meta-Learning algorithms fail to learn this strategy, as is the case for RL^2 (Duan et al., 2016; Wang et al., 2016), PEARL (Rakelly et al., 2019), ProMP (Rothfuss et al., 2019), E-MAML[1] (Stadie et al., 2018) and variBAD, as shown in Table 8.1a. HyperX in contrast successfully meta-learns the correct task-adaptation strategy. For all baselines, we used the available open source code.

Exploration in Exact Hyper-State Space To investigate how the exploration bonuses in HyperX help solve this task, we first assume that we have access to the true hyper-state $s_t^+=(s_t, b_t)$, including the true belief which we define as follows. The prior belief is $b_0=[0.5, 0.5]$ and it can be updated to the posterior belief $b=[1, 0]$ (left) or $b=[0, 1]$ (right) once the agent observes a single reward outside of the interval $[-5, 5]$. Since we can manually compute this belief, we can train a Belief Oracle using standard reinforcement learning, by conditioning the policy on the exact hyper-state. Table 8.1b shows the performance of the Belief Oracle, with and without reward bonuses. Without the bonus, even this Belief Oracle does not learn the correct behaviour for this seemingly simple task. When adding the exploration bonus $r^{\text{hyper}}(b, s)$ on the hyper-state, the policy learns approximately Bayes-optimal behaviour.

Figure 8.5 shows how the bonuses incentivise the agent to explore, with the red gradient in the background visualising the reward bonus (darker meaning more bonus). When the agent walks outside the sparse-reward interval and updates its belief, the reward bonus in the opposite direction becomes high since it has not yet visited that area with the updated belief very often. Table 8.1 (top) shows that a policy trained with a reward bonus only on the state, $r(s)$, performs worse. The

[1] E-MAML/ProMP/PEARL are not designed to adapt within a single episode, so we do the gradient update (E-MAML/ProMP) / posterior sampling (PEARL) after half an episode.

reason is that the agent is not incentivised to explore states to the far right after its belief has changed. Inspection of the learned policies shows that agents trained with a state exploration bonus do go outside the interval, and just return to and stay in the sparse-reward zone if the direction was wrong (see Appendix D.2).

Method	Avg Return
VariBAD	-1.1
E-MAML	-0.4
ProMP	-0.4
Humplik et al.	-0.1
RL^2	-0.7
PEARL	-0.1
HyperX	819.6

(a) Method Comparison

Method	Avg Return
Belief Oracle	-3.0
Belief Oracle + $r(b)$	-3.6
Belief Oracle + $r(s)$	639
Belief Oracle + r^{hyper}	824

(b) Ground Truth Beliefs + Exploration Bonuses

Method	Avg Return
HyperX , r^{error} only	-0.7
HyperX , r^{hyper} only	462
RL^2 + r^{state}	463
RL^2 + r^{hyper}	477
Humplik et al. + r^{hyper}	840
Humplik et al. + r^{hyper} + r^{error}	850
VariBAD + r^{metaCURE}	-0.3
VariBAD + r^{metaCURE} + r^{state}	548
VariBAD + r^{metaCURE} + r^{hyper}	811

(c) Ablation Studies.

Table 8.1: HalfCheetahDir meta-test performance. (a) HyperX successfully solves this task, while existing Meta-Learning methods fail. (b) Not even an agent with access to the correct belief is able to solve this task without appropriate exploration bonus. (c) Both exploration bonuses are necessary to succeed, but different realisations with similar effects (e.g., metaCURE) work as well. The bonuses can be used with other methods, if they provide an approximate belief (for r^{hyper}) and a way of measuring how good the inference is (for r^{error}).

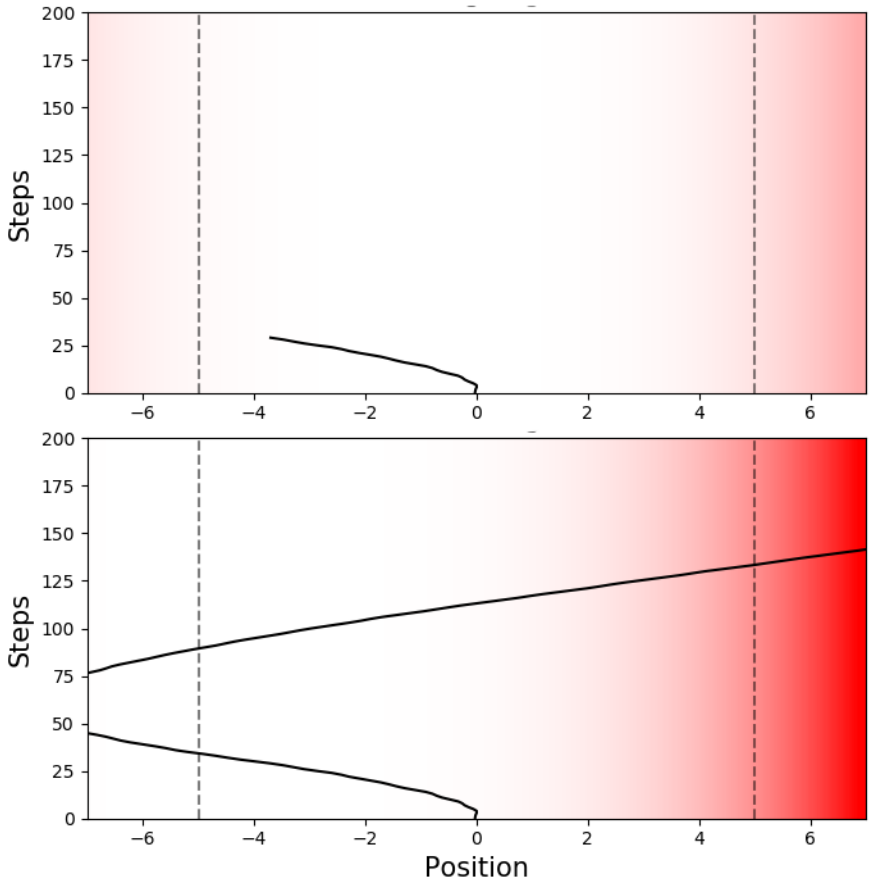


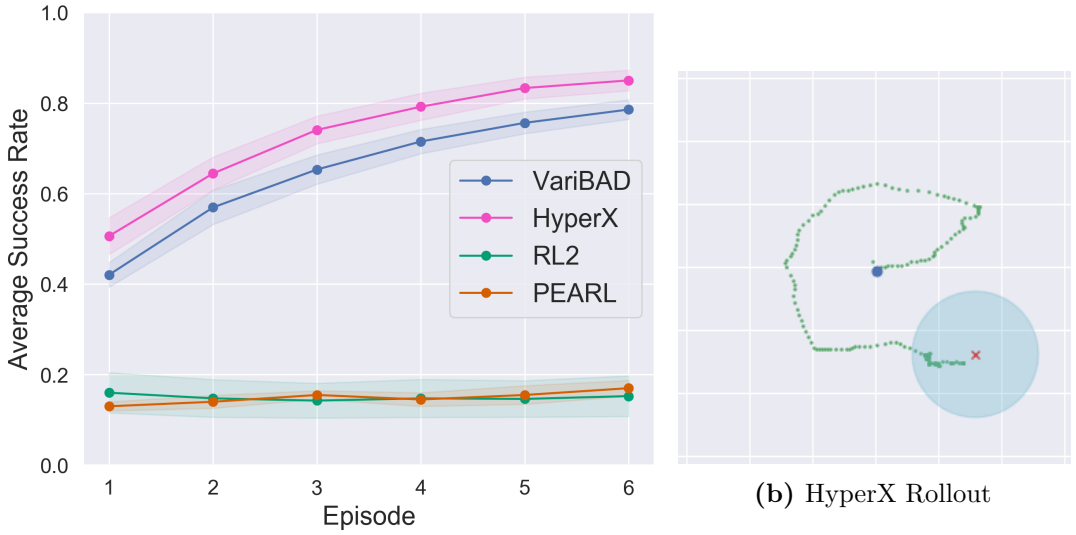
Figure 8.5: HalfCheetahDir example rollouts for the Belief Oracle, early in meta-training (1e6 frames), trained with the hyper-state-bonus $r^{hyper}(s^+)$. The y-axis denotes time in agent steps. The background visualises the hyper-state-bonus: darker means higher bonus. **Top**: At the beginning of the episode, the agent's belief is the prior, and the exploration bonus incentivises it to explore away from the familiar start position. **Bottom**: Once the agent enters the dense-reward zone, it infers the task and updates its belief. Now, the states on the right side seem novel since the agent has not seen them together with the posterior belief.

HyperX: Exploration in Approximate Hyper-State Space Above we assumed access to the true belief b_t. When meta-learning how to perform approximate belief inference alongside the policy however, these beliefs change over time and are initially inaccurate. As Table 8.1c (top) shows, using only the hyper-state exploration bonus r^{hyper}, which worked well for the Belief Oracle, performs sub-optimally. This is because early in training the belief inference is inaccurate, and the hyper-state bonus is meaningless: the agent prematurely and wrongly assumes it has sufficiently explored. Only when adding the error reward bonus r^{error} as well

to incentivise the agent to explore areas where the belief inference makes mistakes, can we meta-learn approximately Bayes-optimal behaviour for this task. Using only the error reward bonus r^{error} performs poorly as well.

Other Meta-Learners While we build HyperX on variBAD, the same exploration bonuses can be used for other Meta-Learning methods that provide (a) a belief representation, and (b) a measure of how good the belief inference is. One such method is the work by Humplik et al. (2019) who train a belief model using the ground-truth task description. This makes meta-learning inference easier, and the exploration bonus r^{error} may not be necessary: for sparse HalfCheetahDir, using only the hyper-state bonus (r^{hyper}) is sufficient, as shown in Table 8.1c (middle). This result is not directly comparable to HyperX since it uses privileged information, whereas HyperX meta-learns inference in an unsupervised way. For the method RL^2, the RNN hidden state can be used as a belief proxy to compute the hyper-state bonus. The second exploration bonus (r^{error}) however, cannot be estimated because the hidden state is only used implicitly by the agent. As Table 8.1c shows, an exploration bonus on the state (r^{state}) or on the hyper-state (r^{hyper}) for RL^2 is not sufficient to solve the task.

Comparison to MetaCURE Recently Zhang et al. (2021) proposed MetaCURE for meta-exploration. They use information gain as an intrinsic reward, defined as the difference between the prediction errors (of states/rewards) given the agent's current experience or given the ground-truth task. For the sparse HalfCheetahDir task this is high when the agent first steps over the interval bound. Even though MetaCure is defined for episodic task-adaptation, we can use its bonus in the online adaptation setting as well. Compared to HyperX it requires training two additional prediction networks, and it relies on privileged task information during meta-training to do so. Table 8.1c shows that this exploration bonus alone is not sufficient to solve the task – it only incentivises the agent to go to the interval boundary. Adding a hyper-state bonus is required to solve the task.



(a) Success Rates

(b) HyperX Rollout

Figure 8.6: Sparse AntGoal results. 8.6a shows the success rate per episode (10 seeds, standard error shaded). 8.6b shows a cherry-picked test rollout of the HyperX agent during the first episode.

8.4.4 Sparse MuJoCo AntGoal

To show that HyperX can scale to more complex environments, we evaluate it on a harder MuJoCo task, a sparse version of the Ant-Goal-2D (Rothfuss et al., 2019; Rakelly et al., 2019) that we also used in Chapter 6. In the dense version, the agent gets a reward relative to its distance to the goal. We make this task significantly harder by sparsifying the rewards, giving the agent the dense reward only if it is within a certain distance of the goal (the blue shaded area in Figure 8.6b). The best exploration strategy is therefore to spiral outwards until a reward signal is found.

Figure 8.6a shows the success rates of HyperX, variBAD, RL^2 and PEARL across different episodes, where an agent is successful if it enters the goal circle. Both RL^2 and PEARL only learn to go to goals close to the starting position, and therefore have low success rate. HyperX and variBAD learn to sometimes solve the task, with HyperX having a slightly higher success rate. This result illustrates that a lack of good exploration can crucially affect final performance: the success rate in the 6th episode is good iff early exploration was good. Plots for the returns across episodes and learning curves can be found in Appendix D.2.

Figures D.5 and D.6 (Appendix D.2) show example behaviours of the meta-trained HyperX and variBAD agent. HyperX learns to efficiently search the space of possible goals, occasionally in the shape of a spiral (Fig 8.6b), finding the goal in the first episode. VariBAD can also learn to search in a spiral but is less efficient and more likely to fail. Once the agent reaches dense-reward radius around the goal, it is able to determine where the goal is and heads there directly. In subsequent episodes, the agent returns directly to the goal.

Overall our empirical results show that HyperX can meta-learn excellent adaptation behaviour on challenging sparse reward tasks where existing methods fail. We also evaluated HyperX on sparse environments used in the literature, like sparse PointRobot (Rakelly et al., 2019), and sparse Meta-World ML1 (Yu et al., 2019) but refer to these in Appendix D.2 since variBAD can already solve these (Chapter 6).

8.5 Conclusion

This chapter showed that existing Meta-Learning methods can fail if the environment rewards are not densely informative with respect to the task, and myopic exploration during meta-training is insufficient. We highlighted that in this case, special attention needs to be paid to *meta-exploration*. This applies to many different problem settings, but we focused on online adaptation where the agent aims to maximise expected online return. Here, task-exploration is particularly challenging since the agent has to trade off exploration and exploitation.

We proposed HyperX, which uses two exploration bonuses to incentivise the agent to explore in approximate hyper-state space during meta-training. This way, it collects the data necessary to learn approximate belief inference (incentivised by r^{error}), and tries out different task-exploration strategies during meta-training (incentivised by r^{hyper}). We demonstrated empirically how Meta-Learning without explicit meta-exploration can fail and why, and showed that HyperX can solve these tasks.

Part III

Discussion

9
Open Research Areas

Contents

In this thesis, we addressed big challenges around how to develop agents that can adapt quickly and efficiently in complex environments, and offered scalable solutions for single agent, multi-agent, and sparse reward settings. We took a context-based approach (introduced in Chapter 5), and paired it with Bayesian reasoning, to develop agents that can autonomously adapt to novel environments or other agents, all the while efficiently trading off exploration and exploitation (Chapters 6-8).

We believe that there exist at least three major open problems that will become more relevant as the field advances: constructing better benchmarks and an understanding of testbed requirements, dealing with task distribution shifts between training and testing, and building better belief models. The latter two can be a direct continuation of the work presented in this thesis. We discuss all three challenges in the following sections.

9.1 Benchmarks

In our experiments, we relied on (at the time) existing Meta-RL benchmarks and toy tasks to evaluate our methods. On some of these, we observe a ceiling effect (e.g., for some MuJoCo tasks in Section 6.3.3). This is problematic because it stalls research, and makes comparison between approaches difficult. Moving forward, we need more challenging benchmarks or modifications to existing ones (like sparsifying rewards as in Chapter 8), to push further the capabilities of fast-adapting agents. While doing so, it may be useful to understand and focus on specific sub-challenges within Meta-RL. Several benchmarks that do so have recently emerged. The Meta-World benchmarks ML-10 and ML-45 (Yu et al., 2019) have a separate training and test set, requiring the agent to generalise more at test time (see next section). The Alchemy benchmark (Wang et al., 2021) requires an agent to learn about underlying structure and do hypothesis testing together with online inference. The NetHack learning environment (Küttler et al., 2020) challenges agents to condition on natural language, and tests for generalisation since the levels are procedurally generated. Applying the methods we presented in this thesis to these new benchmarks is an exciting way to see how well they scale, identify concrete challenges that lie ahead, and work towards addressing them.

Requirements of Training Task Distributions Apart from the need for new benchmarks, we believe that it is also necessary to develop a better understanding of the requirements for meta-training distributions in the first place. E.g., how many tasks are necessary to learn good performance, and how does this relate to the properties of the method? How can we detect whether meta-training is working, and transfer across tasks is successful? Answering some of these questions may require measuring task similarity (Carroll et al., 2005), which can be useful to sample tasks for meta-training, or to determine if a test task is close to what was seen during training. Other exciting research directions are to combine unsupervised environment generation (Jiang et al., 2021) with Meta-Learning, or to develop benchmarks that have procedurally generated tasks (Cobbe et al., 2019; Küttler et al., 2020).

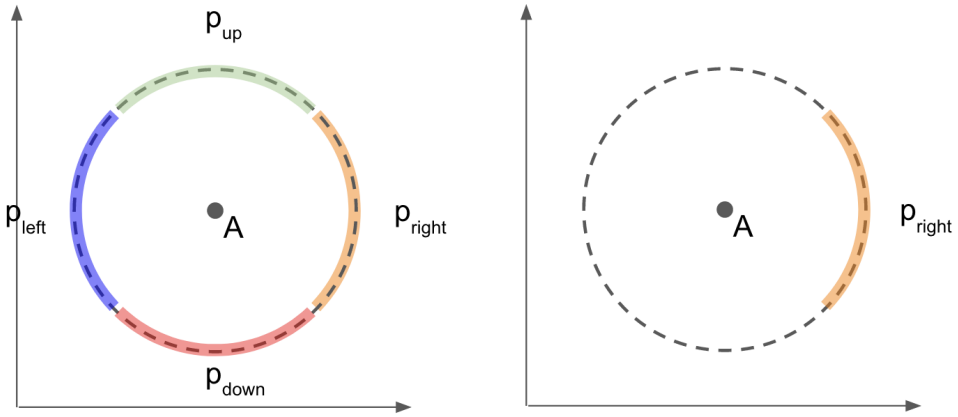
9.2 Dealing with Task Distribution Shifts

Throughout the thesis, we assumed that the tasks for meta-training and testing come from the same distribution. However, when deploying agents in the real world, there will likely be a shift in the distribution, e.g., because we meta-train in a simulator that is not perfect.

Example To illustrate how challenging it can be to deal with distribution shifts, we perform the following experiment. In a 2D environment, an agent has to navigate to a goal position that is somewhere on a circle around its starting position. We consider four different meta-training distributions shown in Figure 9.1, where the goals all lie either on the left, top, right, or bottom, of the circle. The meta-test distribution only includes tasks where the goal lies on the right side. Figure 9.2 shows how long it takes for variBAD models to adapt at test time (using gradient descent), after meta-learning on the different training task distributions. We see that it takes the pre-trained models long to recover; *much longer* than training from scratch (Fig 9.2a). It is even worse in the sparse reward setting: some agents do not recover at all (Fig 9.2b).

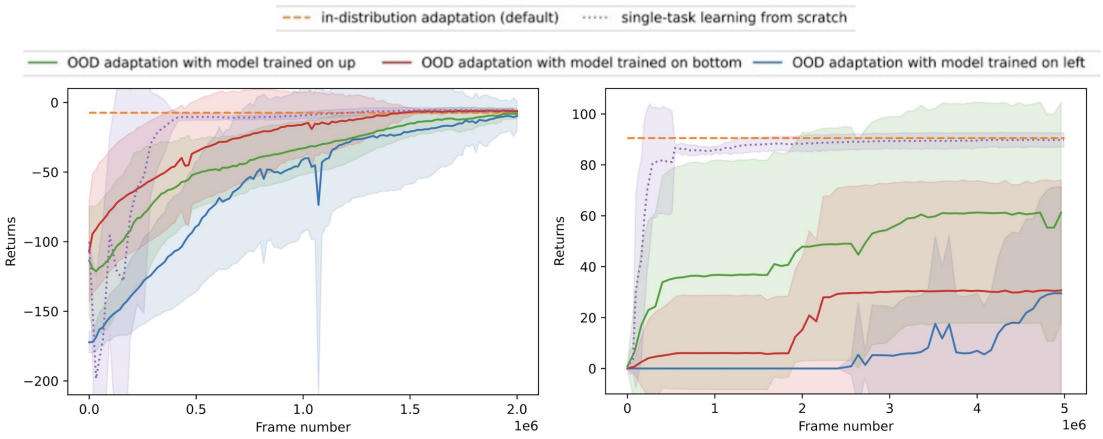
These results are not specific to variBAD, but are expected to apply to most current approaches for Fast Adaptation (see results for RL^2 and MAML in Xiong et al. (2021)). They illustrate the immense challenge of dealing with distribution shifts, and the need for methods that can detect such a shift, and address it appropriately.

Solutions Not much research has been done on explicitly dealing with distribution shifts in the Meta-Learning for Fast Adaptation setting, but some promising initial successes have been achieved on Gridworld and MuJoCo tasks with disjoint training and test distributions (Lee et al., 2021; Mendonca et al., 2020; Fakoor et al., 2020). These methods mostly rely on *generating* data that is different from the training tasks distribution, either during meta-training to train an agent that can generalise to more tasks than it has seen (Lee et al., 2021), or at meta-test time to improve sample complexity (Mendonca et al., 2020; Fakoor et al., 2020).



(a) Different Training Distributions **(b)** Test Distribution

Figure 9.1: Experimental set-up for dealing with shifts between training and task distributions. We use 4 different meta-training distributions (a) and one meta-test distribution (b). Training and testing on p_{right} is the set-up we used throughout the thesis. Training on p_{left} / p_{up} / p_{down} and testing on p_{right} requires the agent to generalise or adapt to tasks that are very different from what it has seen during meta-training.



(a) Dense Rewards **(b)** Sparse Rewards

Figure 9.2: Illustration of adaptation performance when the meta training and test distributions differ, $p_{train} \neq p_{test}$. We trained four variBAD agents separately on four different task distributions: goals on the left/top/right/bottom from the agent's starting position, $p_{train} \in \{p_{left}, p_{up}, p_{right}, p_{down}\}$ as shown in Figure 9.2 (three seeds each). At meta-test time, we adapt them on test tasks that lie only on the right, $p_{test} = p_{right}$. The variBAD agent where $p_{train} = p_{test} = p_{right}$ can adapt within a single episode (dashed line at top). The other agents however need more time to adapt. To this end, we adapt them on single test-tasks using gradient descent on the entire model (policy and encoder). Surprisingly, pre-training an agent on a slightly different distribution gives us a *disadvantage* compared to training from scratch: it takes them longer to learn the test task (Fig 9.1a). When rewards are sparse, it is even worse: some agents do not recover at all (Fig 9.1b), most likely due to a lack of reward signals since the right side is never explored. These results were published in Xiong et al. (2021).

For next steps, one possible research goal is to meta-train agents that can generalise well, so that at test time we can simply do the standard inner loop update (e.g., doing a few gradient steps in CAVIA or unrolling the encoder and policy in variBAD). One way to achieve this is with innovations in the parameterisation of the policy or learning algorithm. Out-of-domain generalisation has been successful in the Long Horizon Meta-Learning literature (where the agent is trained from scratch for hundreds or more episodes) by introducing architecture symmetries (Kirsch et al., 2022) or learning part of an update rule (Oh et al., 2020). In the Fast Adaptation case some initial promising results on tasks that are outside the meta-training distribution were obtained when using hypernetworks to generate policies at meta-test time (Sarafian et al., 2021). Another way to meta-train agents that can generalise better is to train them on many more environments (such as procedurally generated ones like Cobbe et al. (2019) and Küttler et al. (2020)).

Alternatively, we can focus our efforts on developing ways to continue learning (beyond what the inner-loop dictates) at meta-test time. To do so, the agent must decide when, and for how long, to continue learning on the new task. Doing this without human input means that the agent should detect whether a distribution shift has occurred, and somehow quantify its uncertainty about the new tasks. This could be done, e.g., by combining Meta-RL methods with uncertainty quantification (van Amersfoort et al., 2021; Osband et al., 2018). An open question is which kind of approaches and architectures are amenable to fast adaptation at test time, since, as we have seen above in Figure 9.4a, simply using current algorithms and continuing to train them via gradient descent is likely not sufficient.

9.3 Building Better Belief Models

Chapters 6-8 rely on belief modelling, so that the agent can take uncertainty into account when making decisions. To this end, we developed a novel training objective for a sequential VAE when the reward / transition function changes (Chapter 6) or when there are other agents in the environment whose strategies are unknown (Chapter 7). We saw that our belief model adequately captures the belief (Chapters

6 and 7), allows the agent to act approximately Bayes-optimally (Chapters 6-8), and can be used for meta-exploration in hyper-state space (Chapter 8).

In the following, we illustrate two concrete open research problems that follow from our work using sequential VAEs for belief modelling.

Generating Typical Samples A shortcoming of our current VAE models is that it does not necessarily provide *typical* samples, but rather outputs expected values. A typical sample is one that has high probability under the true distribution: e.g., if we sample a reward function $R \sim q_\phi(R|\tau_{:t})$ from the approximate posterior (by sampling a latent vector and passing it through the decoder), we want the probability of this sample under the true task distribution $p(R|\tau_{:t})$ to be high. To understand what this means using an example, consider again the Gridworld from Section 6.3.1 in Chapter 6, and let us take a closer look at the kind of samples our belief model produces when sampling from the prior. We know that for this task distribution, the prior distribution over reward functions is such that one of the (admissible) cells in the Gridworld has a reward of one ($r = 1$), and all other cells have a reward of zero ($r = 0$). I.e., each sample from the prior distribution of environments would give us one such reward function. The expected value for the rewards in each cell is $\mathbb{E}[r] = \frac{1}{20}$, since there are 20 cells where a goal can be.

Next, let us have a look at the types of predictions we get for samples from the meta-learned prior, from a fully trained variBAD VAE model. To this end, we take samples from the prior and pass it through the VAE decoder to predict rewards for all cells in the Gridworld. This is what Figure 9.3 shows, once for the case where we train the model with a Sigmoid at the output (which we used throughout our experiments in Chapter 6), and once when training with a Softmax (using our domain knowledge that there can only be one goal at a time).

The first thing we observe is that most samples (in red) are not *typical*: i.e., they predict some average reward value for all cells, rather than typical reward functions where the reward is 1 in one cell and 0 otherwise. The tendency of (vanilla) VAE decoders to produce outputs that are close to expected values, rather than typical

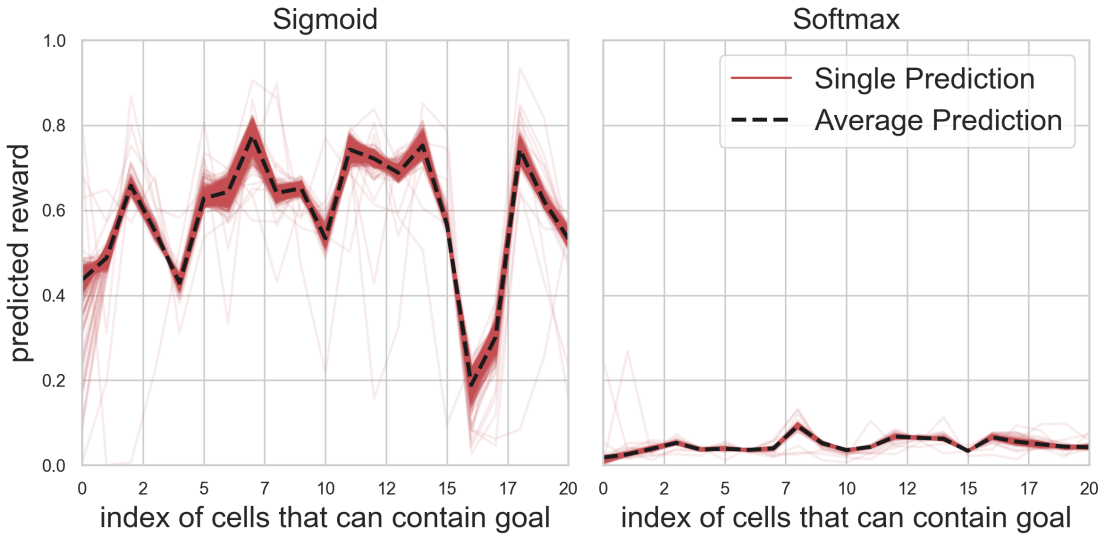
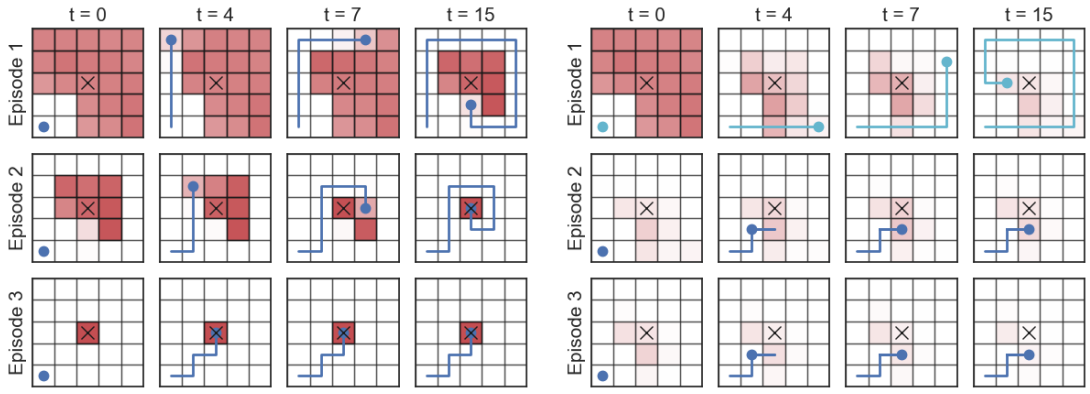


Figure 9.3: VAE reward predictions in the Gridworld environment, for different samples from the prior, when trained with a Sigmoid (left) or trained with a Softmax (right) at the decoder output layer. We know that the true reward structure is such that the reward equals 1 at exactly one cell, and 0 otherwise. This figure shows that the VAE does not output *typical* samples that reflect this reward structure. Instead, the decoder predicts values closer to the expected value, which is a known problem for vanilla VAEs (Dumoulin et al., 2016; Cemgil et al., 2020). It is also not calibrated well, which we hypothesise is because the VAE overfits to the policy's induced state distribution (which is not uniform across all states). This is not problematic for meta-training a policy: all the information is still there, i.e., which goal positions are still possible and which are excluded (see Figure 6.3a in Chapter 6), where we see that the posterior goes to zero for empty cells the agent has visited), It is however problematic if we want the environment model, e.g., to do planning.

samples, is a known phenomenon in the VAE literature (Dumoulin et al., 2016; Cemgil et al., 2020), and the reason that VAEs tend to produce blurry images compared to, e.g., generative adversarial networks (Goodfellow et al., 2014). In our experiments, this phenomenon was never problematic: all the information that the policy needs for approximately Bayes-optimal decision-making is still captured in the approximate belief predicted by the VAE. If, however, we do want to make use of the decoder, e.g., to do planning, then we do want typical samples. To this end, follow-up work may adapt solutions from the VAE literature (Cemgil et al., 2020) to the sequential setting.

Shifts in the Policy In Chapters 6-8 we train a belief model using a sequential VAE alongside a policy that conditions on the learned beliefs. Therefore they depend on each other: the policy learns to interpret the VAE's latent beliefs, and the VAE learns to encode the policy's trajectories. But this also means that if the agent acts differently at test time, the VAE might not model beliefs appropriately. To test this, we ran the following experiment. We trained a variBAD model to convergence on the Gridworld from Section 6.3.1, and evaluated (a) the meta-trained variBAD model, compared to (b) a hand-coded *different* Bayes-optimal policy at test time in the first episode, with the variBAD model taking over from episode 2. Figure 9.4 shows the behaviour and beliefs (in red) for these two policies. VariBAD does not adequately model the belief for the hand-coded policy (first row in Fig 9.4b), likely because the VAE and policy have overfit to each other. This will cause problems if we cannot guarantee joint deployment (e.g., if we combine learned policies with human agents that sometimes take over). Possible ways to overcome this is to train ensembles of VAEs and/or policies, build more robust belief models, continue updating the VAE at meta-test time if a distribution shift (in the policy or environment) is detected, or build on innovations of dealing with distribution shifts from other research areas (Antonova et al., 2020; Lee et al., 2020; Rahimian et al., 2019; Zhang et al., 2020).



(a) VariBAD Model **(b)** OOD Model in Episode 1

Figure 9.4: Effects of changing the policy at meta-test time. (a) Behaviour and beliefs for variBAD. (b) Behaviour and beliefs when rolling out a hand-coded policy in episode 1, and the variBAD policy thereafter. The VAE does not adequately update the belief for a policy that it has not been meta-trained together with, which in turn means as the variBAD policy takes over in episode 2 (second row), it does not find the goal.

Looking forward, we believe that developing more specialised belief models will be necessary to address new challenges in Fast Adaptation. For example, if we move towards more complex meta-training distributions such as multi-modal or tailed distributions, we might exhaust our current belief models' capabilities. Another challenge that might occur in some tasks are noisy, distracting, or generally complex (like image-based) observations. A simple way to deal with this in the variBAD (Chapter 6) or MeLIBA (Chapter 7) models is to simply backpropagate the RL loss through the encoder, to inform it what is "useful" to encode in the belief with regards to the agent. However, this might become more difficult to optimise, we need to tune more hyper-parameters, and training can be slowed down. Finding better ways to inform belief modelling of "usefulness", or dealing with distracting observations, is an interesting avenue of future research.

With many open research questions, in particular on belief modelling, only time and new challenging benchmarks will allow us to pinpoint the exact challenges that are important and which ones should therefore be prioritised.

10 Conclusion

This thesis addressed the challenge of Fast Adaptation using Meta-Learning, which we defined and motivated in Chapter 3. We argued that quickly learning new tasks should be conceptually divided into "task inference" and "task solving", since this might make optimisation easier and the individual parts can specialise better or be used in downstream tasks. To implement this idea, we proposed a context-based approach, where the model (e.g., a classifier or a policy) conditions on a "context vector" that represents the task. In Chapter 5, we first used a deterministic context that was learned via gradient descent in the Supervised Learning setting (Section 5.2), to establish that this can indeed improve model performance and that the context can adequately capture the task. We then moved to the Reinforcement Learning setting (Section 5.3), where we also observe an improvement when updating only the context when learning new tasks as opposed to the entire model as in MAML (Finn et al., 2017a).

Since our ultimate goal was to do Online Adaptation, i.e., have agents that can perform well from the very first time they interact with an environment to learn a new task, we then moved beyond the deterministic approach and gradient-based adaptation. Instead, we introduced Bayesian reasoning over the context vector, coupled with a model that can update this context vector on-the-fly using

recurrent neural networks instead of gradient updates. By combining the context-based idea, ideas from Meta-RL that use recurrent networks that have access to previous actions and rewards for learning, and approximate variational inference, we developed methods that can compute approximately Bayes-optimal agents for the single-agent setting in Chapter 6 and the multi-agent setting in Chapter 7. Finally, Chapter 8 addressed a big open challenge in Meta-RL: learning with sparse rewards. This is important since many real world tasks are best formulated with sparse rewards, like a binary success criterion for completing a task. Compared to crafting dense reward signals, this is easier, less time-consuming, and less prone to reward hacking (Abbeel et al., 2004; Hadfield-Menell et al., 2017). We observed that existing Meta-RL methods can fail entirely if rewards are sparse, and proposed one way to overcome this by exploring in Hyper-State Space during meta-training.

The contributions in this thesis significantly advance the field of Fast Adaptation in Meta-RL. The agents developed in this thesis can adapt faster than any previous methods across a variety of tasks, and we can compute approximately Bayes-optimal policies for much more complex task distributions than we were previously able to. Follow-up work inspired by our contributions has looked at using variBAD (Chapter 6) in the offline RL setting (Dorfman et al., 2021) (which is another useful step towards real world deployment of RL, see the discussion in Section 1.1), and work that addresses the same sparse reward problem as HyperX (Chapter 8), by developing new rewards structures that take into account value of currently available information (Ambrogioni, 2021).

The work on developing agents that can autonomously learn like humans has only just begun, and there is exciting research left to be done in the years and decades to come. We hope that we as a research community can do so responsibly, possibly using AI to address some of the most pressing challenges of our time, in a way that benefits our planet and its inhabitants.

Appendices

A

Appendix for Chapter 5, CAVIA

A.1 Implementation - Practical Tips

The context parameters ξ can be added to any network, and do not require direct access to the rest of the network weights like MAML. In PyTorch this can be done as follows. To add CAVIA parameters to a network, it is necessary to first initialise them to zero when the model is initialised:

```
self.context_params = torch.zeros(size=[self.num_context_params],
requires_grad=True)
```

Add a way to reset the context parameters to zero (e.g., a method that just does the above). During the forward pass, add the context parameters to the input by concatenating it (when using a fully connected network):

```
x = torch.cat((x, self.context_params.expand(x.shape[0], -1)), dim=1)
```

(This is for fully connected networks. We refer the reader to our implementation for how to use FiLM to condition CNNs.) To correctly set the computation graph for the outer loop, it is necessary to assign the context parameters manually with their gradient. In the inner loop, compute the gradient:

```
grad = torch.autograd.grad(task_loss, model.context_params,
create_graph=True)[0]
```

The option *create_graph* makes sure that you can take the gradient of *grad* again. Then, update the context parameters using one gradient descent step

```
model.context_params = model.context_params - lr_inner * grad
```

If you now do another forward pass and compute the gradient of the model parameters θ (for the outer loop), these include higher order gradients because *grad* above includes gradients of θ, and because we kept the computation graph via the option *grad*. To see how to train CAVIA and aggregate the meta-gradient over several tasks, see our code at `https://github.com/lmzintgraf/cavia`.

A.2 Experiment Details

Hyperparameter Selection The choice of network architecture/size and context parameters can be guided by domain knowledge. E.g., for the few-shot image classification problem, an appropriate model is a deep convolutional model. For the context parameters, it is important to make sure they are not underparameterised. CAVIA can deal with larger than necessary context parameters (see Table 5.1), although it might start overfitting in the inner loop at some point (we have not experiences this in practise). Regarding learning rates, we always started with an inner loop learning rate of 1 and the Adam optimiser with the standard learning rate of 0.001 for the outer loop.

For CNNs, we found that adding the context parameters not at the input layer, but after several (in our case after the third out of four) convolutions works best. We believe this is because the lower-level features that the first convolutions extract are useful for any image classification task, and we only want our task embedding to influence the activations at the deeper layers. In our experiments we used a FiLM network with no hidden layers. We tried deeper versions, but this resulted in inferior performance.

We also tested to add context parameters at several layers instead of only one. However, in our experience this resulted in similar (regression and RL) or worse (in the case of CNNs) performance.

Mini-Imagenet Hyperparameters For Mini-Imagenet, our model takes as input images of size $84 \times 84 \times 3$ and has 5 outputs, one for each class. The model has four modules that each consist of: a $2D$ convolution with a 3×3 kernel, padding 1 and 128 filters, a batch normalisation layer, a max-pooling operation with kernel size 2, if applicable a FiLM transformation (only at the third convolution, details below), and a ReLU activation function. The output size of these four blocks is $5 \times 5 \times 128$, which we flatten to a vector and feed into one fully connected layer. The FiLM layer itself is a fully connected layer with inputs ϕ and a 256-dimensional output and the identity function at the output. The output is divided into γ and β, each of dimension 128, which are used to transform the filters that the convolutional operation outputs. The context vector is of size 100 (other sizes tested: 50, 200) and is added after the third convolution (other versions tested: at the first, second or fourth convolution).

The network weights are initialised using He et al. (2015), the bias parameters are initialised to zero (except at the FiLM layer). We use the Adam optimiser for the meta-update step with an initial learning rate of 0.001. This learning rate is annealed every $5,000$ steps by multiplying it by 0.9. The inner learning rate is set to 0.1 (others tested: 1.0, 0.01). We use a meta batchsize of 4 and 2 tasks for 1-shot and 5-shot classification respectively. For the batch norm statistics, we always use the current batch – also during testing. I.e., for 5-way 1-shot classification the batch size at test time is 5, and we use this batch for normalisation.

A.3 Additional Results: CelebA Image Completion

The following images show additional results for the CelebA image completion task.

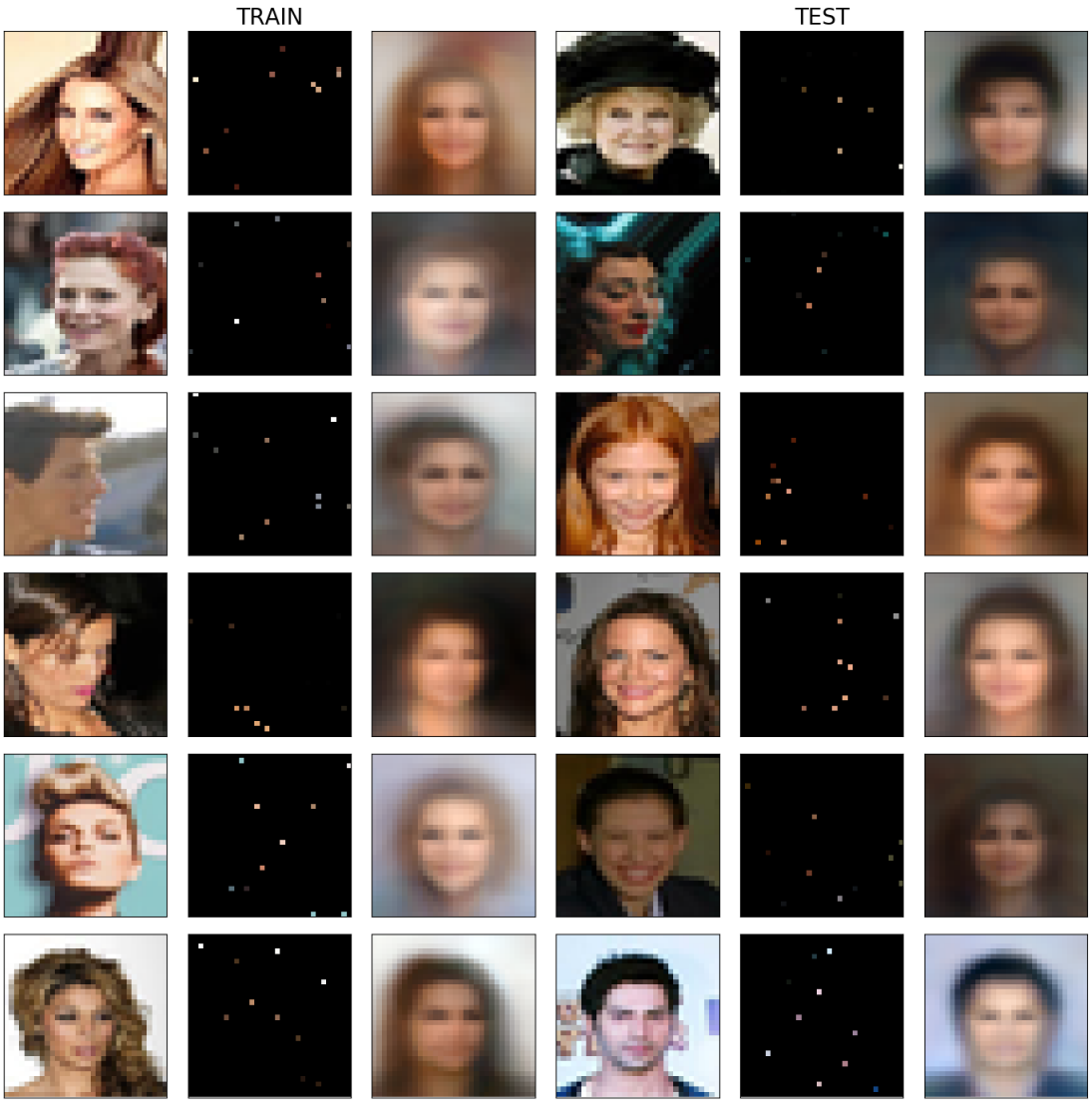


Figure A.1: Additional image completion results for the CelebA image completion problem, when $k = 10$ pixels are given.

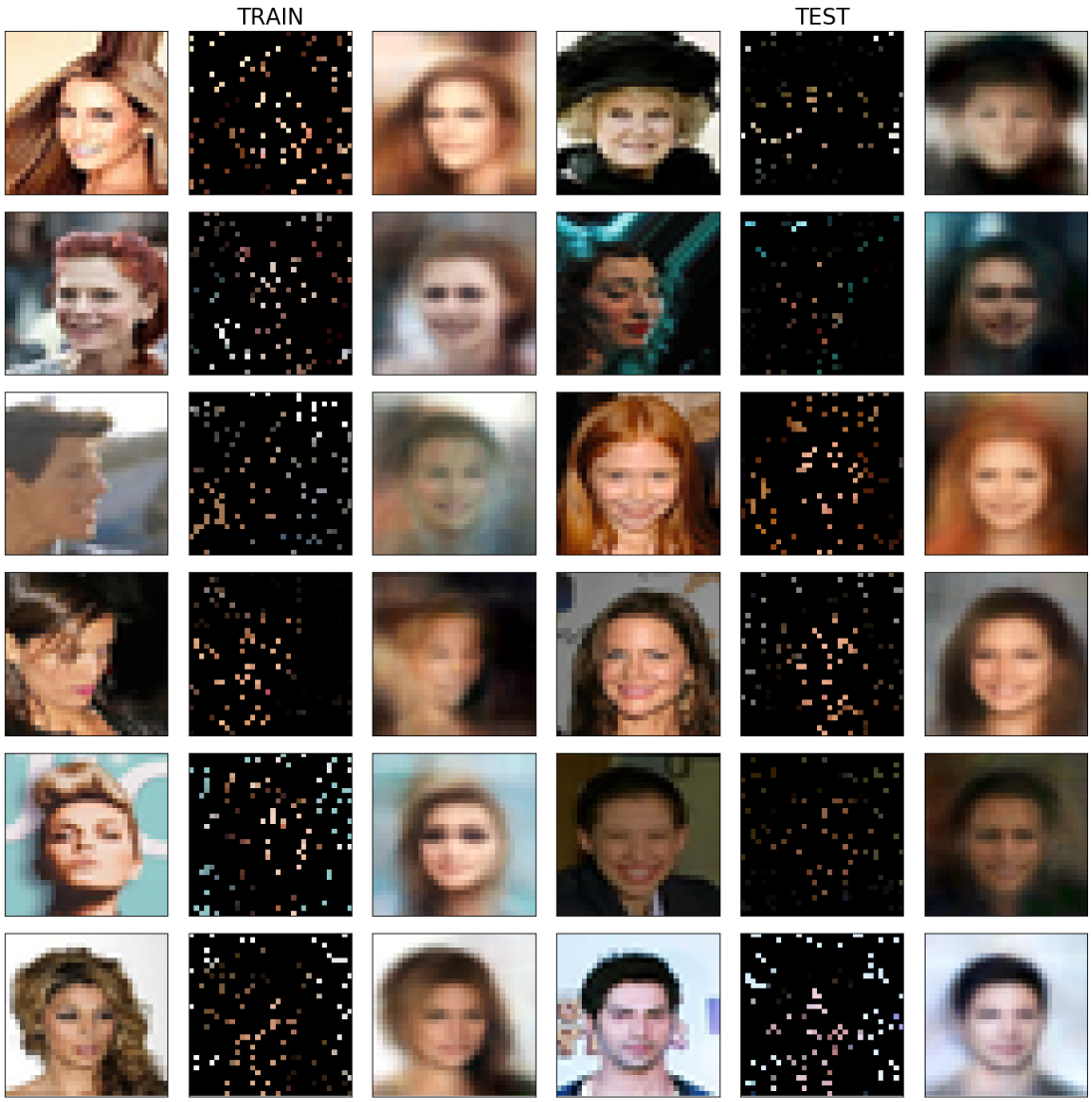


Figure A.2: Additional image completion results for the CelebA image completion problem, when $k = 10$ pixels are given.

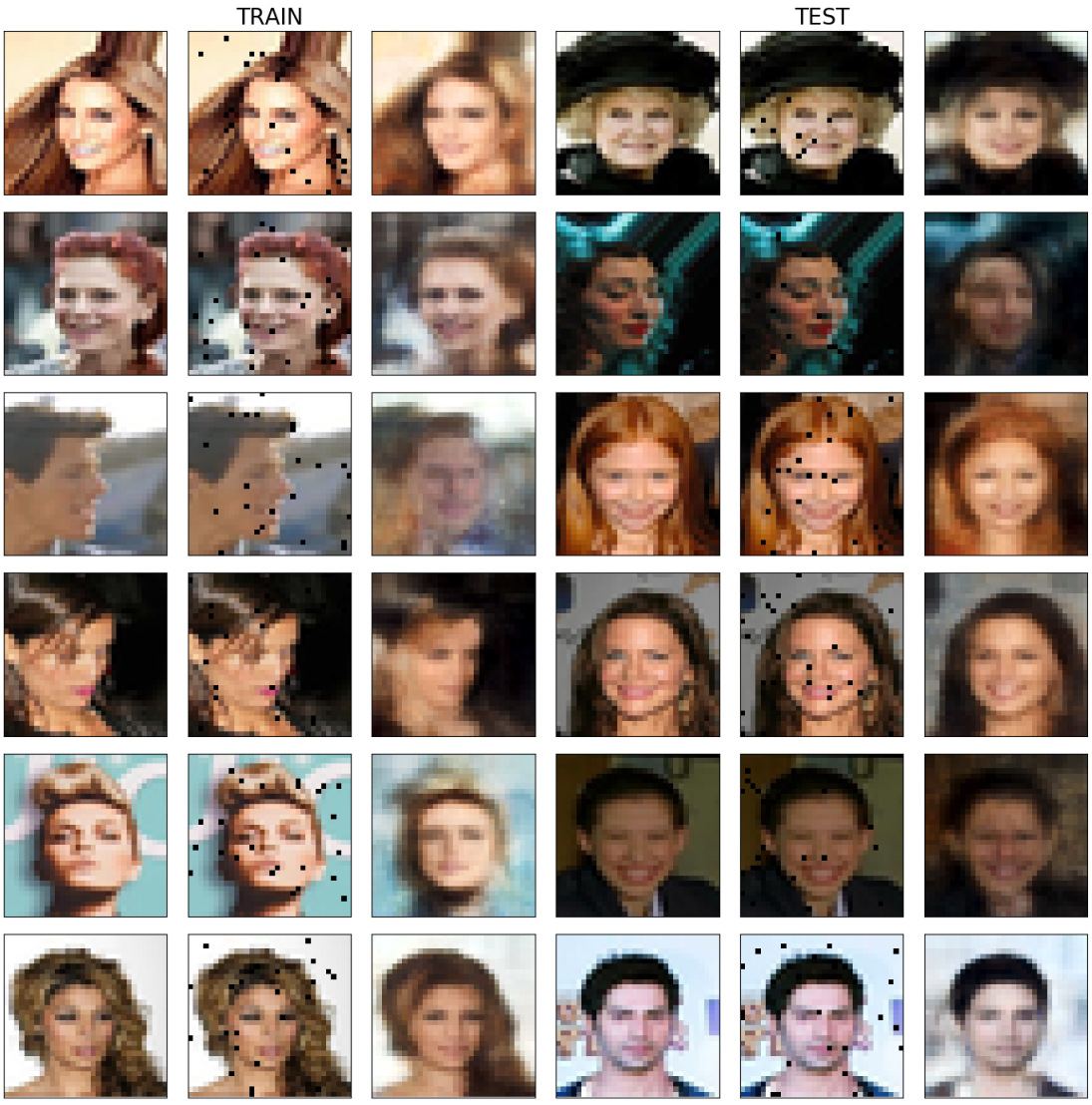


Figure A.3: Additional image completion results for the CelebA image completion problem, when $k = 10$ pixels are given.

B

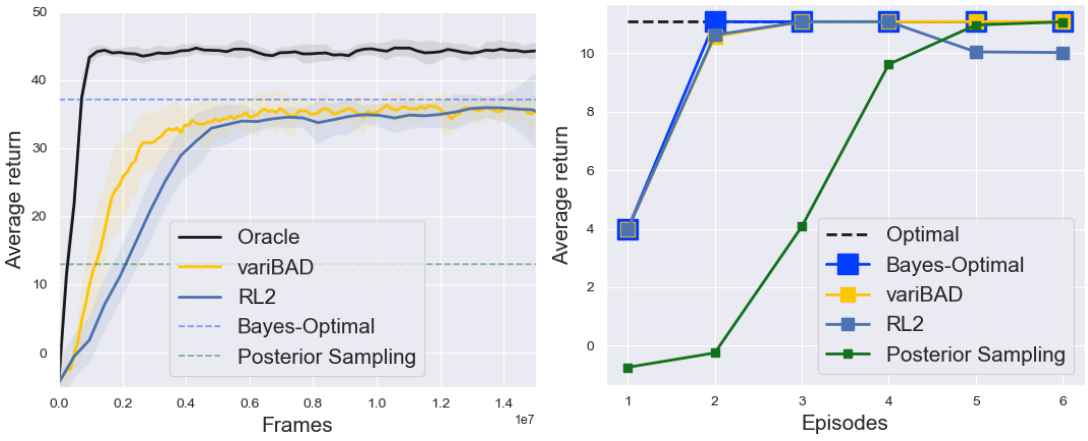
Appendix for Chapter 6, VariBAD

B.1 Experiments: Gridworld

Here we provide additional remarks and figures for the variBAD Gridworld results from Section 6.3.1.

Additional Remarks Figure 6.3c 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 0 and variance 1, 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.

Comparison to RL^2 Figure B.1a shows the learning curves for variBAD and RL^2, in comparison to a multitask policy (which has access to the goal position). We trained these policies on a horizon of $H^+ = 4 \times H = 60$, i.e., on a BAMDP in which the agent has to maximise online return within four episodes. We indicate the values of a hard-coded Bayes-optimal policy, and a hard-coded posterior sampling policy using dashed lines.



(a) Learning curves. **(b)** Average return per test episode.

Figure B.1: Results for the Gridworld toy environment. (a) Learning curves over meta-training. (b) Per-episode online return at meta-test time. VariBAD and RL^2 were trained to maximise the online return across 4 episodes. VariBAD can extrapolate to 6 episodes, but RL^2 is unstable. Results are averages over 20 seeds (with 95% confidence intervals for the learning curve).

Figure B.1b shows the end-performance of variBAD and RL^2, compared to the hard-coded optimal policy (which has access to the goal position), Bayes-optimal policy, and posterior sampling policy. VariBAD and RL^2 both closely approximate the Bayes-optimal solution. By inspecting the individual runs, we found that variBAD learned the Bayes-optimal solution for 4 out of 20 seeds, RL^2 zero times. Both otherwise find solutions that are very close to Bayes-optimal, with the difference that during the second rollout, the cells left to search are not all on the shortest path from the starting point.

VariBAD and RL^2 were trained on 4, and evaluated on 6 episodes. After the fourth rollout, we do *not* fix the latent / hidden state, but continue rolling out the policy as before. We find that the performance of RL^2 drops again after the fourth episode: this is likely due to instabilities in the 128-dimensional hidden state. VariBAD's latent representation, the approximate task posterior, is concentrated and does not change with more data.

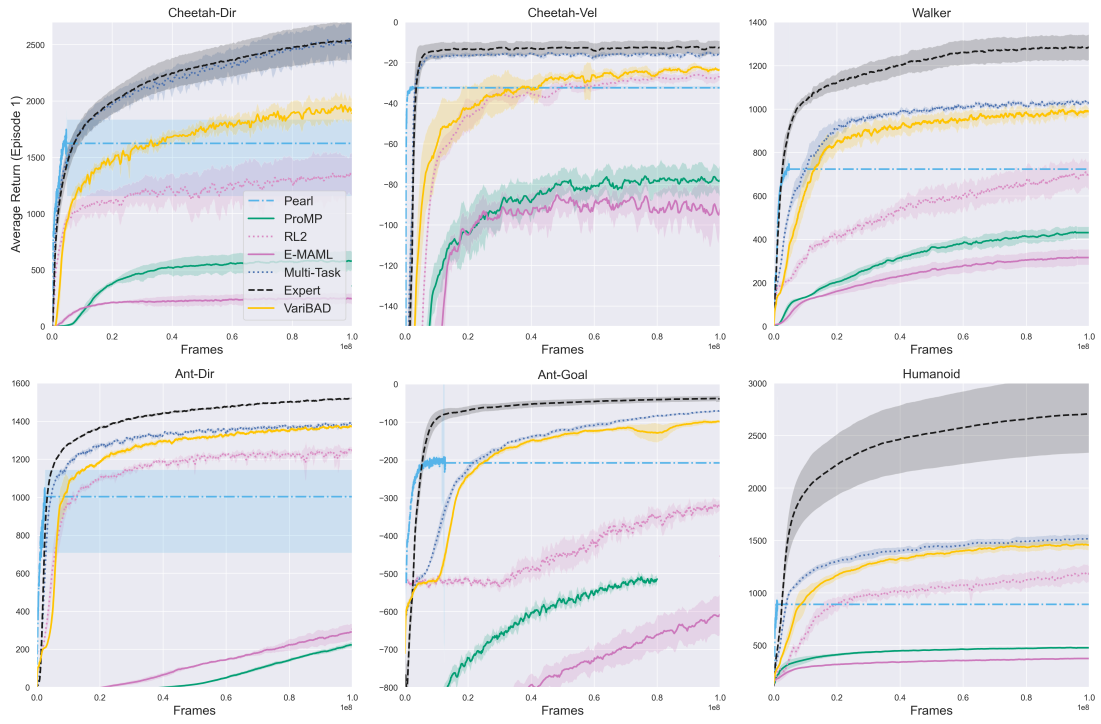
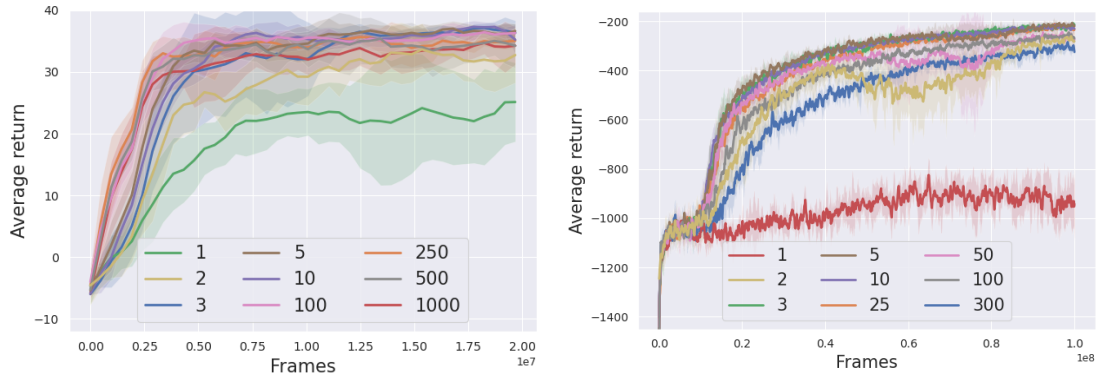


Figure B.2: Learning curves for the MuJoCo results presented in Section 6.3.3. The plots show the performance at the N-th rollout. For variBAD and RL^2, $N = 2$. For PEARL, $N = 10$. For ProMP and E-MAML, $N = 30 - 60$ (3 gradient steps on rollouts of length 10-20 depending on the environment).

B.2 Experiments: MuJoCo

In this section we provide the learning curves for the MuJoCo environments from Section 6.3.3 (Fig B.2) and runtime comparisons. We also provide additional analyses of how performance scales with the latent dimension (Figure B.3), the learned agent behaviour (Figure B.4),how the latent space behaves at test time (Figure B.5), and performance when backpropagating the RL loss through the VAE encoder (Figure B.6).

Learning Curves Figure B.2 shows the learning curves for the MuJoCo environments for all approaches. The multitask and expert policies were trained using PPO. 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).



(a) Gridworld. **(b)** AntGoal.

Figure B.3: Learning curves for different VAE latent dimensions, for the Gridworld environment (a) and the MuJoCo AntGoal environment (b).

As we can see, PEARL is much more sample efficient in terms of number of frames than the other methods (Fig B.2), 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 has been done in Dorfman et al. (2021). 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).

For all MuJoCo environments, we trained variBAD with a reward decoder only (even for Walker, where the dynamics change, we found that this has superior performance).

CheetahDir Test Time Behaviour To get a sense for where these differences between the different approaches might stem from, consider Figure B.4 which shows example behaviour of the policies during the first three rollouts in the HalfCheetahDir environment, for the task “go left". 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. For a visualisation of the variBAD latent space at test time for this environment see Figure B.5.

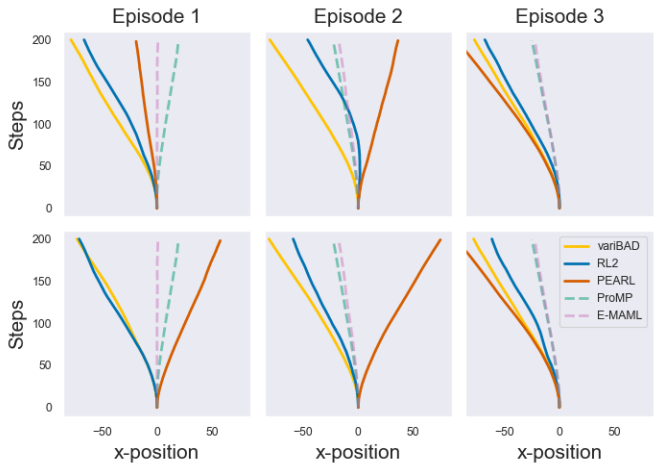


Figure B.4: Test time behaviour for the task "walk left" in HalfCheetahDir. The x-axis reflects the agent's position; the y-axis the environment steps (to be read from bottom to top). Rows are separate examples, columns the number of rollouts.

Latent Space Visualisation A nice feature of variBAD is that it can give us insight into the uncertainty of the agent about what task it is in. Figure B.5 shows the latent space for the HalfCheetahDir tasks "go right" (top row) and "go left" (bottom row). We observe that the latent mean and log-variance adapt rapidly, within just a few environment steps (left and middle figures). This is also how fast the agent adapts to the current task (right figures). As expected, the variance decreases over time as the agent gets more certain. It is interesting to note that the values of the latent dimensions swap signs between the two tasks.

Visualising the belief in the reward/state space directly, as we have done in the Gridworld example, is more difficult for MuJoCo tasks, since we now have continuous states and actions. What we could do instead, is to additionally train a model that predicts a ground-truth task description (separate from the main objective and just for further analysis, since we do not want to use this privileged information for meta-training). This would give us a more direct sense of what task it thinks it is in.

Runtime Comparison The following are rough estimates of average run-times for the HalfCheetahDir 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).

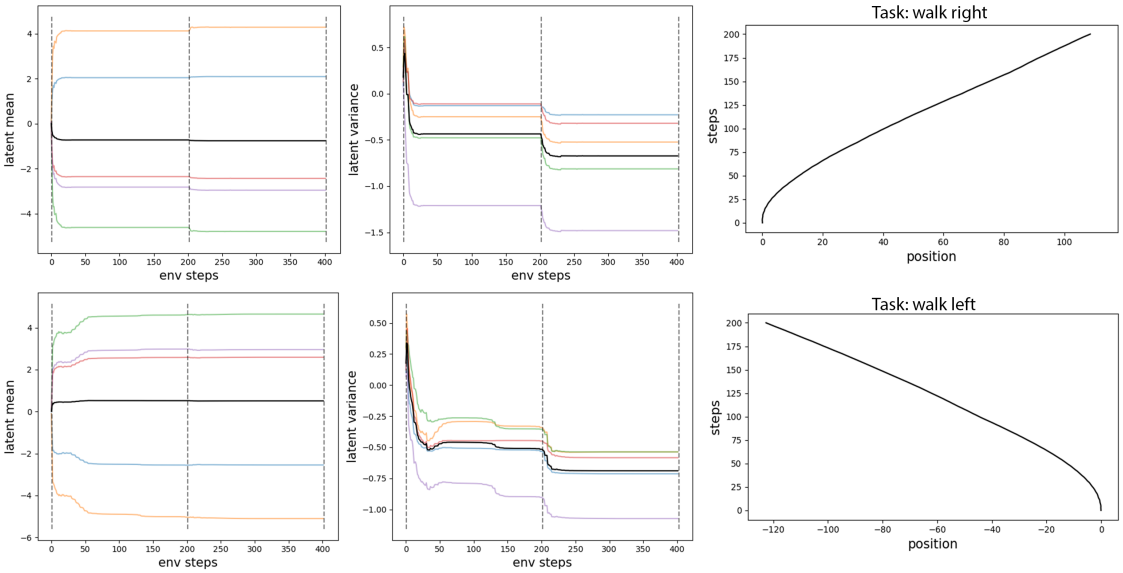
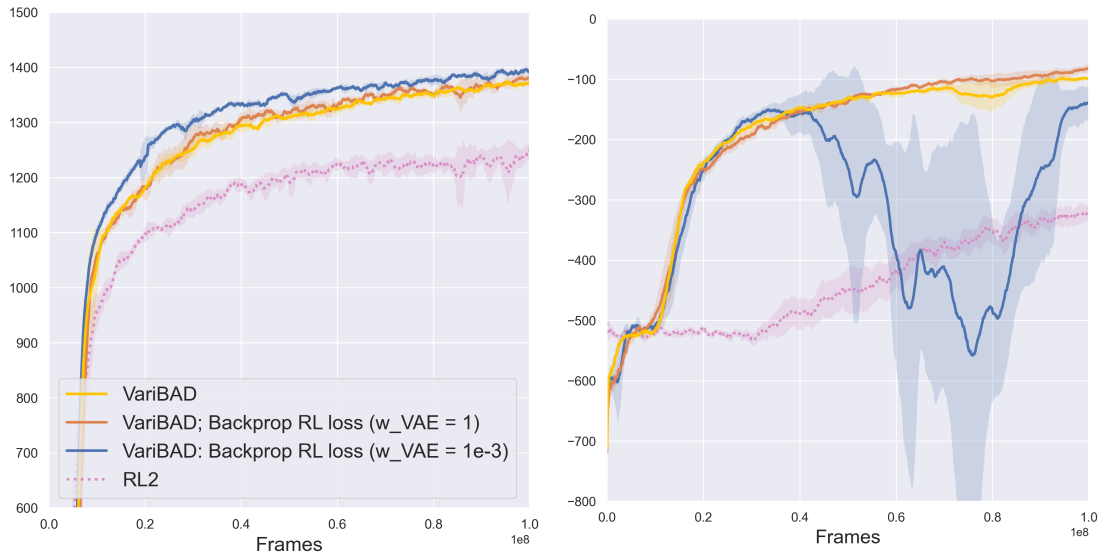


Figure B.5: Visualisation of the latent space at meta-test time, for the HalfCheetahDir environment and the tasks "go right" (top) and the task "go left" (bottom). Left: value of the posterior mean during a single rollout (200 environment steps). The black line is the average value. Middle: value of the posterior log-variance during a single rollout. Right: Behaviour of the policy during a single rollout. The x-axis show the position of the Cheetah, and the y-axis the step (should be read from bottom to top).

- 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 than RL^2 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). This difference would likely be less if we used a different RL algorithm which does not rely on this many forward/backward passes per policy update.



(a) AntDir **(b)** AntGoal

Figure B.6: Learning curves for the MuJoCo AntDir (a) and AntGoal (b) environments, when backpropagating the RL loss through the encoder. Depending on the task, this can help (a) or hurt (b) performance, and critically depends on the relative weight between VAE and RL loss.

Ablation Study: Backpropagating the RL loss Through the Encoder In the main experiments presented in the paper, we do not backpropagate the RL loss through the encoder. Instead, we alternate between updating the VAE with the ELBO loss, and updating the policy with the PPO loss (and detaching the gradient when feeding the belief into the policy). We do so because this performs sufficiently well, and has two advantages: first, we do not have to calibrate the relative weighting between the RL and the VAE loss; second, it is much faster in practice because otherwise we would have to re-compute the belief embedding for each PPO mini-batch.

Figure B.6 shows the learning curves of the variBAD agent in the AntDir and AntGoal environments, when backpropagating the RL loss compared to our standard setting and compared to RL^2. These results show that sometimes, combining the RL and VAE loss can marginally improve performance (Figure B.6a), but it can also significantly hurt performance (Figure B.6b) if the relative weighting is not calibrated correctly. In terms of experiment runtime, when backpropagating the RL loss through the encoder is as slow as RL^2 (around 66% slower in these environments).

B.3 Experiments: Meta-World

Figure B.7 shows the learning curves for the Meta-World ML1 tasks for variBAD (20 seed averages with 95% confidence intervals). We followed the evaluation protocol of Yu et al. (2019).

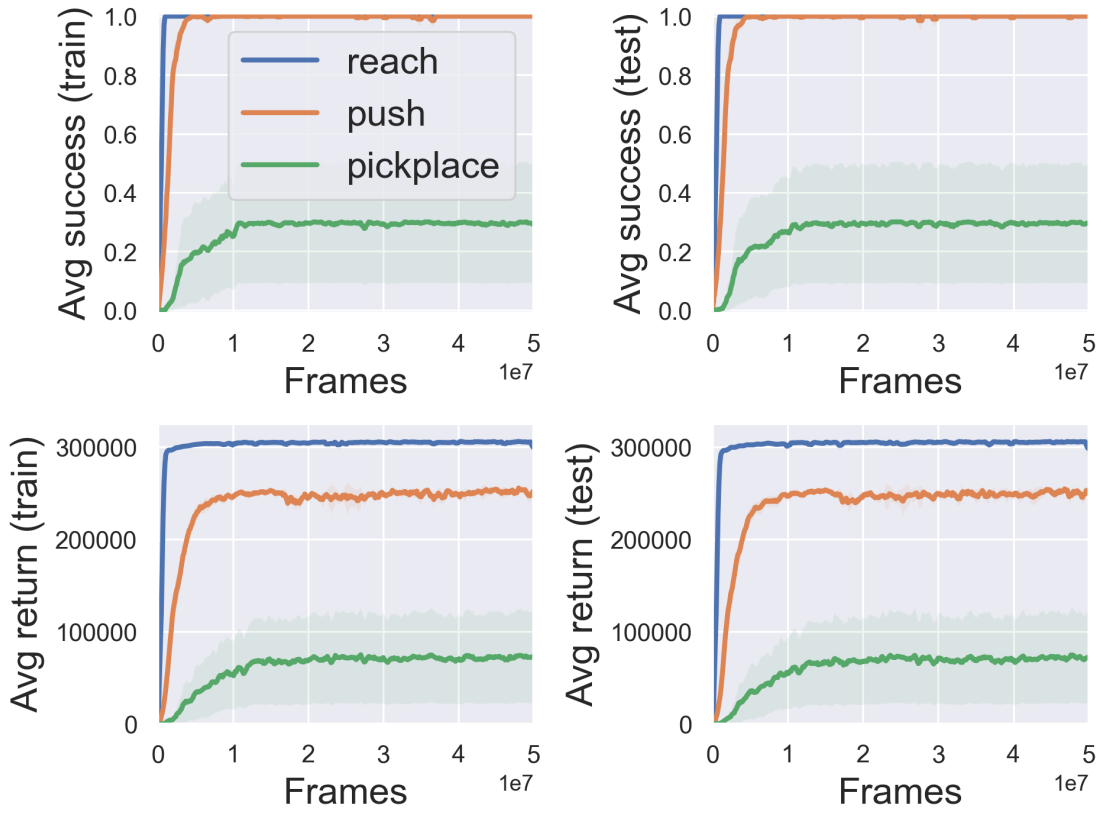


Figure B.7: Learning curves for the different ML1 Meta-World tasks. Shown are the success rates (top row) and average returns (bottom row) for the training set (first column) and test set (second column).

B.4 Hyperparameters

We used the PyTorch framework (Paszke et al., 2017) for our experiments. The hyperparameters for Gridworld, MuJoCo CheetahDir, PointRobot and MetaWorld ML1-Push can be found in the tables below. For more details, see our reference implementation at `https://github.com/lmzintgraf/varibad`.

We used different number of seeds per experiment to balance significance of results and computational required, due to the inherent randomness/difficulty of differ-

ent tasks. For the main experiments, we used 20 seeds for Gridworld/Navigation/Meta-World, and 10 seeds per MuJoCo environment. For the ablation studies, we used fewer for MuJoCo (5 instead of 10), and Gridworld (15 instead of 20) due to computational constraints.

	Grid World	Cheetah Dir	Point Robot	ML1 Push
max_rollouts_per_task	4	2	3	3
policy_state_embedding_dim	16	64	64	64
policy_latent_embedding_dim	16	64	64	64
norm_state_for_policy	True	True	True	True
norm_latent_for_policy	True	True	True	True
norm_rew_for_policy	True	True	True	True
norm_actions_pre_sampling	False	True	False	False
norm_actions_post_sampling	False	False	False	False
policy_layers	[32]	[128, 128]	[128, 128, 128]	[128, 128, 128]
policy_activation_function	tanh	tanh	tanh	tanh
policy_initialisation	normc	normc	normc	normc
policy_anneal_lr	False	False	False	False
policy	ppo	ppo	ppo	ppo
policy_optimiser	adam	adam	adam	adam
ppo_num_epochs	2	16	2	2
ppo_num_minibatch	4	4	8	8
ppo_clip_param	0.05	0.1	0.1	0.1
lr_policy	0.0007	0.0007	0.0007	0.0007
num_processes	16	16	16	16
policy_num_steps	60	800	200	200
policy_eps	1e-08	1e-08	1e-08	1e-08
policy_value_loss_coef	0.5	0.5	0.5	0.5
policy_entropy_coef	0.01	0.01	0.001	0.001
policy_gamma	0.95	0.97	0.99	0.99
policy_use_gae	True	True	True	True
policy_tau	0.95	0.9	0.9	0.9
use_proper_time_limits	False	True	True	True
encoder_max_grad_norm	None	1.0	None	None
decoder_max_grad_norm	None	1.0	None	None
lr_vae	0.001	0.001	0.001	0.001
size_vae_buffer	100000	10000	10000	10000
precollect_len	5000	5000	5000	5000
vae_buffer_add_thresh	1	1	1	1
vae_batch_num_trajs	25	10	15	15
tbptt_stepsize	None	50	None	None
vae_subsample_elbos	None	50	None	None
vae_subsample_decodes	None	50	None	None

	Grid World	Cheetah Dir	Point Robot	ML1 Push
vae_avg_reconstruction_terms	False	False	False	False
num_vae_updates	3	1	3	3
pretrain_len	0	0	0	0
kl_weight	0.01	0.1	1.0	1.0
action_embedding_size	0	16	16	16
state_embedding_size	8	32	32	32
reward_embedding_size	8	16	16	16
encoder_layers_before_gru	[]	[]	[]	[]
encoder_gru_hidden_size	64	128	128	128
encoder_layers_after_gru	[]	[]	[]	[]
latent_dim	5	5	5	5
decode_reward	True	True	True	True
rew_loss_coeff	1.0	1.0	1.0	1.0
input_prev_state	False	True	False	False
input_action	False	True	False	False
reward_decoder_layers	[32, 32]	[64, 32]	[64, 32]	[64, 32]
decode_state	False	False	False	False
state_loss_coeff	1.0	1.0	1.0	1.0
state_decoder_layers	[32, 32]	[64, 32]	[64, 32]	[64, 32]
disable_kl_term	False	False	False	False
rlloss_through_encoder	False	False	False	False

C

Appendix for Chapter 7, MeLIBA

C.1 ELBO Derivation

We derive the ELBO for modelling the future actions of agent i as follows.

$$\log\ p(u^{(i)}_{t:H}|s_t) \tag{C.1}$$

$$= \log \int p(u^{(i)}_{t:H}, m, m_t|s_t)\frac{q(m, m_t|\tau_{:t})}{q(m, m_t|\tau_{:t})}\ dm\ dm_t \tag{C.2}$$

$$= \log \mathbb{E}_{q(m,m_t|\tau_{:t})}\left[\frac{p(u^{(i)}_{t:H}, m, m_t|s_t)}{q(m, m_t|\tau_{:t})}\right] \tag{C.3}$$

$$\geq \mathbb{E}_{q(m,m_t|\tau_{:t})}\left[\log \frac{p(u^{(i)}_{:H}, m, m_t|s_t)}{q(m, m_t|\tau_{:t})}\right] \tag{C.4}$$

$$= \mathbb{E}_{q(m,m_t|\tau_{:t})}\Big[\log p(u^{(i)}_{t:H}, m, m_t|s_t) - \log q(m, m_t|\tau_{:t})\Big] \tag{C.5}$$

$$= \mathbb{E}_{q(m,m_t|\tau_{:t})}\Big[\log p(u^{(i)}_{t:H}|s_t, m, m_t) + \log p(m, m_t|s_t) - \log q(m, m_t|\tau_{:t})\Big] \tag{C.6}$$

$$= \mathbb{E}_{q(m,m_t|\tau_{:t})}\Big[\log p(u^{(i)}_{t:H}|s_t, m, m_t)\Big] - KL(q(m, m_t|\tau_{:t})||p(m, m_t|s_t)). \tag{C.7}$$

We can then expand the left-hand as follows.

$$\mathbb{E}_{q(m,m_t|\tau_{:t})}\Bigg[\log p(u_{t:H}^{(i)}|s_t,m,m_t)\Bigg]$$

$$= \mathbb{E}_{q(m,m_t|\tau_{:t})}\Bigg[\log \iint p(u_t^{(i)},u_t^{(-i)},s_{t+1},u_{t+1:H}|s_t,m,m_t) \; du_t^{(-i)}ds_{t+1}\Bigg] \tag{C.8}$$

$$= \mathbb{E}_{q(m,m_t|\tau_{:t})}\Bigg[\log \iint p(u_{t+1:H}^{(i)}|u_t^{(i)},u_t^{(-i)},s_{t+1},s_t,m,m_t). \; p(u_t^{(i)},u_t^{(-i)},s_{t+1}|s_t,m,m_t) \; du_t^{(-i)}ds_{t+1}\Bigg] \tag{C.9}$$

$$= \mathbb{E}_{q(m,m_t|\tau_{:t})}\Bigg[\log \iint p(u_{t+1:H}^{(-i)}|u_t^{(i)},u_t^{(-i)},s_{t+1},s_t,m,m_t) \; p(u_t^{(-i)},s_{t+1}|u_t^{(i)},s_t,m,m_t) \; p(u_t^{(i)}|s_t,m,m_t) \; du_t^{(-i)}ds_{t+1}\Bigg] \tag{C.10}$$

$$= \mathbb{E}_{q(m,m_t|\tau_{:t})}\Bigg[\log \iint p(u_{t+1:H}^{(-i)}|s_{t+1},m,m_{t+1}) \; p(u_t^{(-i)},s_{t+1}|u_t^{(i)},s_t,m,m_t) \; p(u_t^{(i)}|s_t,m,m_t) \; du_t^{(-i)}ds_{t+1}\Bigg] \tag{C.11}$$

$$= \mathbb{E}_{q(m,m_t|\tau_{:t})}\Bigg[\log \mathbb{E}_{p(u_t^{(-i)},s_{t+1}|u_t^{(i)},s_t,m,m_t)}\Bigg[p(u_{t+1:H}^{(-i)}|s_{t+1},m,m_{t+1})p(u_t^{(i)}|s_t,m,m_t)\Bigg]\Bigg] \tag{C.12}$$

$$\vdots$$

$$= \mathbb{E}_{q(m,m_t|\tau_{:t})}\Bigg[\log \mathbb{E}_{p(u_{t:H}^{(-i)},s_{t+1:H+1}|u_t^{(i)},s_t,m,m_t)}\Bigg[\prod_{i=t+1}^{H} p(u_i^{(i)}|s_i,m,m_i)\Bigg]\Bigg] \tag{C.13}$$

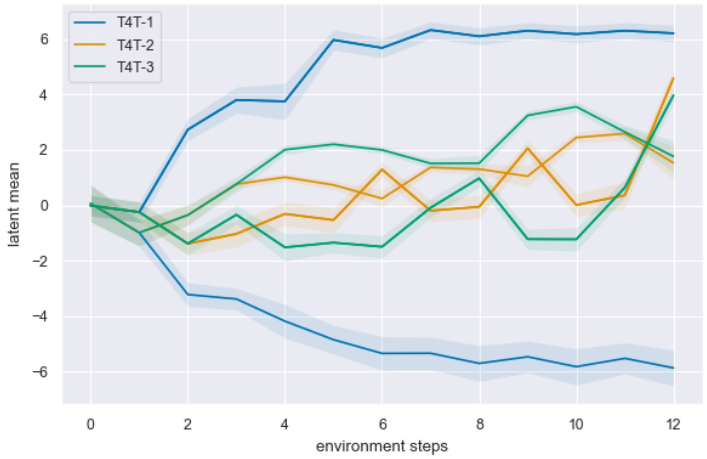
Going from (C.12) to (C.13), we iterate over timesteps, repeating steps (C.8)-(C.12) for $t+1,\ldots,H$. For the right-hand side, we use the previous posterior as the new prior, $q(m,m_t|\tau_{:t-1})$, with univariate Gaussian priors $q(m,m_t|s_0) = \mathcal{N}(0,1)$ at the first timestep.

C.2 Additional Results

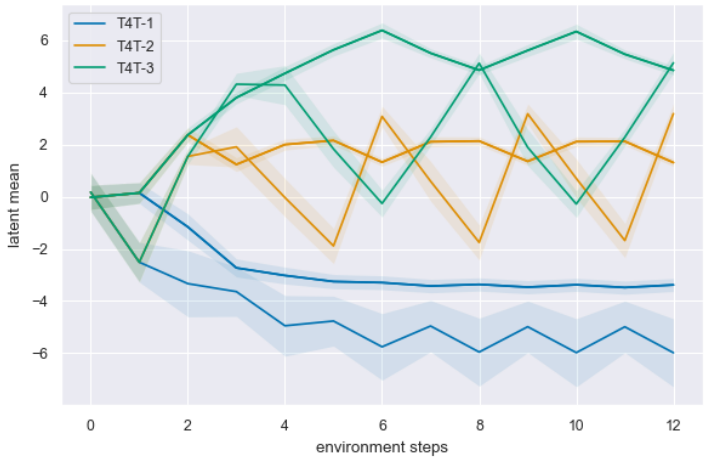
Game of Chicken Figure C.1 shows the latent variables for different ablations of the MeLIBA architecture. Figures C.1a and C.1b show the latent variable per agent type, when using either only the permanent latent variable m (together with a feed-forward decoder) or only the temporal latent variable m_t (together with a recurrent decoder). We first observe that both seem to "count" the number of interactions to some extend, although the permanent latent m does so less pronounced and consistent. Figure C.1c shows the latent variables at test time, when using a non-hierarchical version of MeLIBA. We can see that some of the temporal structure is also captured in m here. This is undesirable, because we want the model to clearly separate temporal and permanent structure (like MeLIBA does, see Figure 7.2b).

Figure C.2 shows additional learning curves for variations of LIOM and ablations of MeLIBA. We see that all MeLIBA architectures learn to solve the task, except when we use only a permanent latent variable m. The reason is a mismatch in model type: the other agent conditions its actions on the interaction history, however, the action decoder is not able to model this since it is a feed-forward network conditioned only on the last state and actions. We also see that LIOM's (Papoudakis et al., 2020) performance increases when either backpropagating the RL loss through the encoder (LIOM (rlloss)), or when using a larger latent dimension (LIOM (10D)). Still, the performance is inferior in both these cases. Since the only difference between MeLIBA and LIOM is that LIOM passes a sample to the policy instead of the mean and variance, we believe that the noisy inputs this generates for the policy lead to an instability in training.

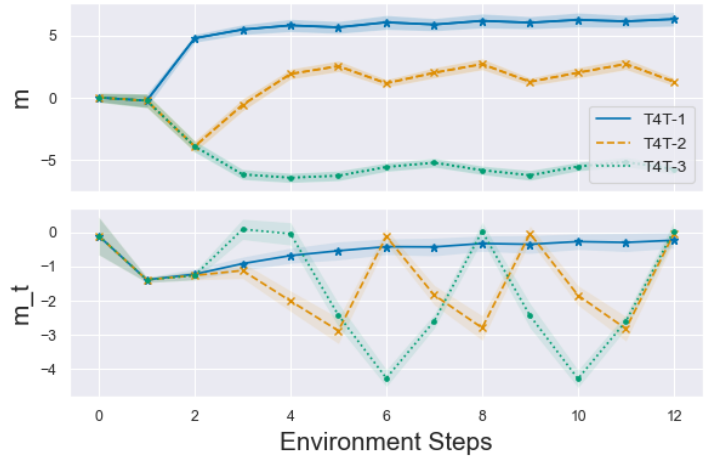
Treasure Hunt We use a 10×10 grid and the agents can choose from five discrete actions *no-op, left, right, up, down.* The rewards are -0.1 if the agents collide, $+0.1$ for picking up a coin, $+1$ for dropping off a coin at the correct bank, and -1 for dropping off a coin at the wrong bank. If the agents attempt to walk into a wall or another agent, their action does not get executed and they stay on the current grid. State transitions are deterministic.



(a) MeLIBA-m



(b) MeLIBA-m_t



(c) MeLIBA-flat

Figure C.1: Latent variables during test rollout, for different MeLIBA architectures.

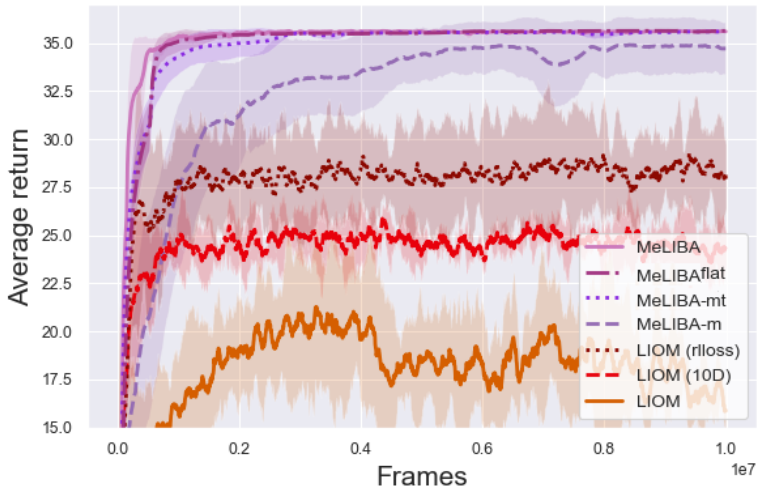


Figure C.2: Additional Learning Curves for the Chicken Game.

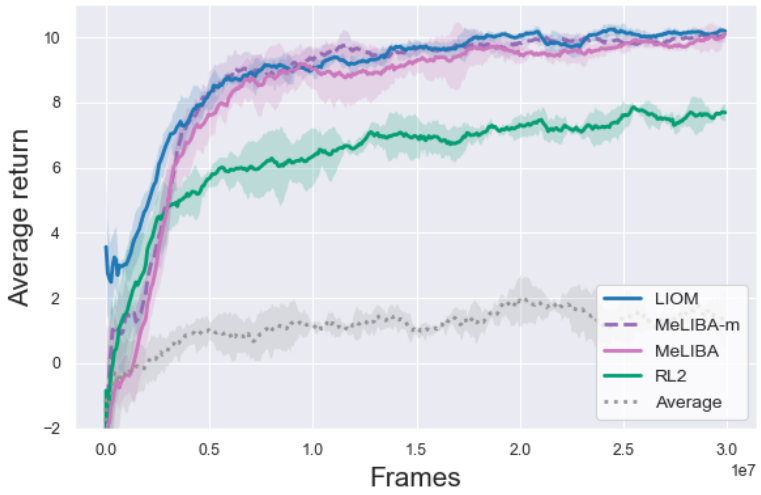


Figure C.3: Treasure Hunt Learning Curves.

The hard-coded agents go directly to the coin colour they prefer, without avoiding other-coloured coins or the other agent. If they pick up a coin of the wrong colour, they drop it off at the correct bank.

Figure C.3 shows the learning curves for the Treasure Hunt game, for 3 seeds per method. All learning curves show a 95% confidence intervals in the shaded areas, across the different seeds.

To classify agent type from latent variables for Figure 7.3c, we fit a Logistic Regression Classifier using the scikit-learn Python package. Per trained model, we use 80 test rollouts for fitting the classifier, and show the test accuracy on

Method	Chicken Game (10M Frames)	Treasure Hunt (25M Frames)
MeLIBA	2-3	5
MeLIBA (m)	1-2	5
RL^2	4	12
Feedforward	1	3

Table C.1: Runtimes in Hours

20 rollouts in Figure 7.3c. The shaded areas are one standard deviation across 3 trained models (different seeds).

Runtimes We trained our models using Nvidia Tesla K80 GPUs. Table C.1 shows the runtimes for the different methods. We typically ran between 1 and 4 experiments per GPU in parallel, leading to some variation in the runtime. These numbers should be seen as a rough guide and are useful to compare relative times between methods.

C.3 Implementation Details

Hyperparameters Hyperparameters for MeLIBA are shown in Table C.2. Additional notes on the hyperparameters:

- *: In the first layer of the encoder, we separately encode the state/action/rew with a FC layer with output 32/16/16 for the Chicken Game, and 64/16/16 for the Treasure Hunt.
- ^{R}: Denotes the (hidden size of the) recurrent layer.
- To Train RL^2, we used a recurrent network with hidden size 128 and three layers.
- We use GRUs as our aggregator in the recurrent parts of the networks.
- We selected the hyperparameters doing a simple linesearch (see Table for the values we considered).

Parameter		Values Chosen		Values Tested
		Chicken Game	Treasure Hunt	
Num Training Frames		1e7	2.5e7	
Policy	Algorithm	PPO	PPO	
	Layers	[128, 128]	[128, 128]	
	Non-Linearity	tanh	tanh	
	Optimiser	RMSProp	RMSProp	Adam
	Learning Rate	7e-4	7e-4	1e-4, 1e-3
	Num Epochs	2	2	1, 2, 4
	Num Mini-Batches	8	4	2, 4, 8
	Value Clip Param	0.1	0.1	
	Batchsize	2,080	1,600	
	Value Loss Coefficient	0.5	0.5	
	Entropy Coefficient	0.2	0.2	0.1, 0.5
	Discount Factor	1	1	
	Tau (for GAE)	0.9	0.9	0.95
	Max Grad Norm	0.5	0.5	
VAE	Learning Rate	0.001	0.001	
	Optimiser	Adam	Adam	
	Data Buffer	2500 (unique traj)	2500	10000
	Batchsize (Num Traj)	50	15	
	KL weight	0.05	0.05	0, 0.01, 0.1
	Pretrain: Num Frames	208,000	32,000	
	Pretrain: Num Updates	5000	1000	
	Latent dim, m	2	5	2, 5, 10
	Latent dim, m_t	2	5	2, 5, 10
	Encoder Layers	$[64^*, 64^R,$ $64, 64^{m_t}]$	$[96^*, 128^R,$ $64, 64^{m_t}]$	
	Decoder Layers	$[32^m, 64^{m_t},$ $64, 64^R, 32]$	$[32^m, 64^{m_t},$ $64, 64^R, 32]$	

Table C.2: MeLIBA Hyperparameters

D

Appendix for Chapter 8, HyperX

D.1 Additional Background

Randomised Prior Functions In Reinforcement Learning, we can use the fact that unseen states can be seen as out-of-distribution data of a model that is trained on all data the agent has seen so far. Getting uncertainty estimates on states can thus quantify our uncertainty about the value of a state and in turn whether we have explored these states sufficiently. We can think about why exploration purely in the state space $\mathcal{S}$ (which is shared across tasks) is not enough: if the agent has explored a state many times in one task and is certain of its value, it should not necessarily exploit this knowledge in a different task, because this same state could have a completely different value. We cannot view these as separate exploration problems however, since we also have to try out different deployed exploration strategies and combine the information to meta-learn Bayes-optimal behaviour.

Therefore, we want to incentivise the agent to explore in the hyper-state space $\mathcal{S}^+ = \mathcal{S} \times \mathcal{B}$. Only if an environment state together with a specific belief has been observed sufficiently often to determine its value should the agent trust its value estimate of that belief-state. This therefore amounts to exploration in a BAMDP state space, which essentially means trying out different exploration strategies in the environments of the training distribution. We use Random Network

Distillation (RND) (Osband et al., 2018; Burda et al., 2019b; Ciosek et al., 2020) to obtain such uncertainty estimates and review them using the formulation of Ciosek et al. (2020) in the following.

Assume we have set of training data $\mathcal{D} = \{s_i\}_{i=1}^N$ of all states the agent has observed. To get uncertainty estimates, we first fit B predictor networks $g_j(s)$ $(j = 1, \ldots, B)$ to a random prior process $f_j(s)$ each (a network with randomly initialised weights, which is fixed and never updated). We then estimate the uncertainty for a state s_* as

$$\sigma^2(s_*) = \max(0,\ \sigma_\mu^2(s_*) + \beta v_\sigma(s_*) - \sigma_A^2), \tag{D.1}$$

where $\sigma_\mu^2(s_*)$ is the sample mean of the squared errors between the B predictor networks and the prior processes; $v_\sigma(s_*)$ is the sample variance of the squared error. The first quantifies our uncertainty, whereas the second quantifies our uncertainty over what our uncertainty is. In practice, $B = 1$ is typically sufficient and the second term disappears (Ciosek et al., 2020). The term σ_A^2 is the aleatoric noise inherent in the data which is an irreducible constant. In theory, this can be learned as well and depends on how much information can be extracted about the value of states and actions from the data. In practice, we set this term to 0.

Given a hyper-state $s_t^+ = (s_t, b_t)$, an ensemble of B prior networks $\{f^i(s^+)\}_{i=1}^B$ and corresponding predictor networks $\{h^i(s^+)\}_{i=1}^B$, the reward bonus is defined as

$$r_c(s_t^+) = \max(0,\ \sigma_m u^2(s_t^+) + \beta v_\sigma(s_t^+) - \sigma_A^2) \tag{D.2}$$

where $\sigma_m u^2(s_t^+)$ is the sample mean of the squared error between prior and predictor networks and $v_\sigma(s_t^+)$ is the sample variance of that error.

D.2 Additional Results

In this section we provide additional experimental results. This includes results on sparse environments previously used in the literature (sparse Meta-World and 2D navigation), where our baselines already performed well. In addition, we provide more details and results for the experiments in the main paper. The source code is available at `https://github.com/lmzintgraf/hyperx`.

		Dense Rewards			Sparse Rewards		
Method	Episode	Reach	Push	Pick-Place	Reach	Push	Pick-Place
MAML*	10	48	74	12	-	-	-
PEARL*	10	38	71	28	-	-	-
RL^{2*}	10	45	87	24	-	-	-
E-RL^{2+}	10	-	-	-	28	7	-
MetaCURE+	10	-	-	-	46	25	-
VariBAD	**1**	**100**	**100**	29 (6/20)	**100**	**100**	2 (1/20)
VariBAD	2	100	100	29	100	100	2
HyperX	**1**	**100**	**100**	**43** (9/20)	100	100	2 (1/20)
HyperX	2	100	100	43	100	100	2

Table D.1: Meta-test success rates on the ML1 Meta-World benchmark, for the dense and the sparse reward version. *Results taken from Yu et al. (2019). +Results taken from Zhang et al. (2021). We ran VariBAD and HyperX for 5 random seeds for dense reach/push, and 20 seeds for dense pick-place. VariBAD and HyperX were trained to maximise expected online return within 2 episodes. The first (few) episodes often *includes exploratory actions*, yet have higher success rate than existing methods that maximise final episodic return. For the sparse Pick-Place environment, in brackets we report the number of seeds that learned something.

Meta-World To test how our method scales up to more challenging problem settings, we evaluate it on the Meta-World benchmark (version 1; Yu et al., 2019), where a simulated robot arm has to perform tasks. We evaluate our method on the ML1 benchmark, of which three different versions exist: reach/push/pick-and-place (in increasing order of complexity). In each of these, task distributions are generated by varying the starting position of the agent and the goal/object positions.

Each environment has a dense reward function that was designed such that an agent trained on a single task (i.e., fixed starting/object/goal position) can learn to solve it. Evaluation is done in terms of success rate (rather than return), which is a task-specific binary signal indicating whether the task was accomplished (at any moment during the rollout). Yu et al. (2019) proposed a sparse version of this benchmark that uses this binary success indicator, rather than the dense reward, for training. This sparse version is used in Zhang et al. (2021), on ML1-reach and ML1-push.

The agent is trained on a set of 50 environments and evaluated on unseen

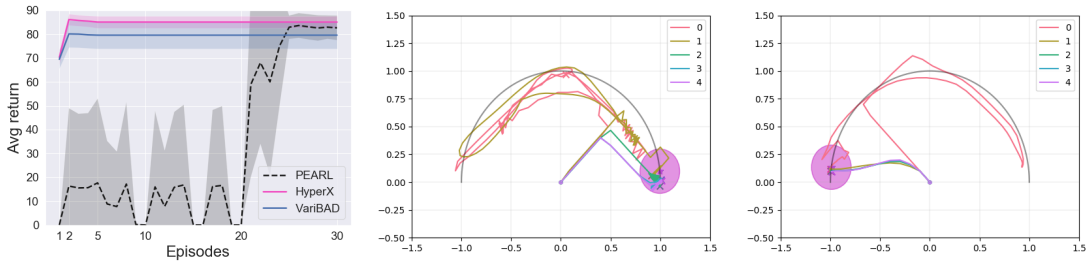
environments from the same task distribution. In all baselines, the agent has 10 episodes to adapt, and performance is measured in the last rollout. Since we consider the *online adaptation* setting where we want the agent has to perform well from the start, we trained VariBAD and HyperX to maximise online return during the first two episodes. This is more challenging since it includes exploratory actions.

Table D.1 shows the results for both the dense and sparse versions of ML1.

ML1-reach / ML1-push VariBAD achieves 100% success rate on both the dense *and the sparse* version of ML1-reach and ML1-push *in the first rollout*. Compared to other existing methods – even MetaCURE (Zhang et al., 2018) which explicitly tries to deal with sparsity – this is a significant improvement. We confirm in our experiments that HyperX does not decrease performance and also reaches 100% success rate on these environments.

ML1-pick-place The environment ML1-pick-place is more challenging, because the task consists of two steps: picking up an object and placing it somewhere (where both the object and goal location differ across tasks). Even on the dense version, existing methods struggle. HyperX achieves state of the art on this task with 44.5% success rate, suggesting HyperX can help meta-learning even when rewards are dense. For VariBAD and HyperX we found that our agents either learn the task near perfectly (and have close to 100% success rate in the first rollout), or not at all. VariBAD learned something for 6 out of 20 seeds, and HyperX learned something for 9 out of 20 seeds. For the sparse version of this environment, we only saw some success for 1/20 seeds for both VariBAD or HyperX.

We suspect that the main challenge in ML1-Pick-Place is the short horizon (150), which does not give the agent enough time to explore during meta-training. This is why HyperX can give some improvement even in the dense version. In an upcoming version of Meta-World (Yu et al., 2019), the horizon will be increased to 200, opening up interesting opportunities for future research on sparse Pick-Place.



(a) Meta-Test Performance **(b)** VariBad example rollout **(c)** HyperX example rollout

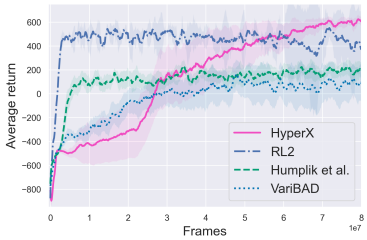
Figure D.1: Meta-test performance on Sparse 2D Navigation. (a) Performance averaged over the task distribution at the end of training. Because PEARL is not optimising for optimal exploration, it requires more episodes to find the goal. Both VariBad and HyperX optimise for optimal exploration and are able to quickly find the goal (b and c). However, VariBad's exploration is suboptimal, not covering all possible goal locations equally well (b), explaining the lower performance compared to HyperX.

Sparse 2D Navigation We evaluate on a Point Robot 2D navigation task used by Gupta et al. (2018), Rakelly et al. (2019), and Humplik et al. (2019). The agent must navigate to an unknown goal sampled along the border of a semicircle of radius 1.0, and receives a reward relative to its proximity to the goal when it is within a goal radius of 0.2. Thus far, only Humplik et al. (2019) successfully meta-learn to solve this task by meta-training with sparse rewards, though they rely on privileged information during meta-training (the goal position). The other methods meta-train with dense rewards and evaluate using sparse rewards. We use a horizon of 100 here (instead of 20 as in the papers above) to give VariBAD and HyperX enough time to demonstrate interesting exploratory behaviour.

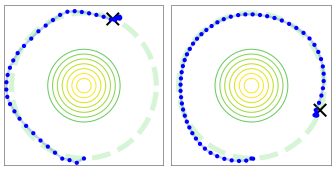
Figure D.1a shows the performance of PEARL, VariBAD, and HyperX at test time, when rolling out for 30 episodes. Both VariBAD and HyperX adapt to the task quickly compared to PEARL, but HyperX reaches slightly lower final performance.

To shed light on these performance differences, Figures D.1b and D.1c visualise representative example rollouts for the meta-trained VariBAD and HyperX agents. We picked examples where the target goals are at the end of the semi-circle, which we found are most difficult for the agents. VariBAD (D.1b) struggles to find the goal, taking several attempts to reach it. Once the goal is found, it does return to it but on a sub-optimal trajectory. By contrast, HyperX searches the space of goals more strategically, and returns to the goal faster in subsequent episodes.

(a) Learning curves for ablations.



(b) Learning curves for baselines.



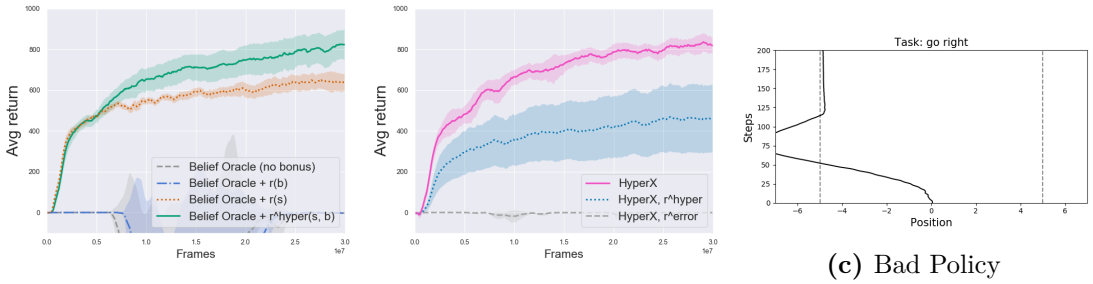
(c) VariBAD **(d)** RL^2

Figure D.2: Treasure Mountain - Additional Rollouts. Shown are example rollouts for the final agents of VariBAD and RL^2 (Duan et al., 2016; Wang et al., 2016). They follow the inferior exploration strategy of walking around the circle until the treasure is found, instead of climbing the mountain to directly observe the treasure and get there faster.

Treasure Mountain Figure D.2a shows the learning curves for the HyperX, in comparison to ablating different exploration bonuses. When using only the hyper-state novelty bonus r^{hyper}, HyperX learns the inferior strategy of walking in a circle: it has no incentive to go up the mountain early in training (because beliefs there are meaningless because the VAE has not learned yet to interpret the hint) and stars avoiding the mountain. When using only the VAE reconstruction error bonus r^{error}, the agent learns the superior strategy of walking up the mountain to see the goal location 70% of the time (7/10 seeds). In contrast, HyperX, which uses both exploration bonuses, learns the superior strategy for all 10 seeds. Lastly, we tested VariBAD with a simple state novelty exploration bonus: this again learns the inferior circle-walking strategy only, because it quickly learns to avoid the mountain top.

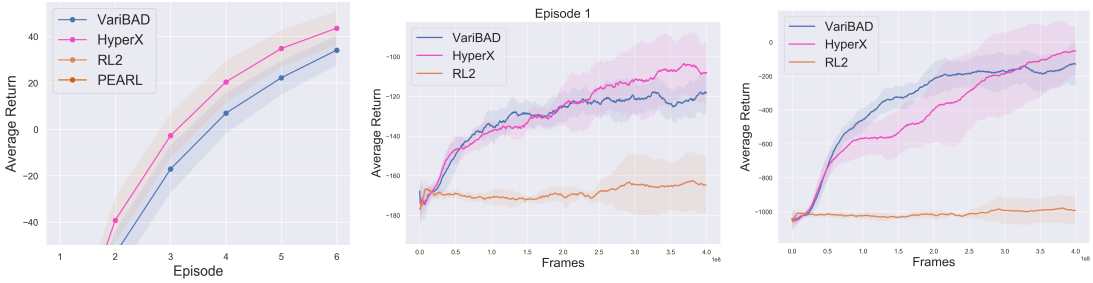
Figure D.2b shows the learning curves for HyperX, VariBAD, as well as additional baselines RL^2 (Duan et al., 2016; Wang et al., 2016) and the Belief Learning method of Humplik et al. (2019). Both these baselines also only learn the inferior circle-walking strategy, because the correct incentives for meta-exploration are missing.

Figures D.2c and D.2d show meta-test time behaviour of VariBAD and RL^2: both methods learn to walk in a circle until the goal is found. This was consistent across all (10) seeds.



(a) Belief Oracle. **(b)** HyperX **(c)** Bad Policy

Figure D.3: HalfCheetahDir: additional results. Learning curves for the Belief Oracle (a) and HyperX (b), with and without reward bonus, averaged over 20 seeds. Figure (c) shows the behaviour of a policy which failed to learn Bayes-optimal behaviour. We observe such behaviour often when training HyperX with the reward bonus on the hyper-states only, $r^{hyper}(b, s)$.



(a) Online Return at Test Time **(b)** Learning curves (Ep 1) **(c)** Learning Curves (Ep 1-6)

Figure D.4: Sparse AntGoal: additional plots. (a) Return per episode at meta-test time (standard error shaded). RL^2 and PEARL do not learn to solve the task and achieve a reward of around -150 per episode. (b) Learning curve over meta-training for the online returns in episode 1. (c) Learning curve over meta-training for the online return accumulated over 6 episodes. Results are averaged over 10 seeds.

Sparse CheetahDir Figure D.3 shows the learning curves for the Sparse CheetahDir experiments, with 95% confidence intervals (over 20 seeds). Fig D.3a shows this for the Belief Oracle, with different exploration bonuses. Fig D.3b shows this for HyperX, with different exploration bonuses.

Figure D.3c shows example behaviour of a suboptimal policy at test time. The agent returns back into the zero-reward zone after realising that the task was not "go left", but stays in there instead of behaving optimally, which is going further to the right and into the dense reward area beyond the sparse interval border.

Sparse MuJoCo AntGoal Figure D.4a shows the returns achieved by the agents across different episodes. Figures D.4b show the learning curves for the returns during the *first* episode, with 95% confidence intervals (shaded areas, 10 seeds). Figure D.4c shows the combined learning curves, comprising of all 6 episodes, with 95% confidence intervals (shaded areas, 10 seeds). Figures D.6 and D.5 show example rollouts for VariBAD and HyperX.

Dense AntGoal We also evaluated HyperX on the dense AntGoal environment. VariBAD and HyperX were trained to maximise performance within a single episode. PEARL was trained with the default hyperparameters provided by the open-sourced code of the authors. The results are:: VariBAD: -123 (Episode 1), HyperX: -127 (Episode 1), PEARL: -200 (Episode 6). This confirms that HyperX does not impact performance, but that there is also not much room for improvement.

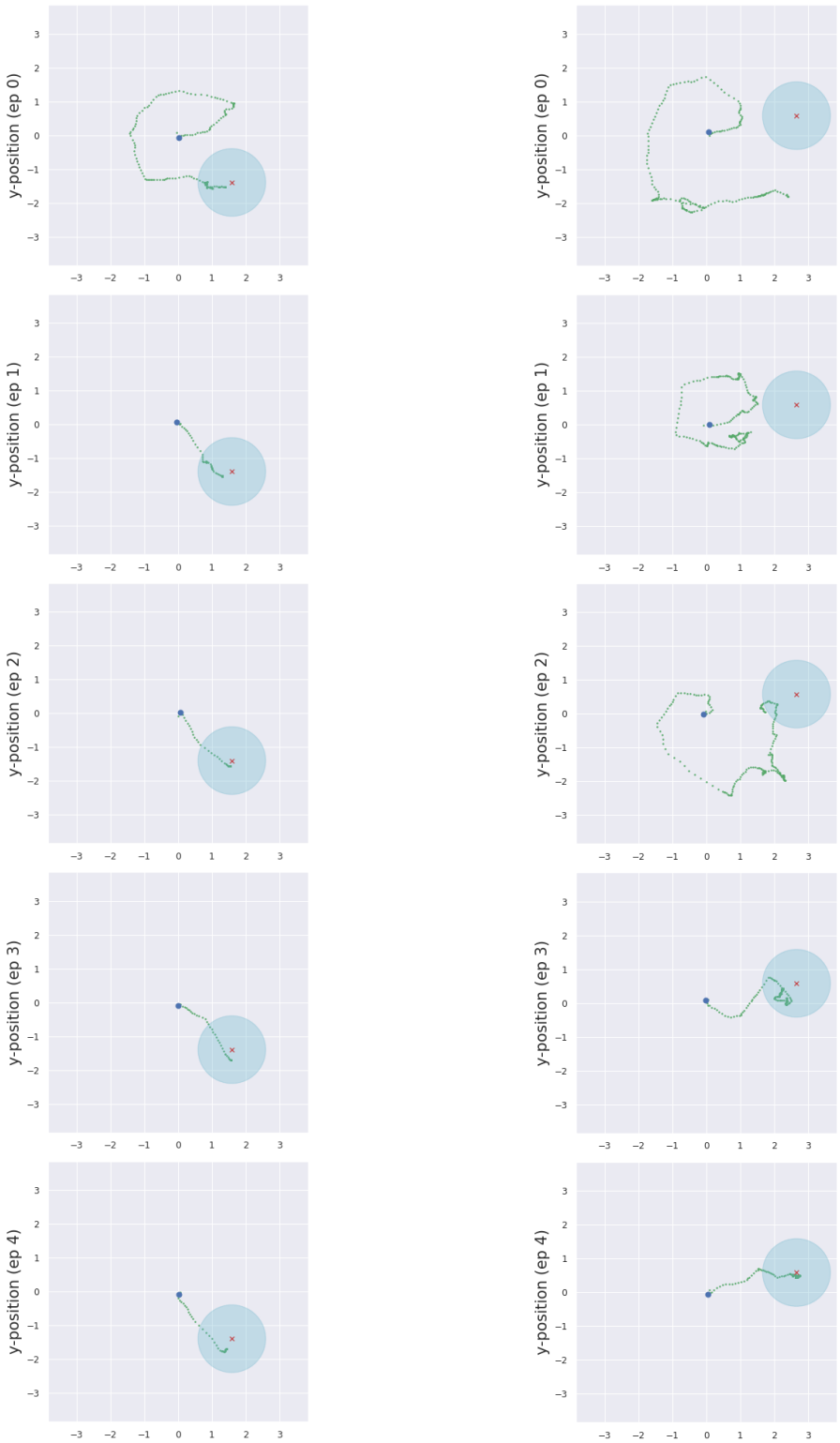


Figure D.5: HyperX Example Rollouts

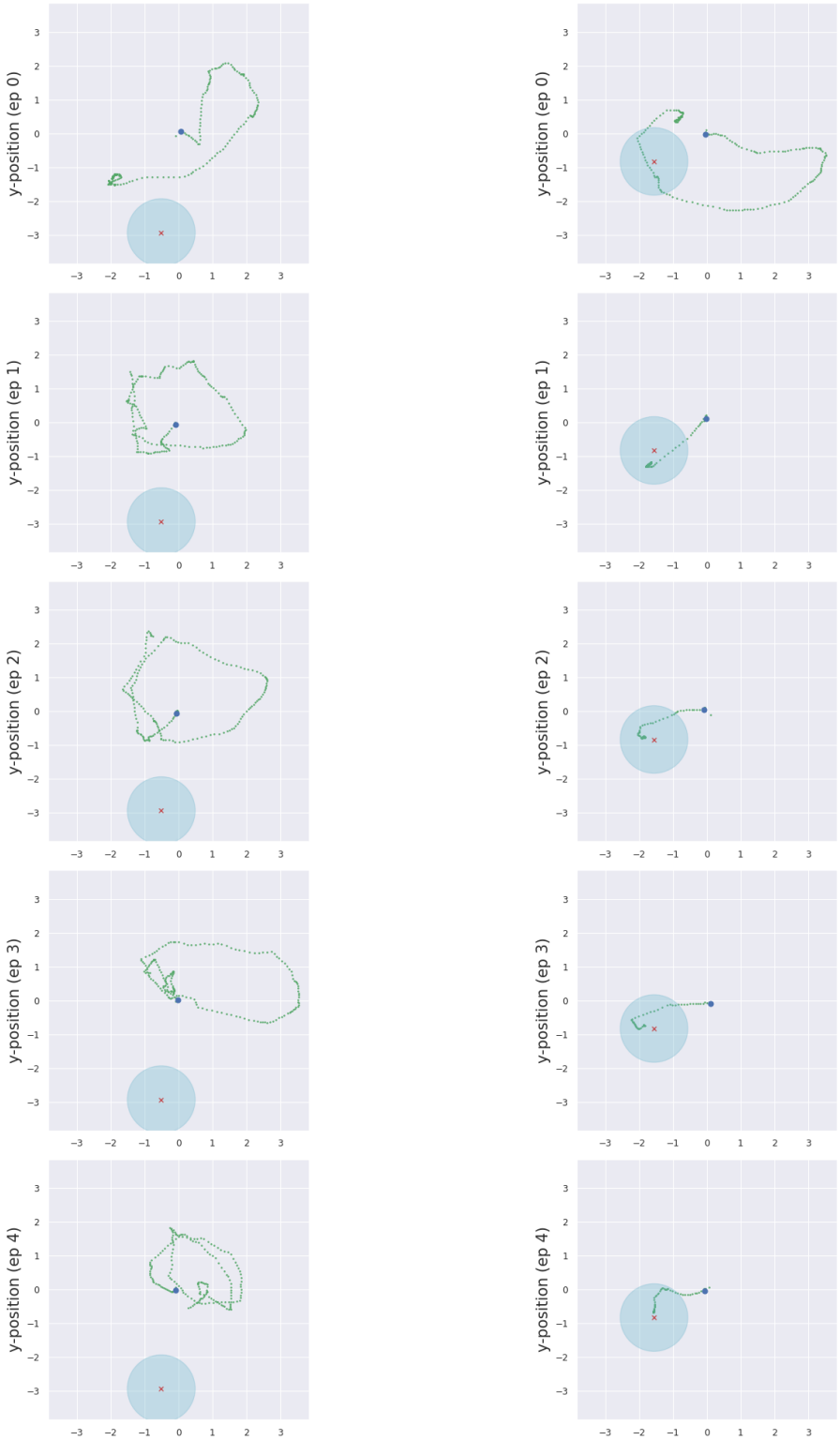


Figure D.6: VariBAD Example Rollouts

D.3 Implementation Details

Here we provide the environment specifications, runtimes, and hyperparameters.

Treasure Mountain This environment is implemented as follows. The treasure can be anywhere along a circle of radius 1. Within that circle is a mountain – implemented as another circle with radius 0.5. The horizon is 100 and there are no resets. The agent always starts at he bottom of the circle. It receives a reward of 10 when it is within a Euclidean distance of 0.1 within the treasure (the treasure does not disappear, so it keeps receiving this reward if it stays there). It receives a penalty for climbing the mountain, given my $-5.5 + ||(x, y)||_2$ where (x, y) is the agent's position (the mountain center is $0, 0$, and the mountain radius 0.5). If not at the treasure or on the mountain, the agent gets a timestep penalty of at least -5, which increases as the agent walks further outside the outer circle (to discourage it from walking too far). The agent cannot walk outside $[-1.5, 1.5]$ in either direction.

The observations of the agent are 4D and continuous. The first two dimensions are the agent's (x, y)-position. The last two dimensions are zero if the agent is not on the mountain top, and are the (x, y)-coordinates of the treasure when the agent is on the mountain top (within a radius of 0.1). The agent's actions are the (continuous) stepsize it takes in (x, y)-direction, bounded in $[-0.1, 0.1]$.

Multi-Stage Gridworld The layout of this environment is depicted in Fig 8.3. It consist of three rooms which are of size 3×3 grid, and corridors that connect the rooms of length 3. The environment state is the (x, y) position of the agent, unnormalised. There are five available actions: *no-op, up, right, down, left.*

Three (initially unknown) goals (G1-G3) are placed in corners of rooms: G1 in the middle room, G2 in the room that is on the side where G1 was placed, and G3 in the middle room (but not where G1 was placed). The agent always starts in the middle of the centre room and has $H = 50$ steps. The goals provide increasing rewards, i.e. $r_1 = 1$, $r_2 = 10$ and $r_3 = 100$, but are only sequentially unlocked; G2 (r_2) is only available after G1 has been reached; G3 (r_3) is only available after

G2 has been reached. The environment is partially observable (Poupart et al., 2008; Cai et al., 2009) as the agent only observes its position in the environment and not which goals are unlocked. If the agent is not on an (available) goal it gets $r = -0.1$. When the agent stands on a goal, it keeps receiving the respective reward while standing there (the goal does *not* disappear). The best strategy is to search the first room for G1, then search the appropriate room for G2, and then return to the middle room to find G3.

Sparse MuJoCo HalfCheetahDir We use the commonly used HalfCheetahDir Meta-Learning benchmark, and sparsify it as follows. If the agent's x-position is within $[-5, 5]$ it only gets the control penalty; otherwise it gets the standard dense reward comprised of the sum of the control penalty and the 1D velocity in the correct direction.

Sparse MuJoCo AntGoal We use the commonly used AntGoal Meta-Learning benchmark (based on code of Rakelly et al. (2019)), and sparsify it as follows. We extend the environment's state space by including the x and y-position of the agent's torso. In the original AntGoal, the goal is sampled from within a circle of radius of 3 with a higher chance of the goal being sampled away from the centre of the circle. Unlike the dense version where the agent receives a dense goal-related reward at all times, our sparse AntGoal only receives goal-related rewards when within a radius of 1 of the goal.

The agent receives at all time a control penalty and contact forces penalty. When outside the goal circle, the agent receives an additional constant negative reward that is equivalent to the negative goal radius, i.e. -1. When within the goal circle, the agent receives a reward of 1 for being within the goal circle and a penalty equivalent to the negative distance to the goal, essentially encouraging the agent to walk towards the centre of the goal circle.

For the MuJoCo environments, we only used the relevant state information for the RND hyper-state bonus (the x-axis for HalfCheetahDir, and the x-y-position for AntGoal).

Meta-World We use the official version of Meta-World as provided by Yu et al. (2019) at `https://github.com/rlworkgroup/metaworld`. As suggested by Yu et al. (2019) and as tested in Zhang et al. (2021), for the sparse version of this environment, we use the *success* criterion which the environment returns, and give the agent a reward of 0 if *success=False* and a reward of 1 if *success=True*. The success criterion depends on the environment; in 'Reach' for example it is *true* if the agent put its gripper close to the (initially unknown) goal position, and *false* otherwise. For evaluation, we report 'Success' if the agent was successful at any moment during an episode, following the evaluation protocol proposed by Yu et al. (2019).

Runtimes Table D.2 shows the runtimes for our experiments. Unless otherwise stated, we used a NVIDIA GeForce GTX 1080 GPU. These runtimes should serve as a rough estimate, and can vary depending on hardware and concurrent processes.

Environment	Frames	Runtime (ca.)
Treasure Mountain	$8e{+}7$	35h
Multi-Stage Gridworld	$1e{+}8$	65h (CPU)
Sparse HalfCheetahDir	$3e{+}7$	20h (CPU)
Sparse AntGoal	$4e{+}8$	65h
Meta-World	$5e{+}7$	45h
Sparse 2D Navigation	$5e{+}7$	12h

Table D.2: Runtimes

RND Hyperparameter Sensitivity To assess how sensitive HyperX to choices of hyperparameters that affect the hyperstate exploration bonus, we evaluated it on a range of different choices, shown in Table D.3. There is little sensitivity to architecture depth and batchsize, as well as to the output dimension of the RND networks. Performance is stable for learning rates $10^{-3}{-}10^{-6}$ (possibly because we use the Adam optimiser), but we found that the best frequency ($freq$) at which the RND network is updated to be environment dependent. Performance is sensitive to the scaling factor (wsi in the table) for the initial prior network weights. We used

a scaling factor of 10 in our experiments, and found that too small or too large scaling factors can hurt performance. An interesting direction for future work is to find more principled ways to guide the choice of the hyperparameters that are particularly sensitive to the exploration and across environments.

RND $dim_{out} = 32$ (default 128)	737
RND $dim_{out} = 256$ (default 128)	812
RND $depth = 1$ (default 2)	794
RND $depth = 3$ (default 2)	814
RND $batchsize = 32$ (default 128)	856
RND $batchsize = 256$ (default 128)	867
RND $lr = 1e-2$ (default $1e-4$)	108
RND $lr = 1e-3$ (default $1e-4$)	883
RND $lr = 1e-5$ (default $1e-4$)	845
RND $lr = 1e-6$ (default $1e-4$)	766
RND $wsi = 1$ (default 10)	597
RND $wsi = 5$ (default 10)	766
RND $wsi = 15$ (default 10)	533

Table D.3: Additional Sparse CheetahDir Results, for different RND hyperparameter settings (averaged over three seeds). *wsi* stands for weight scale initialisation of the fixed random prior network.

Hyperparameters We train the policy using PPO, and we add the intrinsic bonus rewards to the extrinsic environment reward and use the sum when learning with PPO. We normalise the intrinsic and extrinsic rewards separately by dividing by a rolling estimate of the standard deviation. The next two pages show the hyperparameters used for the policy, the VAE, and the exploration bonuses. Hyperparameters were selected using a simple (non-exhaustive) gridsearch.

	Treasure	Gridworld	CheetahDir	AntGoal	PointRobot	ML1-Reach	ML1-Push	Ml1-Pick-Place
max_rollouts_per_task	1	1	1	6	3	2	2	2
policy_state_embedding_dim	None	32	32	64	64	64	64	64
policy_latent_embedding_dim	None	32	32	64	64	64	64	64
norm_state_for_policy	False	True	True	True	True	True	True	True
norm_latent_for_policy	False	True	False	True	True	True	True	True
norm_rew_for_policy	True	True	True	False	True	True	True	True
norm_actions_pre_sampling	False	False	False	True	True	False	False	False
norm_actions_post_sampling	True	False	False	False	False	False	False	True
norm_rew_clip_param	100	NaN	NaN	100000	100000	100000	100000	100000
policy_layers	[128, 128]	[64]	[128, 128]	[128, 128, 128]	[128, 128, 128]	[128, 128]	[128, 128]	[128, 128]
policy_activation_function	tanh	tanh	tanh	tanh	tanh	tanh	tanh	tanh
policy_initialisation	orthogonal	normc	normc	normc	normc	normc	normc	normc
policy_anneal_lr	False	False	False	False	False	False	False	False
policy	ppo	ppo	ppo	ppo	ppo	ppo	ppo	ppo
policy_optimiser	adam	adam	adam	adam	adam	adam	adam	adam
ppo_num_epochs	2	8	2	2	2	2	2	2
ppo_num_minibatch	8	4	4	8	8	8	8	8
ppo_clip_param	0.05	0.05	0.1	0.1	0.1	0.1	0.1	0.1
lr_policy	0.0007	0.0007	0.0007	0.0003	0.0007	0.0007	0.0007	0.0007
num_processes	16	16	16	16	16	8	8	8
policy_num_steps	150	50	200	1200	600	600	600	600
policy_eps	1e-08	1e-05	1e-08	1e-08	1e-08	1e-08	1e-08	1e-08
policy_value_loss_coef	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
policy_entropy_coef	0.001	0.1	0.0001	0.001	0.001	0.001	0.001	0.001
policy_gamma	0.97	0.98	0.97	0.99	0.99	0.97	0.97	0.97
policy_use_gae	True	True	True	True	True	True	True	True
policy_tau	0.9	0.95	0.9	0.9	0.9	0.9	0.9	0.9
use_proper_time_limits	True	False	True	True	True	True	True	True
vae_squash_targets	True	NaN	NaN	True	True	NaN	NaN	NaN
lr_vae	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
size_vae_buffer	10000	100000	10000	10000	10000	10000	10000	10000
precollect_len	100	5000	500	50000	5000	5000	5000	5000
vae_batch_num_trajs	15	25	10	10	10	10	10	10
tbptt_stepsize	None	None	None	50	50	None	None	None
vae_subsample_elbos	None	None	None	50	50	50	50	50
vae_subsample_decodes	None	None	None	None	50	50	50	50
vae_avg_elbo_terms	True	False	True	True	False	False	False	False
vae_avg_reconstruction_terms	False	False	False	False	False	False	False	False

	Treasure	Gridworld	CheetahDir	AntGoal	PointRobot	ML1-Reach	ML1-Push	Ml1-Pick-Place
num_vae_updates	1	1	1	10	3	1	1	3
pretrain_len	0	0	0	0	0	0	0	0
kl_weight	1.0	0.1	1.0	1.0	1.0	1.0	1.0	1.0
action_embedding_size	16	0	16	16	16	16	16	16
state_embedding_size	32	32	32	32	32	32	32	32
reward_embedding_size	16	8	16	16	16	16	16	16
encoder_layers_before_gru	[]	[]	[]	[]	[]	[]	[]	[]
encoder_gru_hidden_size	128	128	128	128	128	128	128	128
encoder_layers_after_gru	[]	[]	[]	[]	[]	[]	[]	[]
latent_dim	25	10	5	5	5	5	5	5
decode_reward	True	True	True	True	True	True	True	True
normalise_rew_targets	True	NaN	NaN	False	False	True	True	True
rew_loss_coeff	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
input_prev_state	True	False	True	True	True	True	True	True
input_action	True	False	True	True	True	True	True	True
reward_decoder_layers	[64, 32]	[64, 64]	[64, 32]	[64, 32]	[64, 32]	[64, 32]	[64, 32]	[128, 64, 32]
decode_state	True	False	False	False	False	False	False	True
state_loss_coeff	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
state_decoder_layers	[64, 32]	[32, 32]	[64, 32]	[64, 32]	[64, 32]	[128, 64, 32]	[64, 32]	[128, 64, 32]
rlloss_through_encoder	False	False	False	False	False	False	False	False
intrinsic_rew_normalise_rewards	True	True	True	True	True	True	True	True
intrinsic_rew_clip_rewards	None	10.0	None	10.0	None	10.0	10.0	10.0
rpf_weight_hyperstate	1.0	10.0	1.0	5.0	0.1	1.0	1.0	1.0
intrinsic_rew_anneal_weight	True	True	True	True	True	True	True	True
intrinsic_rew_for_vae_loss	True	True	True	True	True	True	True	True
intrinsic_weight_vae_loss	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
lr_rpf	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
rpf_batch_size	128	128	128	128	128	128	128	128
rpf_update_frequency	1	1	1	3	1	1	1	50
size_rpf_buffer	10000	10000000	10000	10000000	10000	10000	10000	10000
rpf_output_dim	128	128	128	128	128	128	128	128
layers_rpf_prior	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[256, 256]
layers_rpf_predictor	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[256, 256]
rpf_use_orthogonal_init	False	False	False	False	False	False	False	False
rpf_norm_inputs	False	NaN	False	False	False	False	False	False
rpf_init_weight_scale	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0
state_expl_idx	None	None	[17]	[0, 1]	None	None	None	None

References

Abbeel, Pieter and Andrew Y Ng (2004). "Apprenticeship learning via inverse reinforcement learning". In: *International Conference on Machine Learning*, p. 1.

Achiam, Joshua and Shankar Sastry (2016). "Surprise-based intrinsic motivation for deep reinforcement learning". In: *NeurIPS Deep RL Workshop*.

Afsar, M Mehdi, Trafford Crump, and Behrouz Far (2021). "Reinforcement learning based recommender systems: A survey". In: *arXiv preprint arXiv:2101.06286*.

Al-Shedivat, Maruan, Trapit Bansal, Yura Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel (2018). "Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments". In: *International Conference on Learning Representations*.

Albrecht, Stefano V, Jacob W Crandall, and Subramanian Ramamoorthy (2016). "Belief and truth in hypothesised behaviours". In: *Artificial Intelligence* 235, pp. 63–94.

Albrecht, Stefano V and Peter Stone (2017). "Reasoning about hypothetical agent behaviours and their parameters". In: *International Conference on Autonomous Agents and Multiagent Systems*.

Albrecht, Stefano V and Peter Stone (2018). "Autonomous agents modelling other agents: A comprehensive survey and open problems". In: *Artificial Intelligence* 258, pp. 66–95.

Alet, Ferran, Martin F Schneider, Tomas Lozano-Perez, and Leslie Pack Kaelbling (2020). "Meta-learning curiosity algorithms". In: *International Conference on Learning Representations*.

Ambrogioni, Luca (2021). "Knowledge is reward: Learning optimal exploration by predictive reward cashing". In: *arXiv preprint arXiv:2109.08518*.

Amersfoort, Joost van, Lewis Smith, Andrew Jesson, Oscar Key, and Yarin Gal (2021). "On Feature Collapse and Deep Kernel Learning for Single Forward Pass Uncertainty". In: *arXiv preprint arXiv:2102.11409*.

Amin, Susan, Maziar Gomrokchi, Harsh Satija, Herke van Hoof, and Doina Precup (2021). "A Survey of Exploration Methods in Reinforcement Learning". In: *arXiv preprint arXiv:2109.00157*.

Amodei, Dario, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané (2016). "Concrete problems in AI safety". In: *arXiv preprint arXiv:1606.06565*.

Andrychowicz, Marcin, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas (2016). "Learning to learn by gradient descent by gradient descent". In: *Advances in Neural Information Processing Systems*, pp. 3981–3989.

Andrychowicz, OpenAI: Marcin, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. (2020). "Learning dexterous in-hand manipulation". In: *The International Journal of Robotics Research* 39.1, pp. 3–20.

Antonova, Rika, Maksim Maydanskiy, Danica Kragic, Sam Devlin, and Katja Hofmann (2020). "Modular latent space transfer with analytic manifold learning". In: *2nd Workshop on Closing the Reality Gap in Sim2Real Transfer for Robotics*.

Argall, Brenna D, Sonia Chernova, Manuela Veloso, and Brett Browning (2009). "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5, pp. 469–483.

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

Arulkumaran, Kai, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath (2017). "Deep reinforcement learning: A brief survey". In: *IEEE Signal Processing Magazine* 34.6, pp. 26–38.

Asmuth, John and Michael L Littman (2011). "Learning is planning: near Bayes-optimal reinforcement learning via Monte-Carlo tree search". In: *Conference on Uncertainty in Artificial Intelligence*.

Astrom, Karl J (1965). "Optimal control of Markov decision processes with incomplete state estimation". In: *J. Math. Anal. Applic.* 10, pp. 174–205.

Bakker, Tijmen, Kees van Asselt, Jan Bontsema, Joachim Müller, and Gerrit van Straten (2006). "An autonomous weeding robot for organic farming". In: *Field and Service Robotics*. Springer, pp. 579–590.

Bard, Nolan and Michael Bowling (2007). "Particle filtering for dynamic agent modelling in simplified poker". In: *the National Conference on Artificial Intelligence*. Vol. 22. 1, p. 515.

Barrett, Samuel, Peter Stone, Sarit Kraus, and Avi Rosenfeld (2013). "Teamwork with limited knowledge of teammates". In: *AAAI Conference on Artificial Intelligence*.

Bechtle, Sarah, Artem Molchanov, Yevgen Chebotar, Edward Grefenstette, Ludovic Righetti, Gaurav Sukhatme, and Franziska Meier (2020). "Meta-Learning via learned loss". In: *International Conference on Pattern Recognition*.

Bellemare, Marc, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos (2016). "Unifying count-based exploration and intrinsic motivation". In: *Advances in Neural Information Processing Systems*.

Bellemare, Marc G, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, Sameera S Ponda, and Ziyu Wang (2020). "Autonomous navigation of stratospheric balloons using reinforcement learning". In: *Nature* 588.7836, pp. 77–82.

Bellman, Richard (1956). "A problem in the sequential design of experiments". In: *Sankhyā: The Indian Journal of Statistics (1933-1960)* 16.3/4, pp. 221–229.

Bellman, Richard (1957). "Dynamic Programming". In: *Princeton University Press*.

Bellman, Richard (1966). "Dynamic programming". In: *Science* 153.3731, pp. 34–37.

Bengio, Samy, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei (1992). "On the optimization of a synaptic learning rule". In: *Preprints Conf. Optimality in Artificial and Biological Neural Networks*. Univ. of Texas, pp. 6–8.

Bergstrom, Carl T and Peter Godfrey-Smith (1998). "On the evolution of behavioral heterogeneity in individuals and populations". In: *Biology and Philosophy* 13.2, pp. 205–231.

Bishop, Christopher M and Nasser M Nasrabadi (2006). *Pattern recognition and machine learning*. Vol. 4. 4. Springer.

Botvinick, Matthew, Sam Ritter, Jane X Wang, Zeb Kurth-Nelson, Charles Blundell, and Demis Hassabis (2019). "Reinforcement learning, fast and slow". In: *Trends in cognitive sciences* 23.5, pp. 408–422.

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

Brunskill, Emma (2012). "Bayes-optimal reinforcement learning for discrete uncertainty domains". In: *International Conference on Autonomous Agents and Multiagent Systems*.

Burda, Yuri, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros (2019a). "Large-Scale Study of Curiosity-Driven Learning". In: *International Conference on Learning Representations*.

Burda, Yuri, Harrison Edwards, Amos Storkey, and Oleg Klimov (2019b). "Exploration by random network distillation". In: *International Conference on Learning Representation*.

Cai, Chenghui, Xuejun Liao, and Lawrence Carin (2009). "Learning to explore and exploit in POMDPs". In: *Advances in Neural Information Processing Systems*, pp. 198–206.

Canaan, Rodrigo, Xianbo Gao, Youjin Chung, Julian Togelius, Andy Nealen, and Stefan Menzel (2020a). "Evaluating the Rainbow DQN Agent in Hanabi with Unseen Partners". In: *arXiv preprint arXiv:2004.13291*.

Canaan, Rodrigo, Xianbo Gao, Julian Togelius, Andy Nealen, and Stefan Menzel (2020b). "Generating and Adapting to Diverse Ad-Hoc Cooperation Agents in Hanabi". In: *arXiv preprint arXiv:2004.13710*.

Carmel, David and Shaul Markovitch (1990). "Exploration strategies for model-based learning in multi-agent systems". In: *Autonomous Agents and Multi-agent Systems* 2, pp. 173–207.

Carroll, James L and Kevin Seppi (2005). "Task similarity measures for transfer in reinforcement learning task libraries". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. IEEE, pp. 803–808.

Carroll, Micah, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan (2019). "On the Utility of Learning about Humans for Human-AI Coordination". In: *Advances in Neural Information Processing Systems*, pp. 5175–5186.

Cassandra, Anthony R., Leslie Pack Kaelbling, and Michael L. Littman (1994). "Acting Optimally in Partially Observable Stochastic Domains". In: *Twelfth National Conference on Artificial Intelligence*. AAAI Classic Paper Award, 2013.

Cemgil, Taylan, Sumedh Ghaisas, Krishnamurthy Dvijotham, Sven Gowal, and Pushmeet Kohli (2020). "The Autoencoding Variational Autoencoder". In: *Advances in Neural Information Processing Systems* 33, pp. 15077–15087.

Chalkiadakis, Georgios (2007). *A Bayesian approach to multiagent reinforcement learning and coalition formation under uncertainty*. University of Toronto.

Chalkiadakis, Georgios and Craig Boutilier (2003). "Coordination in multiagent reinforcement learning: A bayesian approach". In: *International Joint Conference on Autonomous Agents and Multiagent systems*, pp. 709–716.

Chalkiadakis, Georgios, Edith Elkind, Evangelos Markakis, Maria Polukarov, and Nick R Jennings (2010). "Cooperative games with overlapping coalitions". In: *Journal of Artificial Intelligence Research* 39, pp. 179–216.

Chin, Lillian, Jeffrey Lipton, Michelle C Yuen, Rebecca Kramer-Bottiglio, and Daniela Rus (2019). "Automated recycling separation enabled by soft robotic material classification". In: *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)*. IEEE, pp. 102–107.

Cho, Kyunghyun, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio (2014). "On the properties of neural machine translation: Encoder-decoder approaches". In: *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8)*.

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

Christiano, Paul F, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei (2017). "Deep reinforcement learning from human preferences". In: *Advances in Neural Information Processing Systems*.

Chung, Junyoung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio (2015). "A recurrent latent variable model for sequential data". In: *Advances in Neural Information Processing Systems*.

Ciosek, Kamil, Vincent Fortuin, Ryota Tomioka, Katja Hofmann, and Richard Turner (2020). "Conservative Uncertainty Estimation by Fitting Prior Networks". In: *International Conference on Learning Representation*.

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

Cobbe, Karl, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman (2019). "Quantifying generalization in reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 1282–1289.

Collins, Jack, Shelvin Chand, Anthony Vanderkop, and David Howard (2021). "A review of physics simulators for robotic applications". In: *IEEE Access*.

Collins, Liam, Aryan Mokhtari, Sewoong Oh, and Sanjay Shakkottai (2022). "MAML and ANIL Provably Learn Representations". In: *arXiv preprint arXiv:2202.03483*.

Dann, Christoph, Lihong Li, Wei Wei, and Emma Brunskill (2019). "Policy certificates: Towards accountable reinforcement learning". In.

Delétang, Grégoire, Jordi Grau-Moya, Markus Kunesch, Tim Genewein, Rob Brekelmans, Shane Legg, and Pedro A Ortega (2021). "Model-Free Risk-Sensitive Reinforcement Learning". In: *arXiv preprint arXiv:2111.02907*.

Dorfman, Ron, Idan Shenfeld, and Aviv Tamar (2021). "Offline Meta Reinforcement Learning – Identifiability Challenges and Effective Data Collection Strategies". In: *Advances in Neural Information Processing Systems*.

Doshi, Finale, Joelle Pineau, and Nicholas Roy (2008). "Reinforcement learning with limited reinforcement: Using Bayes risk for active learning in POMDPs". In: *International Conference on Machine Learning*.

Doshi-Velez, Finale and George Konidaris (2016). "Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations". In: *International Joint Conference on Artificial Intelligence*.

Du, Yuqing, Olivia Watkins, Trevor Darrell, Pieter Abbeel, and Deepak Pathak (2021). "Auto-tuned sim-to-real transfer". In: *2021 IEEE International Conference on Robotics and Automation*. IEEE, pp. 1290–1296.

Duan, Yan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba (2017). "One-shot imitation learning". In: *Advances in Neural Information Processing Systems* 30.

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

Duff, Michael O'Gordon and Andrew Barto (2002). "Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes". PhD thesis. Univ of Massachusetts at Amherst.

Dumoulin, Vincent, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville (2016). "Adversarially learned inference". In: *arXiv preprint arXiv:1606.00704*.

Everitt, Tom and Marcus Hutter (2016). "Avoiding wireheading with value reinforcement learning". In: *International Conference on Artificial General Intelligence*. Springer, pp. 12–22.

Fabius, Otto and Joost R Van Amersfoort (2015). "Variational recurrent auto-encoders". In: *ICLR Workshop*.

Fakoor, Rasool, Pratik Chaudhari, Stefano Soatto, and Alexander J Smola (2020). "Meta-q-learning". In: *International Conference on Learning Representations*.

Feng, Leo, Luisa Zintgraf, Bei Peng, and Shimon Whiteson (2019). "VIABLE: Fast Adaptation via Backpropagating Learned Loss". In.

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

Finn, Chelsea, Kelvin Xu, and Sergey Levine (2018). "Probabilistic Model-Agnostic Meta-Learning". In.

Finn, Chelsea, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine (2017b). "One-shot visual imitation learning via Meta-Learning". In: *Conference on Robot Learning*.

Flennerhag, Sebastian, Yannick Schroecker, Tom Zahavy, Hado van Hasselt, David Silver, and Satinder Singh (2022). "Bootstrapped Meta-Learning". In: *International Conference on Learning Representations*.

Foerster, Jakob N, Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling (2019). "Bayesian action decoder for deep multi-agent reinforcement learning". In: *International Conference on Machine Learning*.

Franceschi, Luca, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil (2018). "Bilevel programming for hyperparameter optimization and Meta-Learning". In: *International Conference on Machine Learning*. PMLR, pp. 1568–1577.

Gao, Guanyu, Jie Li, and Yonggang Wen (2019). "Energy-efficient thermal comfort control in smart buildings via deep reinforcement learning". In: *arXiv preprint arXiv:1901.04693*.

Garnelo, Marta, Dan Rosenbaum, Chris J Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo J Rezende, and SM Eslami (2018a). "Conditional Neural Processes". In: *International Conference on Machine Learning*.

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

Ghavamzadeh, Mohammad, Shie Mannor, Joelle Pineau, Aviv Tamar, et al. (2015). "Bayesian reinforcement learning: A survey". In: *Foundations and Trends® in Machine Learning* 8.5-6, pp. 359–483.

Gmytrasiewicz, Piotr J and Prashant Doshi (2005). "A framework for sequential planning in multi-agent settings". In: *Journal of Artificial Intelligence Research* 24, pp. 49–79.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). "Generative adversarial nets". In: *Advances in Neural Information Processing Systems*.

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

Grant, Erin, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths (2018). "Recasting gradient-based Meta-Learning as hierarchical bayes". In: *International Conference on Learning Representations*.

Griffith, Shane, Kaushik Subramanian, Jonathan Scholz, Charles L Isbell, and Andrea L Thomaz (2013). "Policy shaping: Integrating human feedback with reinforcement learning". In: *Advances in Neural Information Processing Systems* 26.

Guez, Arthur, David Silver, and Peter Dayan (2012). "Efficient Bayes-adaptive reinforcement learning using sample-based search". In: *Advances in Neural Information Processing Systems*.

Guez, Arthur, David Silver, and Peter Dayan (2013). "Scalable and efficient Bayes-adaptive reinforcement learning based on Monte-Carlo tree search". In: *Journal of Artificial Intelligence Research* 48, pp. 841–883.

Gupta, Abhishek, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine (2018). "Meta-Reinforcement Learning of Structured Exploration Strategies". In: *Advances in Neural Information Processing Systems*.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International Conference on Machine Learning*.

Hadfield-Menell, Dylan, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan (2017). "Inverse reward design". In: *Advances in Neural Information Processing Systems* 30.

Hallak, Assaf, Dotan Di Castro, and Shie Mannor (2015). "Contextual markov decision processes". In: *arXiv preprint arXiv:1502.02259*.

Hausknecht, Matthew and Peter Stone (2015). "Deep recurrent q-learning for partially observable mdps". In: *AAAI fall symposium series*.

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

He, He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III (2016a). "Opponent modeling in deep reinforcement learning". In: *International Conference on Machine Learning*, pp. 1804–1813.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *IEEE International Conference on Computer Vision*, pp. 1026–1034.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016b). "Deep residual learning for image recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Heap, Shaun Hargreaves and Yanis Varoufakis (2004). *Game theory: a critical text.* Psychology Press.

Hernandez-Leal, Pablo, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote (2017). "A survey of learning in multiagent environments: Dealing with non-stationarity". In: *arXiv preprint arXiv:1707.09183.*

Hessel, Matteo, Hado van Hasselt, Joseph Modayil, and David Silver (2019). "On inductive biases in deep reinforcement learning". In: *arXiv preprint arXiv:1907.02908.*

Hoang, Trong Nghia (2014). "New Advances on Bayesian and Decision-Theoretic Approaches for Interactive Machine Learning". PhD thesis. Division of the School of Computing, National University of Singapore.

Hoang, Trong Nghia and Kian Hsiang Low (2013). "A general framework for interacting Bayes-optimally with self-interested agents using arbitrary parametric model and model prior". In: *International Joint Conference on Artificial Intelligence.*

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hochreiter, Sepp, A Steven Younger, and Peter R Conwell (2001). "Learning to learn using gradient descent". In: *International Conference on Artificial Neural Networks.*

Hospedales, Timothy, Antreas Antoniou, Paul Micaelli, and Amos Storkey (2020). "Meta-learning in neural networks: A survey". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence.*

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

Hu, Hengyuan and Jakob N Foerster (2019). "Simplified Action Decoder for Deep Multi-Agent Reinforcement Learning". In: *International Conference on Learning Representations.*

Huisman, Mike, Jan N Van Rijn, and Aske Plaat (2021). "A survey of deep Meta-Learning". In: *Artificial Intelligence Review* 54.6, pp. 4483–4541.

Hula, Andreas, P Read Montague, and Peter Dayan (2015). "Monte carlo planning method estimates planning horizons during interactive social exchange". In: *PLoS computational biology* 11.6.

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

Hussein, Ahmed, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne (2017). "Imitation learning: A survey of learning methods". In: *ACM Computing Surveys (CSUR)* 50.2, pp. 1–35.

Igl, Maximilian, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson (2018). "Deep variational reinforcement learning for POMDPs". In: *International Conference on Machine Learning.*

Iglesias, Ana, Paloma Martínez, Ricardo Aler, and Fernando Fernández (2009). "Learning teaching strategies in an adaptive and intelligent educational system through reinforcement learning". In: *Applied Intelligence* 31.1, pp. 89–106.

Iqbal, Shariq and Fei Sha (2018). "Actor-attention-critic for multi-agent reinforcement learning". In: *International Conference on Machine Learning.*

Jaderberg, Max, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. (2017). "Population based training of neural networks". In: *arXiv preprint arXiv:1711.09846*.

Jiang, Minqi, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel (2021). "Replay-guided adversarial environment design". In: *Advances in Neural Information Processing Systems* 34.

Jiang, Nan, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Robert E Schapire (2017). "Contextual decision processes with low Bellman rank are PAC-learnable". In: *International Conference on Machine Learning*.

Julian, Ryan, Benjamin Swanson, Gaurav Sukhatme, Sergey Levine, Chelsea Finn, and Karol Hausman (2021). "Never Stop Learning: The Effectiveness of Fine-Tuning in Robotic Reinforcement Learning". In: *Conference on Robot Learning*. PMLR, pp. 2120–2136.

Kaelbling, Leslie Pack, Michael L Littman, and Anthony R Cassandra (1998). "Planning and acting in partially observable stochastic domains". In: *Artificial intelligence* 101.1-2, pp. 99–134.

Kamienny, Pierre-Alexandre, Matteo Pirotta, Alessandro Lazaric, Thibault Lavril, Nicolas Usunier, and Ludovic Denoyer (2020). "Learning adaptive exploration strategies in dynamic environments through informed policy regularization". In: *arXiv preprint arXiv:2005.02934*.

Kearns, Michael and Satinder Singh (2002). "Near-optimal reinforcement learning in polynomial time". In: *Machine learning* 49.2-3, pp. 209–232.

Killian, Taylor W, Samuel Daulton, George Konidaris, and Finale Doshi-Velez (2017). "Robust and efficient transfer learning with hidden parameter markov decision processes". In: *Advances in Neural Information Processing Systems*.

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

Kiran, B Ravi, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez (2021). "Deep reinforcement learning for autonomous driving: A survey". In: *IEEE Transactions on Intelligent Transportation Systems*.

Kirsch, Louis, Sebastian Flennerhag, Hado van Hasselt, Abram Friesen, Junhyuk Oh, and Yutian Chen (2022). "Introducing Symmetries to Black Box Meta Reinforcement Learning". In: *AAAI Conference on Artificial Intelligence*.

Kirsch, Louis, Sjoerd van Steenkiste, and Jürgen Schmidhuber (2020). "Improving generalization in meta reinforcement learning using learned objectives". In: *International Conference on Learning Representations*.

Kolter, J Zico and Andrew Y Ng (2009). "Near-Bayesian exploration in polynomial time". In: *International Conference on Machine Learning*.

Kolter, J Zico, Andrew Y Ng, et al. (2007). "Learning omnidirectional path following using dimensionality reduction." In: *Robotics: Science and Systems*, pp. 27–30.

Körber, Marian, Johann Lange, Stephan Rediske, Simon Steinmann, and Roland Glück (2021). "Comparing popular simulation environments in the scope of robotics and reinforcement learning". In: *arXiv preprint arXiv:2103.04616*.

Kullback, Solomon (1997). *Information theory and statistics*. Courier Corporation.

Küttler, Heinrich, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel (2020). "The nethack learning

environment". In: *Advances in Neural Information Processing Systems* 33, pp. 7671–7684.

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

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

Lee, Kyungjun and Hernisa Kacorri (2019b). "Hands holding clues for object recognition in teachable machines". In: *CHI Conference on Human Factors in Computing Systems*.

Lee, Seunghyun, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin (2020). "Addressing Distribution Shift in Online Reinforcement Learning with Offline Datasets". In.

Lee, Suyoung and Sae-Young Chung (2021). "Improving Generalization in Meta-RL with Imaginary Tasks from Latent Dynamics Mixture". In: *Advances in Neural Information Processing Systems* 34.

Lee, Yoonho and Seungjin Choi (2018). "Gradient-Based Meta-Learning with Learned Layerwise Metric and Subspace". In: *International Conference on Machine Learning*, pp. 2933–2942.

Levine, Sergey (2021). "Understanding the World Through Action". In: *Conference on Robot Learning, Blue Sky Track*.

Li, Ke and Jitendra Malik (2017a). "Learning to optimize". In: *International Conference on Learning Representations*.

Li, Lihong, Michael L Littman, Thomas J Walsh, and Alexander L Strehl (2011). "Knows what it knows: a framework for self-aware learning". In: *Machine learning* 82.3, pp. 399–443.

Li, Zhenguo, Fengwei Zhou, Fei Chen, and Hang Li (2017b). "Meta-sgd: Learning to learn quickly for few shot learning". In: *arXiv preprint arXiv:1707.09835*.

Lian, Dongze, Yin Zheng, Yintao Xu, Yanxiong Lu, Leyu Lin, Peilin Zhao, Junzhou Huang, and Shenghua Gao (2019). "Towards fast adaptation of neural architectures with Meta-Learning". In: *International Conference on Learning Representations*.

Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2016). "Continuous control with deep reinforcement learning". In: *International Conference on Learning Representations*.

Liu, Bo, Xidong Feng, Haifeng Zhang, Jun Wang, and Yaodong Yang (2021a). "Settling the Bias and Variance of Meta-Gradient Estimation for Meta-Reinforcement Learning". In: *arXiv preprint arXiv:2112.15400*.

Liu, Evan Zheran, Aditi Raghunathan, Percy Liang, and Chelsea Finn (2020). "Decoupling Exploration and Exploitation for Meta-Reinforcement Learning without Sacrifices". In: *International Conference on Machine Learning*.

Liu, Hanxiao, Karen Simonyan, and Yiming Yang (2019). "Darts: Differentiable architecture search". In: *International Conference on Learning Representations*.

Liu, Siqi, Guy Lever, Zhe Wang, Josh Merel, SM Eslami, Daniel Hennes, Wojciech M Czarnecki, Yuval Tassa, Shayegan Omidshafiei, Abbas Abdolmaleki, et al. (2021b). "From motor control to team play in simulated humanoid football". In: *arXiv preprint arXiv:2105.12196*.

Liu, Ziwei, Ping Luo, Xiaogang Wang, and Xiaoou Tang (2015). "Deep Learning Face Attributes in the Wild". In: *International Conference on Computer Vision*.

Luketina, Jelena, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel (2019). "A survey of reinforcement learning informed by natural language". In: *International Joint Conference on Artificial Intelligence*.

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

Massiceti, Daniela, Luisa Zintgraf, John Bronskill, Lida Theodorou, Matthew Tobias Harris, Edward Cutrell, Cecily Morrison, Katja Hofmann, and Simone Stumpf (2021). "Orbit: A real world few-shot dataset for teachable object recognition". In: *IEEE/CVF International Conference on Computer Vision*, pp. 10818–10828.

McCallum, R Andrew (1993). "Overcoming incomplete perception with utile distinction memory". In: *International Conference on Machine Learning*.

Mendonca, Russell, Xinyang Geng, Chelsea Finn, and Sergey Levine (2020). "Meta-reinforcement learning robust to distributional shift via model identification and experience relabeling". In: *arXiv preprint arXiv:2006.07178*.

Mihatsch, Oliver and Ralph Neuneier (2002). "Risk-sensitive reinforcement learning". In: *Machine learning* 49.2, pp. 267–290.

Mikulik, Vladimir, Grégoire Delétang, Tom McGrath, Tim Genewein, Miljan Martic, Shane Legg, and Pedro A Ortega (2020). "Meta-trained agents implement Bayes-optimal agents". In: *Advances in Neural Information Processing Systems*.

Mishra, Nikhil, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel (2018). "A simple neural attentive meta-learner". In: *International Conference on Learning Representations*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533.

Modi, Aditya and Ambuj Tewari (2019). "Contextual markov decision processes using generalized linear models". In: *Reinforcement Learning for Real Life Workshop at ICML*.

Mu, Jesse, Victor Zhong, Roberta Raileanu, Minqi Jiang, Noah Goodman, Tim Rocktäschel, and Edward Grefenstette (2022). "Improving Intrinsic Exploration with Language Abstractions". In: *arXiv preprint arXiv:2202.08938*.

Nachbar, John H (2005). "Beliefs in repeated games". In: *Econometrica* 73.2, pp. 459–480.

Neal, Radford M (2012). *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media.

Ng, Brenda, Kofi Boakye, Carol Meyers, and Andrew Wang (2012). "Bayes-adaptive interactive POMDPs". In: *AAAI Conference on Artificial Intelligence*.

Nichol, Alex and John Schulman (2018). "Reptile: a Scalable Metalearning Algorithm". In: *arXiv preprint arXiv:1803.02999*.

Oh, Jaehoon, Hyungjun Yoo, ChangHwan Kim, and Se-Young Yun (2021). "Boil: Towards representation change for few-shot learning". In: *International Conference on Learning Representations*.

Oh, Junhyuk, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado P van Hasselt, Satinder Singh, and David Silver (2020). "Discovering reinforcement learning algorithms". In: *Advances in Neural Information Processing Systems*.

Oliehoek, Frans A, Christopher Amato, et al. (2014). "Best response Bayesian reinforcement learning for multiagent systems with state uncertainty". In: *AAMAS Workshop on Multiagent Sequential Decision Making Under Uncertainty*.

Oliehoek, Frans A, Christopher Amato, et al. (2016). *A concise introduction to decentralized POMDPs*. Vol. 1. Springer.

Oreshkin, Boris N, Alexandre Lacoste, and Pau Rodriguez (2018). "TADAM: Task dependent adaptive metric for improved few-shot learning". In: *Advances in Neural Information Processing Systems*.

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

Osband, Ian, John Aslanides, and Albin Cassirer (2018). "Randomized prior functions for deep reinforcement learning". In: *Advances in Neural Information Processing Systems*.

Osband, Ian, Daniel Russo, and Benjamin Van Roy (2013). "(More) efficient reinforcement learning via posterior sampling". In: *Advances in Neural Information Processing Systems*.

Osband, Ian, Daniel Russo, Z Wen, and B Van Roy (2017). "Deep exploration via randomized value functions". In: *Journal of Machine Learning Research*.

Ostrovski, Georg, Marc G Bellemare, Aäron van den Oord, and Rémi Munos (2017). "Count-based exploration with neural density models". In: *International Conference on Machine Learning*.

Papoudakis, Georgios and Stefano V Albrecht (2020). "Variational Autoencoders for Opponent Modeling in Multi-Agent Systems". In: *AAAI-20 Workshop on Reinforcement Learning in Games*.

Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer (2017). "Automatic differentiation in pytorch". In.

Pathak, Deepak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell (2017). "Curiosity-driven exploration by self-supervised prediction". In: *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17.

Paul, Supratik, Vitaly Kurin, and Shimon Whiteson (2019). "Fast efficient hyperparameter tuning for policy gradients". In: *Advances in Neural Information Processing Systems*.

Pearce, Tim, Mohamed Zaki, Alexandra Brintrup, Nicolas Anastassacos, and Andy Neely (2020). "Uncertainty in Neural Networks: Approximately Bayesian Ensembling". In: *International Conference on Artificial Intelligence and Statistics*.

Peng, Xue Bin, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel (2018). "Sim-to-real transfer of robotic control with dynamics randomization". In: *2018 IEEE international conference on robotics and automation*. IEEE, pp. 3803–3810.

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

Perez, Ethan, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville (2017). "Film: Visual reasoning with a general conditioning layer". In: *AAAI Conference on Artificial Intelligence*.

Portugal, Ivens, Paulo Alencar, and Donald Cowan (2018). "The use of machine learning algorithms in recommender systems: A systematic review". In: *Expert Systems with Applications* 97, pp. 205–227.

Poupart, Pascal and Nikos Vlassis (2008). "Model-based Bayesian reinforcement learning in partially observable domains". In: *Proc Int. Symp. on Artificial Intelligence and Mathematics,* pp. 1–2.

Poupart, Pascal, Nikos Vlassis, Jesse Hoey, and Kevin Regan (2006). "An analytic solution to discrete Bayesian reinforcement learning". In: *International Conference on Machine Learning*.

Puterman, Martin L (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Quiñonero-Candela, Joaquin, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence (2008). *Dataset shift in machine learning*. MIT Press.

Rabinowitz, Neil, Frank Perbet, Francis Song, Chiyuan Zhang, SM Ali Eslami, and Matthew Botvinick (2018). "Machine Theory of Mind". In: *International Conference on Machine Learning*, pp. 4218–4227.

Raghu, Aniruddh, Maithra Raghu, Samy Bengio, and Oriol Vinyals (2020). "Rapid learning or feature reuse? towards understanding the effectiveness of maml". In: *International Conference on Learning Representations*.

Rahimian, Hamed and Sanjay Mehrotra (2019). "Distributionally robust optimization: A review". In: *arXiv preprint arXiv:1908.05659*.

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

Ravi, Sachin and Hugo Larochelle (2017). "Optimization as a model for few-shot learning". In: *International Conference on Learning Representations*.

Rei, Marek (2015). "Online representation learning in recurrent neural language models". In: *Conference on Empirical Methods in Natural Language Processing*.

Roijers, Diederik M, Johan Jeuring, and Ad Feelders (2012). "Probability estimation and a competence model for rule based e-tutoring systems". In: *International Conference on Learning Analytics and Knowledge*, pp. 255–258.

Rola, Martin D (2007). "Robotic Painting". In: *Products Finishing(Cincinnati)* 71.7, pp. 22–25.

Ross, Stephane, Brahim Chaib-draa, and Joelle Pineau (2008). "Bayes-adaptive pomdps". In: *Advances in Neural Information Processing Systems*, pp. 1225–1232.

Ross, Stéphane, Joelle Pineau, Brahim Chaib-draa, and Pierre Kreitmann (2011). "A Bayesian approach for learning and planning in partially observable Markov decision processes". In: *Journal of Machine Learning Research* 12.May, pp. 1729–1770.

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

Rusu, Andrei A, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell (2019). "Meta-Learning with Latent Embedding Optimization". In: *nternational Conference on Learning Representations*.

Sadigh, Dorsa, S Shankar Sastry, Sanjit A Seshia, and Anca Dragan (2016). "Information gathering actions over human internal state". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 66–73.

Sæmundsson, Steindór, Katja Hofmann, and Marc Peter Deisenroth (2018). "Meta Reinforcement Learning with Latent Variable Gaussian Processes". In: *Conference on Uncertainty in Artificial Intelligence*.

Sarafian, Elad, Shai Keynan, and Sarit Kraus (2021). "Recomposing the Reinforcement Learning Building Blocks with Hypernetworks". In: *International Conference on Machine Learning*. PMLR, pp. 9301–9312.

Schmidhuber, Jürgen (1987). "Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook". PhD thesis. Technische Universität München.

Schmidhuber, Jürgen (1991). "A possibility for implementing curiosity and boredom in model-building neural controllers". In: *International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pp. 222–227.

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz (2015a). "Trust region policy optimization". In: *International Conference on Machine Learning*. PMLR, pp. 1889–1897.

Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel (2015b). "High-dimensional continuous control using generalized advantage estimation". In: *International Conference on Learning Representations*.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347*.

Seita, Daniel (2018). "BDD100k: A large-scale diverse driving video database". In: *The Berkeley Artificial Intelligence Research Blog. Version* 511, p. 41.

Serrino, Jack, Max Kleiman-Weiner, David C Parkes, and Josh Tenenbaum (2019). "Finding Friend and Foe in Multi-Agent Games". In: *Advances in Neural Information Processing Systems*, pp. 1249–1259.

Shapley, Lloyd S (1953). "Stochastic games". In: *the national academy of sciences* 39.10, pp. 1095–1100.

Silva, Andrew and Matthew Gombolay (2019). "Neural-encoding Human Experts' Domain Knowledge to Warm Start Reinforcement Learning". In: *arXiv preprint arXiv:1902.06007*.

Silver, Daniel L, Ryan Poirier, and Duane Currie (2008). "Inductive transfer with context-sensitive neural networks". In: *Machine Learning* 73.3, p. 313.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587, pp. 484–489.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. (2017). "Mastering the game of go without human knowledge". In: *nature* 550.7676, pp. 354–359.

Smith, Laura, J Chase Kew, Xue Bin Peng, Sehoon Ha, Jie Tan, and Sergey Levine (2021). "Legged Robots that Keep on Learning: Fine-Tuning Locomotion Policies in the Real World". In: *arXiv preprint arXiv:2110.05457*.

Smith, Max Olan, Thomas Anthony, Yongzhao Wang, and Michael P Wellman (2020). "Learning to Play against Any Mixture of Opponents". In: *arXiv preprint arXiv:2009.14180*.

Snell, Jake, Kevin Swersky, and Richard Zemel (2017). "Prototypical networks for few-shot learning". In: *Advances in Neural Information Processing Systems*, pp. 4077–4087.

Song, Jiaming, Hongyu Ren, Dorsa Sadigh, and Stefano Ermon (2018). "Multi-agent generative adversarial imitation learning". In: *Advances in Neural Information Processing Systems*, pp. 7461–7472.

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

Southey, Finnegan, Michael P Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner (2012). "Bayes' bluff: Opponent modelling in poker". In: *Conference on Uncertainty in Artificial Intelligence*.

Stadie, Bradly C, Sergey Levine, and Pieter Abbeel (2015). "Incentivizing exploration in reinforcement learning with deep predictive models". In: *arXiv preprint arXiv:1507.00814*.

Stadie, Bradly C, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever (2018). "Some considerations on learning to explore via meta-reinforcement learning". In.

Stone, Peter, Gal A Kaminka, Sarit Kraus, and Jeffrey S Rosenschein (2010). "Ad hoc autonomous agent teams: Collaboration without pre-coordination". In: *AAAI Conference on Artificial Intelligence*.

Strehl, Alexander L and Michael L Littman (2008). "An analysis of model-based interval estimation for Markov decision processes". In: *Journal of Computer and System Sciences* 74.8, pp. 1309–1331.

Strens, Malcolm (2000). "A Bayesian framework for reinforcement learning". In: *International Conference on Machine Learning*. Vol. 2000, pp. 943–950.

Sung, Flood, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales (2018). "Learning to compare: Relation network for few-shot learning". In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1199–1208.

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

Sutton, Richard S (1990). "Integrated modeling and control based on reinforcement learning and dynamic programming". In: *Advances in Neural Information Processing Systems* 3.

Sutton, Richard S (1995). "Generalization in reinforcement learning: Successful examples using sparse coarse coding". In: *Advances in Neural Information Processing Systems* 8.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, Richard S, David McAllester, Satinder Singh, and Yishay Mansour (1999). "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in Neural Information Processing Systems* 12.

Sutton, Richard S, David A McAllester, Satinder P Singh, and Yishay Mansour (2000). "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in Neural Information Processing Systems*, pp. 1057–1063.

Tan, Ming (1997). "Multi-agent reinforcement learning: Independent vs. cooperative learning". In: *Readings in Agents*, pp. 487–494.

Tang, Haoran, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel (2017). "# exploration: A study of

count-based exploration for deep reinforcement learning". In: *Advances in Neural Information Processing Systems* 30.

Teng, Teck-Hou, Ah-Hwee Tan, and Jacek M Zurada (2014). "Self-organizing neural networks integrating domain knowledge and reinforcement learning". In: *IEEE transactions on neural networks and learning systems* 26.5, pp. 889–902.

Thompson, William R (1933). "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples". In: *Biometrika* 25.3/4, pp. 285–294.

Thrun, Sebastian and Lorien Pratt (1998). "Learning to learn: Introduction and overview". In: *Learning to learn*. Springer, pp. 3–17.

Thrun, Sebastian B (1992). "Efficient exploration in reinforcement learning". In.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012a). "Mujoco: A physics engine for model-based control". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 5026–5033.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012b). "MuJoCo: A physics engine for model-based control." In: *IROS*. IEEE, pp. 5026–5033.

Tschiatschek, Sebastian, Kai Arulkumaran, Jan Stühmer, and Katja Hofmann (2018). "Variational inference for data-efficient model learning in POMDPs". In: *arXiv:1805.09281*.

Van Der Wal, J (1981). *Stochastic Dynamic Programming, Mathematical Centre Tracts, vol. 139*.

Veeriah, Vivek, Matteo Hessel, Zhongwen Xu, Janarthanan Rajendran, Richard L Lewis, Junhyuk Oh, Hado P van Hasselt, David Silver, and Satinder Singh (2019). "Discovery of Useful Questions as Auxiliary Tasks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc.

Vinyals, Oriol, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782, pp. 350–354.

Vinyals, Oriol, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. (2016). "Matching networks for one shot learning". In: *Advances in Neural Information Processing Systems*, pp. 3630–3638.

Vuorio, Risto, Jacob Austin Beck, Gregory Farquhar, Jakob Nicolaus Foerster, and Shimon Whiteson (2021). "No DICE: An Investigation of the Bias-Variance Tradeoff in Meta-Gradients". In: *Deep RL Workshop at NeurIPS*.

Wang, Jane X, Michael King, Nicolas Porcel, Zeb Kurth-Nelson, Tina Zhu, Charlie Deck, Peter Choy, Mary Cassin, Malcolm Reynolds, Francis Song, et al. (2021). "Alchemy: A benchmark and analysis toolkit for meta-reinforcement learning agents". In: *arXiv preprint arXiv:2102.02926*.

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

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

Williams, Ronald J (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3, pp. 229–256.

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

Xiong, Zheng, Luisa Zintgraf, Jacob Beck, Risto Vuorio, and Shimon Whiteson (2021). "On the Practical Consistency of Meta-Reinforcement Learning Algorithms". In: *Workshop on Meta-Learning at NeurIPS*.

Xu, Tianbing, Qiang Liu, Liang Zhao, and Jian Peng (2018a). "Learning to Explore via Meta-Policy Gradient". In: *the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Machine Learning Research. PMLR, pp. 5463–5472.

Xu, Zhongwen, Hado P van Hasselt, Matteo Hessel, Junhyuk Oh, Satinder Singh, and David Silver (2020). "Meta-Gradient Reinforcement Learning with an Objective Discovered Online". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 15254–15264.

Xu, Zhongwen, Hado P van Hasselt, and David Silver (2018b). "Meta-Gradient Reinforcement Learning". In: *Advances in Neural Information Processing Systems*.

Yang, Tianpei, Hongyao Tang, Chenjia Bai, Jinyi Liu, Jianye Hao, Zhaopeng Meng, and Peng Liu (2021). "Exploration in deep reinforcement learning: a comprehensive survey". In: *arXiv preprint arXiv:2109.06668*.

Yannakakis, Georgios N and Julian Togelius (2018). *Artificial intelligence and games*. Vol. 2. Springer.

Yao, Jiayu, Taylor Killian, George Konidaris, and Finale Doshi-Velez (2018). "Direct policy transfer via hidden parameter markov decision processes". In: *LLARLA Workshop, FAIM*.

Yoon, Jaesik, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn (2018). "Bayesian Model-Agnostic Meta-Learning". In: *Advances in Neural Information Processing Systems*, pp. 7343–7353.

Yordanov, Yordan (2019). "Using Instrinsic Motivation for Exploration in Partially Observable Environments". MA thesis. Oxford, UK: University of Oxford.

Yu, Tianhe, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine (2019). "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning". In: *Conference on Robot Learning (CoRL)*. arXiv: `1910.10897 [cs.LG]`.

Zahavy, Tom, Zhongwen Xu, Vivek Veeriah, Matteo Hessel, Junhyuk Oh, Hado van Hasselt, David Silver, and Satinder Singh (2020). "Self-tuning deep reinforcement learning". In: *Advances in Neural Information Processing Systems*.

Zaheer, Manzil, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola (2017). "Deep sets". In: *Advances in Neural Information Processing Systems*.

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

Zhang, Jin, Jianhao Wang, Hao Hu, Tong Chen, Yingfeng Chen, Changjie Fan, and Chongjie Zhang (2021). "Metacure: Meta reinforcement learning with empowerment-driven exploration". In: *International Conference on Machine Learning*. PMLR, pp. 12600–12610.

Zhang, Marvin Mengxin, Henrik Marklund, Nikita Dhawan, Abhishek Gupta, Sergey Levine, and Chelsea Finn (2020). "Adaptive risk minimization: A

Meta-Learning approach for tackling group shift". In: *Advances in Neural Information Processing Systems*.

Zhao, Shengjia, Jiaming Song, and Stefano Ermon (2017). "Learning hierarchical features from deep generative models". In: *International Conference on Machine Learning*. JMLR. org, pp. 4091–4099.

Zheng, Zeyu, Junhyuk Oh, Matteo Hessel, Zhongwen Xu, Manuel Kroiss, Hado Van Hasselt, David Silver, and Satinder Singh (2020). "What can learned intrinsic rewards capture?" In: *International Conference on Machine Learning*. PMLR, pp. 11436–11446.

Zheng, Zeyu, Junhyuk Oh, and Satinder Singh (2018). "On Learning Intrinsic Rewards for Policy Gradient Methods". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc.

Zhou, Fengwei, Bin Wu, and Zhenguo Li (2018). "Deep Meta-Learning: Learning to Learn in the Concept Space". In: *arXiv preprint arXiv:1802.03596*.

Zhu, Pengfei, Xin Li, Pascal Poupart, and Guanghui Miao (2017). "On improving deep reinforcement learning for pomdps". In: *arXiv preprint arXiv:1704.07978*.

Zintgraf, Luisa, Sam Devlin, Kamil Ciosek, Shimon Whiteson, and Katja Hofmann (2021a). "Deep Interactive Bayesian Reinforcement Learning via Meta-Learning (Extended Abstract)". In: *International Conference on Autonomous Agents and MultiAgent Systems*.

Zintgraf, Luisa, Leo Feng, Cong Lu, Maximilian Igl, Kristian Hartikainen, Katja Hofmann, and Shimon Whiteson (2021b). "Exploration in approximate hyper-state space for meta reinforcement learning". In: *International Conference on Machine Learning*.

Zintgraf, Luisa, Sebastian Schulze, Cong Lu, Leo Feng, Maximilian Igl, Kyriacos Shiarlis, Yarin Gal, Katja Hofmann, and Shimon Whiteson (2021c). "VariBAD: Variational Bayes-Adaptive Deep RL via Meta-Learning". In: *Journal of Machine Learning Research* 22.289, pp. 1–39.

Zintgraf, Luisa, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson (2020). "VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning". In: *International Conference on Learning Representation*.

Zintgraf, Luisa M, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson (2019). "Fast Context Adaptation via Meta-Learning". In: *International Conference on Machine Learning*.