Planning with Learned Ignorance-Aware Models

Angelos Filos

Department of Computer Science
University of Oxford

This dissertation is submitted for the degree of
Doctor of Philosophy

Worcester College

Trinity 2022

To my family,
Panagiota, Maria, and Paschalis
♡

Acknowledgments

This thesis would not have been enjoyable or even possible without the support of fantastic individuals and institutions. First and foremost, I would like to express my gratitude to my advisor, Yarin Gal, for providing support and essential feedback throughout the years. Yarin, you have significantly influenced my research while allowing me the time and independence to explore my own path, which has helped me develop as an independent researcher. Moreover, you have introduced me to an array of remarkable researchers and significant opportunities. Your consistent mentoring and encouragement have been invaluable to me. I would also like to thank Sergey Levine, who acted as a co-advisor. Sergey, thank you for always providing helpful advice and insight. I greatly enjoyed our collaboration, even though the COVID-19 lockdown disrupted my visit to your lab. I would like to thank Edward Grefenstette and Jakob Foerster for carefully examining this thesis and providing constructive feedback. I thank Rowan McAllister, Nicholas Rhinehart, Greg Farquhar, Natasha Jaques, Andre Barreto, and Amy Zhang, for being amazing collaborators and mentors. For the work in this thesis, I was lucky to collaborate also with Panos Tigas and Clare Lyle. Thank you for your contributions – I could not have done it without you. I also want to thank the rest of OATML: Sebastian Farquhar, Joost van Amersfoort, Aidan Gomez, Milad Alizadeh, Tim G. J. Rudner, Lewis Smith, Andreas Kirsch, Binxin Ru, Neil Band, Andrew Jesson, Jannik Kossen, Lisa Schut, Gunshi Gupta, and Muhammed T. Razzak, for the interesting discussions and fun times. I would also like to thank all my colleagues at J.P. Morgan: Gregory Sidier, Louis Moussu, Samuel Assefa, Cyrine Chtourou, Mahmoud Mahfouz, Giorgio Vit, Joshua Lockhart, Hans Buehler, Manuela Veloso, Jacobo Roa-Vicens, Tucker Balch. For my colleagues at DeepMind: Andre Barreto, Greg Farquhar, Eszter Vertes, Feryal Behbahani, Zita Marinho, Simon Osindero, Kate Baumli, Matteo Hessel, Hado van Hasselt, David Silver, Diana Borsa, Abram Friesen, Tom Schaul, Ioannis Antonoglou, Wilka Carvalho Carvalho and the RL team, thank you for making me feel so welcome despite the physical distance, and being invested in my internship project. I thank Marilena for assuming and delivering the challenging role of "partner in DPhil during a pandemic" and for her unwavering support. Last but not least, I want to thank my family: Panagiota, Maria, and Pascalis, for always believing in me and encouraging me to be who I am.

Walton Street, Oxford, March 2022

Declaration of originality

I, Angelos Filos hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Angelos Filos
Trinity 2022

Abstract

One of the goals of artificial intelligence research is to create decision-makers (i.e., *agents*) that improve from experience (i.e., *data*), collected through interaction with an environment. Models of the environment (i.e., *world models*) are an explicit way that agents use to represent their knowledge, enabling them to make counterfactual predictions and *plans* without requiring additional environment interactions. Although agents that plan with a perfect model of the environment have led to impressive demonstrations, e.g., superhuman performance in board games, they are limited to problems their designer can specify a perfect model. Therefore, *learning* models from experience holds the promise of going beyond the scope of their designers' reach, giving rise to a self-improving vicious circle of (i) learning a model from the past experience; (ii) planning with the learned model; and (iii) interacting with the environment, collecting new experiences. Ideally, learned models should *generalise* to situations beyond their training regime. Nonetheless, this is ambitious and often unrealistic when finite data is used for learning the models, leading to generally imperfect models, with which naive planning could be catastrophic in novel, *out-of-training distribution* situations. A more pragmatic goal is to have agents that are aware of and quantify their lack of knowledge (i.e., *ignorance* or *epistemic uncertainty*).

In this thesis, we motivate and demonstrate the effectiveness of and propose novel ignorance-aware agents that plan with learned models. Naively applying powerful planning algorithms to learned models can render negative results, when the planning algorithm exploits the model imperfections in out-of-training distribution situations. This phenomenon is often termed overoptimisation and can be addressed by optimising ignorance-augmented objectives, called knowledge equivalents. We verify the validity of our ideas and methods in a number of problem settings, including learning from (i) expert demonstrations (*imitation learning*, §3); (ii) sub-optimal demonstrations (*social learning*, §4); and (iii) interacting with an environment with rewards (*reinforcement learning*, §5). Our empirical evidence is based on simulated autonomous driving environments, continuous control and video games from pixels and didactic small-scale grid-worlds. Throughout the thesis, we use *neural networks* to parameterise the (learnable) models and either use existing scalable approximate ignorance quantification *deep learning* methods, such as ensembles, or introduce novel planning-specific ways to quantify the agents' ignorance.

The main chapters of this thesis are based on publications (Filos et al., 2020, 2021, 2022).

Contents

List of figures

List of tables

List of acronyms

ΨΦL	ΨΦ-learning
a.k.a.	also known as
AdaRIP	adaptive robust imitative planning
AI	artificial intelligence
BC	behaviour cloning
CE	certainty equivalent
CMP	controlled Markov process
CVaR	conditional value at risk
DIM	deep imitative model
DL	deep learning
e.g.	exempli gratia ("for the sake of an example")
EMVE	explicit model value ensemble
EU	expected utility
EVE	explicit value ensemble
GPE	generalised policy evaluation
GPI	generalised policy improvement
GRU	gated recurrent unit
i.e.	id est ("it is")
i.i.d.	independent and identically distributed

iff	if and only if
IL	imitation learning
IRL	inverse reinforcement learning
ITD	inverse temporal difference
IVE	implicit value ensemble
KE	knowledge equivalent
KL	Kullback–Leibler
LIDAR	light detection and ranging
MAP	maximum a posteriori
MC	Monte Carlo
MDP	Markov decision process
ML	machine learning
MLE	maximum likelihood estimation
MLP	multi-layer perceptron
MPC	model predictive control
MPV	model-predicted value
NN	neural network
OOD	out-of-training distribution
PID	proportional–integral–derivative controller
POMDP	partially observed Markov decision process
RIP	robust imitative planning
RL	reinforcement learning
s.t.	such that
SF	successor feature
SGD	stochastic gradient descent

SL	supervised learning
TD	temporal difference
w.r.t.	with respect to

List of symbols

In §2 we provide details about the nomenclature and notation used throughout this thesis.

Φ	cumulants
Ψ	successor feature
$\mathbb{A}$	action space
$\mathcal{B}$	experience
$\mathcal{C}$	controlled Markov process
$\mathcal{D}$	demonstrations
γ	discount factor
$\mathcal{M}$	Markov decision process
$\mathfrak{m}^*$	"true" model of the environment
p	(Markov) transition dynamics
π^*	optimal policy
q	action value (function)
r	reward distribution (function)
$\mathbb{S}$	state space
$\mathscr{T}$	Bellman evaluation operator
Θ	hypothesis space
θ	parameters of learnable function

u	utility function
v	value (function)

"Real knowledge is to know the extent of one's ignorance."
— Confucius (551–479BC)

1 Introduction

Contents

Reasoning about and acting under uncertainty is a hallmark of human intelligence. We arguably make more decisions when facing unknowns than otherwise. But not all the unknowns are the same. On the one hand, there is *risk* also known as (a.k.a.) *aleatoric* or known uncertainty, which refers to uncertainty in familiar situations, e.g., the flip of a coin, or the delay in the train schedule. On other other hand, there is *ignorance* a.k.a. *epistemic* or unknown uncertainty, which refers to uncertainty in unfamiliar situations. Most of the times, we do not know what we are ignorant about until we encounter it for the first time. For instance, when we read a book and come across a "new" word.

When it comes to artificial decision-makers (i.e., *agents*) the story is different. First of all, with the advent of the microprocessor, fast digital computers enabled the development of artificial agents that operate in either fully-certain (Lindsay et al., 1993; Shortliffe, 1977) or closed-world uncertain settings (Markowitz, 1952; Smigel, 2022). Over the last few decades, the progress in such settings has been rapid. Key components to the success have been *planning* and *learning*. Planning refers to the process of thinking about and deciding on a series of actions for achieving a goal in the future. Research on animal and human decision-making (James, 1890; Kahneman, 2011) suggests that logical and deliberate decisions are planned, using an explicit or implicit *model* of the environment (i.e., *world model*). Artificial decision-makers that plan with a perfect model of the environment (Samuel, 1959; Silver et al., 2016; Brown and Sandholm, 2019), while impressive, are limited to problems for which their designer can specify a perfect model (e.g., board/video games). Learning is the process of gaining knowledge. As long as the learning process does not depend on an omniscient designer/supervisor, learned models hold the promise of expanding the range of problems that can be addressed with planning, relying only on data (i.e., *experience*) and weak inductive biases (Sutton, 1991). However, learned models are generally imperfect, and naive planning with them could be catastrophic (Zhou et al., 1996).

To unlock the learning and planning to (real) open-world settings, agents should be equipped with effective mechanisms for reasoning about their ignorance, and for factoring it into their decisions. Agents should be able to say "I do not know" and act accordingly, e.g., either taking a safer course of action avoiding to make decisions that would lead to novel situations or follow an exploratory strategy, aiming to learn about what they do not know. In other words, we need learning agents that quantify and use their ignorance.

1.1 Thesis statement

We now highlight the central question addressed in this work:

Central Question

How do generally capable bounded artificial agents learn and use models to plan?

We answer this question by advancing the following thesis:

Thesis

By drawing on insights from information theory, decision theoretic planning and robust control theory, it is possible to design efficient algorithms for learning useful models for planning that are robust to distribution shifts, enable fast online adaptation and scale to real-world problems.

To defend this thesis, we introduce five desiderata that articulate which learned models are useful for planning. At a high level, these desiderata state the following:

Desiderata

Good models are $\underbrace{\text{easy to plan with}}_{\textbf{D1}}$ and $\underbrace{\text{enable efficient learning}}_{\textbf{D2}}$ of $\underbrace{\text{high value}}_{\textbf{D3}}$, $\underbrace{\text{adaptable}}_{\textbf{D4}}$, $\underbrace{\text{robust policies}}_{\textbf{D5}}$.

We present more detail and justification for these desiderata in §2.

1.2 Contributions and outline

The central chapters of this dissertation are based on published papers. More specifically, §3, §4 and §5 are based on papers published in conference proceedings (Filos et al., 2020, 2021, 2022). In the rest of the chapters (§2 and §6), the majority of the contributions are original to this dissertation. The remaining defence of this thesis is organised as follows.

Chapter 2: Background

We start by reviewing decision-making under uncertainty (§2.1), providing formal definitions for risk, ignorance and decision-theoretic concepts such as *knowledge equivalents* (KE). The latter is a key idea that we use in formalising and characterising ignorance-aware agents. In particular, a KE is the amount of reward/utility an agent would be willing to trade with its ignorance. For example, an ignorance-averse (resp. -seeking) agent would accept a reduced (resp. increased) reward in order to avoid (resp. "find") ignorance. Moreover, we provide background on supervised learning (§2.2) and, in particular, scalable ignorance quantification methods for deep neural networks (§2.2.3), which are the mechanisms we use in the remaining chapters for capturing agents' ignorance in scale. Then, we review the reinforcement learning (RL) problem and solution methods (§2.3). Specifically, we study planning agents (§2.3.3) and highlight the importance of ignorance quantification and KEs (§2.3.5) to overcome the phenomenon of overoptimisation of learned objectives (§2.10), which is the foundation of most failures of planning agents.

These are the main conceptual tools that subsequent chapters of this thesis build upon. To help the reader and declutter 2, any background concepts and assumptions required only for a specific chapter are introduced within the corresponding chapter.

Chapter 3: Plan and adapt from expert demonstrations

We study imitation learning (IL) agents that learn from (static) expert demonstrations and demonstrate that ignorance-unaware agents are susceptible to distribution shifts due to the overoptimisation phenomenon. Then, we propose a novel planning agent, called *robust imitative planning* (RIP), and an adaptive, online variant of it, called *adaptive robust imitative planning* (AdaRIP), which train an ignorance-aware model and plan against an ignorance-averse KE, adhering to the principle of "pessimism in the face of ignorance". The proposed methods outperform the state-of-the-art methods in both prediction and control tasks in a simulated autonomous driving environment from high-dimensional LIDAR observations.

This chapter is based on the following publication:

Filos et al. (2020)

Angelos Filos, Panagiotis Tigkas, Rowan McAllister, Nicholas Rhinehart, Sergey Levine, and Yarin Gal. Can Autonomous Vehicles Identify, Recover From, and Adapt to Distribution Shifts? In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3145–3153. PMLR, 2020.

Chapter 4: Plan and adapt from sub-optimal demonstrations

We relax the assumption from the previous chapter of having access to expert demonstrations and, instead, we study RL agents that learn from sub-optimal demonstrations in order to bootstrap their trial-and-error learning. To do so, we present two novel algorithms for learning and planning with temporal difference (TD) learning. The first, *inverse temporal difference* (ITD) *learning*, is an offline multi-task inverse reinforcement learning (IRL) algorithm for discovering salient task-agnostic environment features from sub-optimal demonstrations without rewards. The second, $\Psi\Phi$*-learning* ($\Psi\Phi$L), combines ITD learning with RL from online experience. Similar to the previous chapter, the $\Psi\Phi$-learner plans against an ignorance-averse KE to avoid overoptimisation, and we show that it scales gracefully to large scale IL, IRL, RL and few-shot RL settings.

This chapter is based on the following publication:

Filos et al. (2021)

Angelos Filos, Clare Lyle, Yarin Gal, Sergey Levine, Natasha Jaques, and Gregory Farquhar. PsiPhi-Learning: Reinforcement Learning with Demonstrations using Successor Features and Inverse Temporal Difference Learning. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3305–3317. PMLR, 2021.

Chapter 5: Planning with model-value inconsistency

In both previous chapters, in practice, the ignorance quantification mechanism was implemented with deep ensembles. Despite ensembles' simplicity, they are not efficient since N copies of the model have to be trained and queried at test (i.e., acting) time, increasing the computational cost by a factor N. In this chapter, we present a novel proxy signal for capturing reinforcement learning (RL) agents' ignorance, termed *model-value inconsistency* or *self-inconsistency* for short. This signal is calculable by any agent equipped with a learned (world) model and value function, and it is readily integrable to planning objectives since it has the same units as the agents' values. We provide empirical evidence in both tabular and function approximation settings from pixels that self-inconsistency is an effective signal to quantify agents' ignorance, while being simpler and more computationally efficient than the standard methods from the literature.

This chapter is based on the following publication:

Filos et al. (2022)

Angelos Filos, Eszter Vértes, Zita Marinho, Gregory Farquhar, Diana Borsa, Abram Friesen, Feryal Behbahani, Tom Schaul, Andre Barreto, and Simon Osindero. Model-Value Inconsistency as a Signal for Epistemic Uncertainty. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 6474–6498. PMLR, 2022.

Chapter 6: Discussion

We conclude with some insights and provide our outlook for future research.

Chapters A-C: Supplementary material

We provide supporting material for main chapters, including details on experimental setups, implementations, ablation studies, proofs, visualisations and extensions.

"One's first step in wisdom is to question everything – and one's last step is to come to terms with everything."

— Georg Christoph Lichtenberg (1742–1799)

2 Background & literature review

Contents

In this chapter, we provide the preliminary material, i.e., necessary background and literature to contextualise the contributions of this thesis, and introduce the notation that will be used throughout the thesis, borrowed from Filos et al. (2020, 2021, 2022). Where necessary, background concepts, related work, and notation required only for a specific chapter are introduced within the corresponding chapter, in a dedicated "background" section.

In §2.1, we introduce decision-making under uncertainty. We define and motivate terms, such as *risk*, *ignorance*, *certainty equivalent* and *knowledge equivalent*. In §2.2, we review *supervised learning* models and algorithms and, in particular, ensembles of deep *neural networks* and how they can be used for ignorance quantification. In §2.3, we present the *reinforcement learning* problem, in which a sequential decision-maker (i.e., *agent*) acts in an unknown environments, and review solution methods from the literature.

Nomenclature. We denote random variables with uppercase letters, e.g., X, Θ, and their values with lowercase letters, e.g., x, θ. We reserve blackboard bold letters for sets or continuous spaces, e.g., $\mathbb{X}$, and we use $\Delta(\mathbb{X})$ to denote the space of probability distributions over $\mathbb{X}$. We use superscripts $\bullet^k$ to indicate internal steps of the agent's model, which have no necessary connection to time steps $\bullet_t$ of the environment. We reserve the Greek letter θ for function parameters from the hypothesis space $\mathbb{\Theta} \subseteq \mathbb{R}^n$ and subscript it with the function name when necessary, e.g., $\theta_f \in \mathbb{\Theta}_f$, $\theta_g \in \mathbb{\Theta}_g$ for functions f and g, respectively. We separate the function parameters from other inputs with a semicolon, e.g., $f(x, z; \theta_f) \equiv f_{\theta_f}$. We denote functions of interest (i.e., *target* functions) with an asterisk superscript, e.g., f^*, g^*, and our approximation with hat notation, e.g., $\hat{f}$, $\hat{g}$.

2.1 Decision-making under uncertainty

We study a decision-maker (i.e., *agent*), that issues, for simplicity, a *single* action $A \sim \pi$ and receives scalar reward $R \sim r(A)$, where $\pi \in \Delta(\mathbb{A})$ is the agent's policy, $\mathbb{A}$ is the action space, i.e., set of admissible actions, and $r : \mathbb{A} \to \Delta(\mathbb{R})$ is the (stochastic) reward function. The agent's goal is to find a (stochastic) policy π^* that leads to high rewards.

The agent's uncertainty stems from (i) the inherent and irreducible stochasticity of the reward function (i.e., *risk*), and (ii) the fact that initially the reward function is unknown to the agent (i.e., *ignorance*) and can learn about it by taking actions and observing their consequences. Next, we expand on the distinction of these two types of uncertainty.

> ***Remark 2.1*** (types of uncertainty for decision-making)***:*** There are two fundamental types of uncertainty in decision-making: risk and ignoranc (Knight, 1921). Risk, also known as (a.k.a.) *aleatoric* or known or closed-world or irreducible uncertainty, refers to the uncertainty in familiar situations where the exact outcome of an event is uncertain but probabilities can be computed, as in coin flips or dice. In contrast, ignorance, a.k.a. *epistemic* or unknown or open-world or reducible or model uncertainty, applies to unfamiliar situations where the probabilities are unknown or cannot be determined, e.g., answering whether "Cydophines are also Abordites" without knowing the meaning of these terms (Gilboa et al., 2009; Grau-Moya et al., 2022).

We can think of an agent as taking an action in two steps: (i) assigning a scalar value to each action, which incorporates all sources of uncertainty; and (ii) making a decision.

Next, we review decision-making under risk, i.e., we assume that the agent has direct access to the reward function and has to deal with the fact that rewards are stochastic.

2.1.1 Decision-making under risk

For stochastic reward functions, the designer of the agent has to pick a mechanism for reducing the multiple reward outcomes to a single representative "optimisable" scalar value, which is also known as (a.k.a.) the *certainty equivalent* (CE) in the economics literature (Bernoulli, 1738; Von Neumann and Morgenstern, 1944; Savage, 1954).

> ***Remark 2.2*** (certainty equivalent (CE) in layman's terms)***:*** CE is, as the name suggests, the *guaranteed* reward that the agent would accept, rather than taking a chance on a higher, but *uncertain*, in terms of risk, reward. It is a systematic way to evaluate risk and convert it into the same unit with rewards.

Formally, one can think of CEs as functions that reduce distributions over rewards into a single (deterministic) scalar, i.e., mappings of the form $\Delta(\mathbb{R}) \to \mathbb{R}$. For instance, the expected value or the standard deviation or the p-percentile mean are valid such mappings.

For the rest of this section, we use the *Figure 2.2a* to explain and motivate different CEs.

Expected utility. In economics (Rubinstein, 1998) and reinforcement learning (§2.3, Sutton and Barto, 2018), we usually conform with the *expected utility* (EU) paradigm, in which an agent assigns CEs equal to the expected reward (Von Neumann and Morgenstern, 1944):

$$u_{\text{EU}}(a) \triangleq \mathbb{E}_{R\sim r(a)}[R]. \tag{2.1}$$

For instance, in *Figure 2.2a*, the EU for both actions is equal to 0 and hence the agent is indifferent between them, despite the fact that the two reward distributions differ in terms of other statistics, e.g., entropy. Therefore we say that the EU is a *risk-neutral* CE, i.e., insensitive to the shape of the distribution over rewards except for the expected value.

Risk-sensitivity. In contrast, other CEs take into account higher-order moments of the reward distribution, e.g., the mean-variance (utility) CE (Markowitz, 1952):

$$u_{\text{MVU}[\beta]}(a) \triangleq \mathbb{E}_{R\sim r(a)}[R] + \beta \text{Var}_{R\sim r(a)}[R], \tag{2.2}$$

where by varying $\beta \in \mathbb{R}$ different risk profiles are induced: for (i) $\beta < 0$ we obtain a *risk-averse* CE; (ii) $\beta > 0$ a *risk-seeking* CE; and (iii) $\beta = 0$ we recover EU, i.e., risk-neutral. There are other more "extreme" risk-sensitive CEs that are only sensitive to the minimum (i.e., risk-averse) or the maximum (i.e., risk-seeking) admissible value of the reward distribution (Arrow, 1965; Pratt, 1978; Whittle, 1981; Pichler and Schlotter, 2020):

$$u_{\mathrm{WCU}}(a) \triangleq \min_{R \in \mathrm{support}(r(a))} R \quad \text{and} \quad u_{\mathrm{BCU}}(a) \triangleq \max_{R \in \mathrm{support}(r(a))} R, \tag{2.3}$$

where support$(p_{\mathbb{X}})$ denotes the *support* of the $p_{\mathbb{X}}$ distribution, i.e., informally, the values of $x \in \mathbb{X}$ for which there is non-zero probability under $p_{\mathbb{X}}$. Intuitively, the u_{WCU} (resp. u_{BCU}) CE represents the reward distribution with the worst (resp. best) -case outcome. Therefore, we often refer to this CE as an instance of the principle of "pessimism (resp. optimism) in the face of risk" (Markowitz, 1952; Arrow, 1965; Pratt, 1978).

For example, in *Figure 2.2a*, risk-sensitive CEs value the two actions differently. In particular, risk-averse (resp. risk-seeking) CEs evaluate the `LEFT` (resp. `RIGHT`) action higher.

> ***Remark 2.3*** (conforming to expected utility (EU) by default)***:*** For the rest of this thesis, unless stated otherwise, we design agents that conform to the expected utility (EU) paradigm, i.e., a risk-neutral certainty equivalent (CE).

CEs provide a systematic mechanism to evaluate decisions under risk and hence derive policies for settings with known unknowns. Next, we review decisions under ignorance.

2.1.2 Decision-making under ignorance

If the reward function is (partially) unknown, the agent acts under ignorance. Formally, let parameters θ_r^* from hypothesis space Θ_r that fully specify the "true" reward function, i.e.,

$$r(\cdot; \theta_r^*) \equiv r_{\theta_r^*} \triangleq r. \tag{2.4}$$

For the purposes of this thesis, the agent's ignorance about the reward function is equivalent to uncertainty about θ_r^*. We adopt a probabilistic approach, treating $\theta_r \in \Theta_r$ as a random variable: The agent's (and its designer's) initial belief about the reward function is encoded in the *prior* $p(\Theta_r)$, and upon observing N samples (data) from the reward function $\mathcal{D} \triangleq \{r^{(n)} | r^{(n)} \sim r_{\theta_r^*}(\cdot)\}_{n=1}^{N}$, it forms a *posterior* belief $p(\Theta_r|\mathcal{D})$ via probabilistic inference, as depicted in *Figure 2.1*.[†] In the limit of infinite samples, i.e., $N \to \infty$, the posterior distribution concentrates at θ_r^* and the agent "knows" everything about the reward function, or it is not ignorant. Otherwise, the agent needs to factor its ignorance about the reward function into its decision-making, extending the framework CEs (see §2.1.1), which we review next.

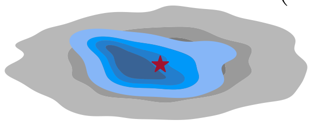


Figure 2.1: The agent's beliefs about the "true" reward function θ_r^*: The uninformed initial (a.k.a. prior) belief $p(\Theta_r)$ and the more concentrated posterior belief $p(\Theta_r|\mathcal{D})$ after observing reward samples $\mathcal{D}$.

[†]We provide a comprehensive review of probabilistic inference methods in §2.2.1 and §2.2.3.

Knowledge equivalent. Akin to CEs for decision-making under risk in §2.1.1, an agent integrates its ignorance about the unknown reward function into a single representative "optimisable" scalar value, which we term *knowledge equivalent* (KE). In the economics literature (Knight, 1921; Wald, 1939; Ellsberg, 1961), often, there is no distinction between CE and KE, instead CE is used to refer to both. In this work, we introduce the term KE to highlight the difference between decision-making under risk and ignorance.

> ***Remark 2.4*** (knowledge equivalent (KE) in layman's terms)***:*** KE is, as the name suggests, the *guaranteed* reward that the agent would accept, rather than taking a chance on a higher, but *unknown* (i.e., *uncertain* in terms of ignorance) reward. It is a systematic way to evaluate ignorance and convert it into the same unit with rewards.

For example, in *Figure 2.2b*, the agent is presented with two options, where the reward for the `RIGHT` action is deterministic but unknown to the agent—it can be any colour from red (-1), white (0) or green ($+1$). If the agent is allowed to make repeated decisions, as long as it selects the `RIGHT` action once, from that point onward it will know the colour of the marble, i.e., it will eliminate its ignorance about the reward function, and from its perspective, the reward function will be deterministic[†] and known. This is different from the example in *Figure 2.2a*, in which the agent knows the colours of the marbles for the `RIGHT` action and hence the reward function but the draw of the marbles is stochastic. No matter how many times the agent selects the `RIGHT` action, the reward function remains stochastic. This is the main distinction between ignorance (i.e., reducible uncertainty in *Figure 2.2b*) and risk (i.e., irreducible uncertainty in *Figure 2.2a*).

Formally, one can think of KEs as functions that reduce the (e.g., posterior) belief over reward functions parameters into a single reward function, i.e., mappings of the form $\Delta(\Theta_r) \to \theta_r$. For instance, the expected value or the standard deviation are valid such mappings. In the general case, a KE should be combined with a CE to evaluate a decision under both ignorance and risk. Next, we review common in the literature KEs and to keep the presentation general, we denote with the CE for parameters θ_r with $u_{\mathrm{CE}}(\cdot;\theta_r)$.

Model average. A natural candidate for incorporating the agent's ignorance about the reward function, captured by the posterior over parameters θ_r after observing data $\mathcal{D}$, into its decision-making is to assign KE as the expected value under the model posterior:

$$u_{\mathrm{MA}}(a;\mathcal{D}) \triangleq \mathbb{E}_{\theta_r \sim p(\Theta_r|\mathcal{D})} u_{\mathrm{CE}}(a;\theta_r), \tag{2.5}$$

and we refer to this KE as the *model average* (Savage, 1954; Barber, 2012). For example, in *Figure 2.2b* with a uniform prior/posterior, the u_{MA} for both actions is equal to 0 and hence the agent is indifferent between them, in spite of the fact that the two reward distributions differ in terms of other statistics, e.g., variance. Therefore we say that the model average is an *ignorance-neutral* KE, i.e., insensitive to the shape of the distribution.

[†]This is not true in general, it happens to be the case for this didactic example.

Ignorance-sensitivity. Akin to the CEs in §2.1.1, there are ignorance-sensitive KEs that incorporate information about higher-order moments of the posterior distribution:

$$u_{\mathrm{MVM}[\xi]}(a;\mathcal{D}) \triangleq \mathbb{E}_{\theta_r \sim p(\Theta_r|\mathcal{D})} u_{\mathrm{CE}}(a;\theta_r) + \xi \mathrm{Var}_{\theta_r \sim p(\Theta_r|\mathcal{D})} u_{\mathrm{CE}}(a;\theta_r). \tag{2.6}$$

where by varying $\xi \in \mathbb{R}$ different ignorance profiles are induced: for (i) $\xi < 0$ we obtain a *ignorance-averse* KE; (ii) $\xi > 0$ a *ignorance-seeking* KE; and (iii) $\xi = 0$ we recover model average, i.e., ignorance-neutral. There are more "extreme" ignorance-sensitive KEs that are only sensitive to the minimum (i.e., ignorance-averse) or the maximum (i.e., ignorance-seeking) admissible value of the reward distribution (Wald, 1939; Savage, 1954):

$$u_{\mathrm{WCM}}(a;\mathcal{D}) \triangleq \min_{\theta_r \in \mathrm{support}(p(\Theta_r|\mathcal{D}))} u_{\mathrm{CE}}(a;\theta_r) \quad \text{and} \tag{2.7a}$$

$$u_{\mathrm{BCM}}(a;\mathcal{D}) \triangleq \max_{\theta_r \in \mathrm{support}(p(\Theta_r|\mathcal{D}))} u_{\mathrm{CE}}(a;\theta_r). \tag{2.7b}$$

For instance, the evaluation of the two actions in *Figure 2.2b* according to ignorance-sensitive KEs is differently. Specifically, ignorance-averse (resp. ignorance-seeking) KEs evaluate the `LEFT` (resp. `RIGHT`) action higher due to its higher (resp. lower) ignorance.

Example 2.1 (selecting boxes with coloured marbles under uncertainty)***:*** Consider an agent that faces the decision-making problems in *Figure 2.2*, in which it is asked to choose one of $a \in \{\texttt{LEFT}, \texttt{RIGHT}\}$ boxes. Each box has coloured marbles and the agent receives a reward corresponding to the colour of a marble drawn randomly from the chosen box. There are different types of marbles: (i) the white marbles that equal to 0 reward; (ii) the red marbles that equal to -1 reward; (iii) the green marbles that equal to $+1$ reward; and (iv) the blue marbles that have *unknown* reward and you get to only learn about their reward after selecting them.

In *Figure 2.2a*, the agent is presented with a decision-making problem under risk. Its `LEFT` action returns a guaranteed 0 reward but its `RIGHT` action has a 50% chance to return a $+1$ reward and otherwise -1. The agent is uncertain about the reward of the `RIGHT` action but knows exactly the probabilities of each outcome. The agent's decision depends on the certainty equivalent (CE, see §2.1.1) its designer selected:

Certainty equivalent (CE)	$\pi^*(A = \texttt{LEFT})$	$\pi^*(A = \texttt{RIGHT})$
Risk-averse (e.g., $u_{\mathrm{MVU}[\beta<0]}$, u_{WCU})	1.0	0.0
Risk-neutral (e.g., u_{EU})	0.5	0.5
Risk-seeking (e.g., $u_{\mathrm{MVU}[\beta>0]}$, u_{BCU})	0.0	1.0

In *Figure 2.2b*, the agent is presented with a decision-making problem under ignorance. Its `RIGHT` action has uncertain rewards—both the outcomes *and* their probabilities are unknown. By selecting the RIGHT action, the agent can learn about the colour of the marble and *reduce* its uncertainty (i.e., ignorance). The agent's designer can choose a knowledge equivalent (KE, see §2.1.2) that balances the trade-off between ignorance and guaranteed reward, modulating the agent's decision:

Knowledge equivalent (KE)	$\pi^*(A = \texttt{LEFT})$	$\pi^*(A = \texttt{RIGHT})$
Ignorance-averse (e.g., $u_{\text{MVM}[\xi<0]}$, u_{WCM})	1.0	0.0
Ignorance-neutral (e.g., u_{MA})	0.5	0.5
Ignorance-seeking (e.g., $u_{\text{MVM}[\xi>0]}$, u_{BCM})	0.0	1.0

In *Figure 2.2c*, the agent is presented with a decision-making problem under both risk and ignorance. The LEFT action has risk and the RIGHT action has ignorance. To make decisions, the agent has to be provided with both a CE and a KE, which imply a trade-off between risk and ignorance. For example, according to a risk-neutral CE and an ignorance-seeking KE, the agent prefers the RIGHT action.

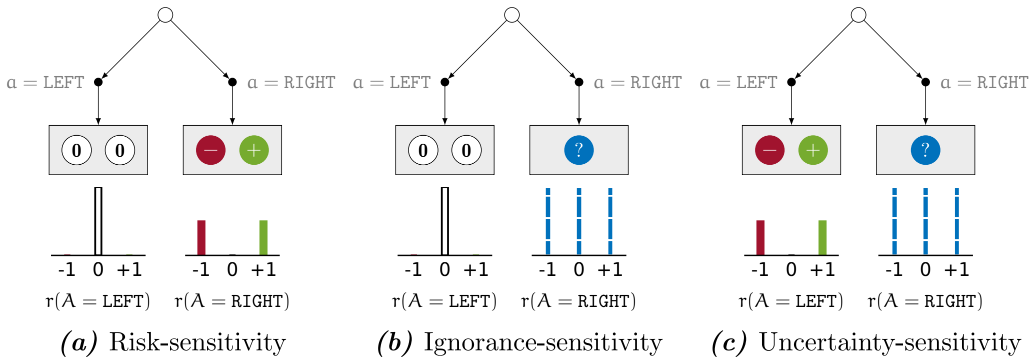


Figure 2.2: Single-step decision-making under uncertainty. The agent selects one of the {LEFT, RIGHT} boxes and receives a reward that depends on the colour of the randomly drawn marbles: 0 reward for white marbles; +1 for green; and −1 for red. ***(a)*** The RIGHT action has an irreducibly uncertain reward (risk). ***(b)*** The reward function for the RIGHT action is unknown (ignorance) and deterministic. ***(c)*** A decision-making problem that involves both risk ($a = \texttt{LEFT}$) and ignorance ($a = \texttt{RIGHT}$).

So far we have presented how to design agents that made decisions under uncertainty, assuming access to learned reward functions (i.e., models) and posterior distributions over their parameters. Next, we review algorithms for learning such models from data.

2.2 Supervised learning (SL)

We present machine learning (ML) algorithms for learning from data an unknown *target* function $f^* : \mathbb{X} \to \mathbb{Y}$, where $\mathbb{X}$, $\mathbb{Y}$ are the "input" and "output" spaces, e.g., identifying the unknown reward function of §2.1.2 from action→reward data. We cast the problem of learning from data as a parameter inference problem for a function approximator:

$$F \triangleq \{f : \mathbb{X} \times \Theta_f \to \mathbb{Y}\}, \tag{2.8}$$

where Θ_f is the hypothesis space of "learnable" parameters, i.e., finding the "best":

$$\theta_f^* \in \Theta_f, \quad \text{such that (s.t.)} \quad f_{\theta_f^*} \approx f^*. \tag{2.9}$$

Data. In supervised learning (SL), we assume access to input-output pair samples from f^*:

$$\mathcal{D}^{\text{train}} \triangleq \left\{\left(x^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \quad \text{s.t.} \quad y^{(n)} = f^*(x^{(n)}) \quad \text{and} \quad \left\{x^{(n)}\right\}_{n=1}^{N} \sim P_{\mathbb{X}}, \tag{2.10}$$

where $P_{\mathbb{X}} \in \Delta(\mathbb{X})$ is the unknown (to the agent) input data (generating) distribution. The objective of a supervised learner is to identify the target function f^*, or, in practice, find the "best" parameters θ_f^* of *Eqn.* (2.9), e.g., using probabilistic inference methods.

Generalisation. The selection of θ_f^* is made based on some data $\mathcal{D}^{\text{train}}$, and the expectation is that it *generalises*, i.e., it performs sufficiently well on some *unseen* data $\mathcal{D}^{\text{test}}$. It can be shown that without any further assumptions on $\mathcal{D}^{\text{test}}$, the problem of generalisation is ill-posed and uninteresting (see "no free lunch theorem", Wolpert and Macready, 1997). In statistical learning theory, if both $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{test}}$ are independent and identically distributed (i.i.d.) samples from a sample space $\mathbb{D}$, i.e., $\mathcal{D}^{\text{train}}, \mathcal{D}^{\text{test}} \overset{\text{i.i.d.}}{\sim} \mathbb{D}$, then, under non-trivial assumptions, the performance on the data $\mathcal{D}^{\text{test}}$ of a model trained on $\mathcal{D}^{\text{train}}$ can be lower-bounded (Vapnik and Chervonenkis, 1971; Baum and Haussler, 1988; McAllester, 1998; Vapnik, 1999; Bousquet and Elisseeff, 2002), providing us with provable guarantees about the performance of a learned model. Nonetheless, the looseness of this bound and the fact that, in practice, the i.i.d. assumption is most of the times violated, make these results over-pessimistic and hence impractical.

Informally, we refer to $\mathcal{D}^{\text{test}}$ and other data from $\mathbb{D}$ for which the i.i.d. assumption holds as the *in-distribution* data, and, otherwise, we call it *out-of-training distribution* (OOD) data, as illustrated in *Figure 2.3*. When using a ML model with OOD data, we say that it experiences a *distribution shift*, and, in this regime, the level of its performance cannot be predicted or guaranteed.

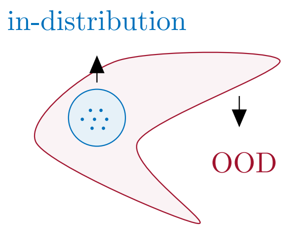


Figure 2.3: Data distribution.

2.2.1 Probabilistic inference

In this section, we treat θ_f as the parameters of a conditional density model:

$$p(Y = y|x; \theta_f)\text{: likelihood of } y \text{ given } x, \underline{\text{under model}} \text{ parameters } \theta_f, \tag{2.11}$$

and the data is assumed to be conditionally independent given model parameters:

$$p(\mathcal{D}^{\text{train}}|\theta_f) = \prod_{n=1}^{N} p(Y = y^{(n)}|x^{(n)}; \theta_f). \tag{2.12}$$

For instance, (i) if $\mathbb{Y}$ is a continuous space, $p(Y = y|x; \theta_f)$ is the density of a conditional multivariate normal distribution, and θ_f is the mean vector and the covariance matrix, and (ii) if $\mathbb{Y}$ is a discrete set, $p(Y = y|x; \theta_f)$ is the probability of a conditional categorical distribution, and θ_f is the vector of event probabilities. Throughout this section, we use the reward modelling *Example* 2.2 to compare different SL methods, in which $p(Y = y|x; \theta_f)$ is a normal distribution with mean θ_f and fixed (not learned) standard deviation parameter.

Bayesian inference. We treat the model parameters as random variables, i.e., we place a prior distribution $p(\Theta_f)$ over possible model parameters $\theta_f \in \mathbb{\Theta}_f$, and after observing the data $\mathcal{D}^{\text{train}}$, we update it to the posterior distribution according to the Bayes' rule:

$$\underbrace{p(\theta_f|\mathcal{D}^{\text{train}})}_{\text{posterior}} \triangleq \frac{\overbrace{p(\mathcal{D}^{\text{train}}|\theta_f)}^{\text{likelihood}}\ \overbrace{p(\Theta_f)}^{\text{prior}}}{\underbrace{p(\mathcal{D}^{\text{train}})}_{\text{evidence}}}. \tag{2.13}$$

Remark 2.5 (posterior predictive and types of uncertainty for supervised learning (SL))***:*** The posterior distribution over model parameters allows us to make predictions, using the *posterior predictive* distribution, a.k.a. *Bayesian model average*, given by:

$$p(Y = y|x; \mathcal{D}^{\text{train}}) = \int_{\mathbb{\Theta}_f} p(Y = y|x; \theta_f) p(\Theta_f = \theta_f|\mathcal{D}^{\text{train}}) d\theta_f, \tag{2.14}$$

which intuitively answers to the question: 'What is the posterior distribution of the model output $y \in \mathbb{Y}$ for an input x, after observing data D^{train}?'. The posterior predictive captures both sources of uncertainty: (i) the inherent and irreducible stochasticity of the data, i.e., *aleatoric uncertainty* a.k.a. <u>*risk*</u>; and (ii) the reducible (in the limit of infinite data) learner's <u>*ignorance*</u> about the correct model, a.k.a. *epistemic uncertainty* or *model uncertainty* or *ambiguity* (*Remark* 2.1; Knight, 1921).

In *Figure 2.4c*, we visualise the mean (in solid purple) and the 3-standard deviations interval of the posterior predictive distribution, assuming a multivariate diagonal normal prior distribution with zero mean and standard deviation one, i.e., $p(\Theta_r) = \mathcal{N}(\theta_r; \mathbf{0}, \mathbf{I})$. The choice of prior is intended to match the likelihood (i.e., *conjugate prior*, Bishop, 2006) so that the posterior distribution can be computed analytically. As expected, the mean tracks the observed data and the standard deviation is low close to the data and high in OOD.

Next, we present *point estimation* inference methods, which select a single parameter configuration θ_f and hence are, generally, more scalable than exact Bayesian inference.

Maximum likelihood estimation. A popular point estimation method is *maximum likelihood estimation* (MLE), i.e., to maximise the (log-)likelihood of the observed data:

$$\begin{aligned}
\theta_{\text{MLE}} &\triangleq \arg\max_{\theta_f \in \Theta_f} \log p(\mathcal{D}^{\text{train}} | \theta_f) && \text{(2.15a)} \\
&\overset{(2.12)}{=} \arg\max_{\theta_f \in \Theta_f} \log \left(\prod_{n=1}^{N} p(y^{(n)} | x^{(n)}; \theta_f) \right) && \text{(2.15b)} \\
&= \arg\max_{\theta_f \in \Theta_f} \sum_{n=1}^{N} \log p(y^{(n)} | x^{(n)}; \theta_f). && \text{(2.15c)}
\end{aligned}$$

For the reward modelling *Example* 2.2, there is a closed-form for the MLE solution to the *Eqn.* (2.15), visualised in *Figure 2.4b*. Although the MLE solution perfectly fits the training data, i.e., *in-sample*, it does not generalise since it is not tracking the target function f^* *out-of-sample*. We refer to this phenomenon as *overfitting* (Vapnik and Chervonenkis, 1971; Bishop, 2006) and, next, we discuss a *regularisation* technique to alleviate it.

Maximum a posteriori inference. To constraint, i.e., *regularise*, the impact of the likelihood and avoid overfitting, the *maximum a posteriori* (MAP) inference advocates for selecting the mode of the posterior distribution over model parameters (see *Eqn.* (2.13)):

$$\begin{aligned}
\theta_{\text{MAP}} &\triangleq \arg\max_{\theta_f \in \Theta_f} \log p(\theta_f | \mathcal{D}^{\text{train}}) && \text{(2.16a)} \\
&\overset{(2.13)}{=} \arg\max_{\theta_f \in \Theta_f} \log \left(\frac{p(\mathcal{D}^{\text{train}} | \theta_f) p(\Theta_f)}{p(\mathcal{D}^{\text{train}})} \right) && \text{(2.16b)} \\
&= \arg\max_{\theta_f \in \Theta_f} \left(\underbrace{\log p(\mathcal{D}^{\text{train}} | \theta_f)}_{\text{MLE}} + \underbrace{\log p(\Theta_f)}_{\substack{\text{prior/} \\ \text{regulariser}}} - \underbrace{\cancel{\log p(\mathcal{D}^{\text{train}})}}_{\partial / \partial \theta_f = 0} \right) && \text{(2.16c)} \\
&\overset{(2.15)}{=} \arg\max_{\theta_f \in \Theta_f} \left(\sum_{n=1}^{N} \log p(y^{(n)} | x^{(n)}; \theta_f) + \log p(\Theta_f) \right), && \text{(2.16d)}
\end{aligned}$$

where the evidence term $\log p(\mathcal{D}^{\text{train}})$ is cancelled in *Eqn.* (2.16c) since it is not a function of the optimising variable θ_f. The MAP objective ends up being in *Eqn.* (2.16d) similar to the MLE solution, with an extra prior/regularisation term.

For instance, in the reward modelling *Example* 2.2, we assume a multivariate diagonal normal prior distribution with zero mean and standard deviation one, i.e., $p(\Theta_r) = \mathcal{N}(\theta_r; \mathbf{0}, \mathbf{I})$. In *Figure 2.4b*, we note that the (regularised) MAP solution generalises better than the MLE solution, especially between the observed data points. Nonetheless, both solutions perform poorly outside the range of the observed data.

Remark 2.6 (point estimates and ignorance quantification)***:*** The MLE and MAP estimators, and more broadly, point estimation methods lack a mechanism for quantifying ignorance (MacKay, 1992; Bishop, 2006; Barber, 2012; Gal, 2016).

Example 2.2 (supervised learning (SL) for reward modelling with polynomial functions)***:*** Consider the SL problem with scalar input-output pairs, i.e., reward modelling from scalar continuous action to stochastic reward, illustrated in *Figure 2.4a*:

$$r^{(n)} = r^*(a^{(n)}) + \epsilon^{(n)}, \tag{2.17a}$$

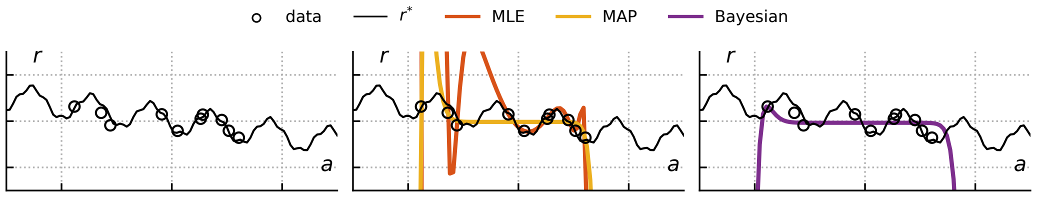
$$\text{where} \quad r^*(a) = \sin(\pi a) + 0.2\cos(4\pi a) - 0.3a \quad \text{and} \quad \epsilon^{(n)} \sim \mathcal{N}(0, 0.1). \tag{2.17b}$$

We choose 20-th degree polynomials as the class of functions to parametrise the mean of a normal distribution with mean θ_r, and fixed standard deviation:

$$p(R = r|a; \theta_r) = \mathcal{N}(r; r_{\text{poly-20}}(a; \theta_r), \sigma^2_{\text{fixed}}), \tag{2.18a}$$

$$\text{where} \quad r_{\text{poly-20}}(a; \theta_r) \triangleq \theta_{r,0} + \theta_{r,1}a + \theta_{r,2}a^2 + \cdots + \theta_{r,20}a^{20}. \tag{2.18b}$$

A risk-neutral, e.g., expected utility (EU), agent (see 2.1.1 and *Remark* 2.3), needs to only model the expected reward. An ignorance-neutral, e.g., model average, agent (see §2.1.2), can make a decision by maximising either the point estimates in *Figure 2.4b* or the mean of the posterior in *Figure 2.4c*. In contrast, an ignorance-sensitive agent cannot do without quantifying its ignorance about the reward function and, as of *Remark* 2.6, cannot rely on point estimators.



(a) Noisy data ***(b)*** Point estimates ***(c)*** Ignorance-aware model

Figure 2.4: Supervised learning (SL) with probabilistic inference for the coefficients of 20-th degree polynomial functions. ***(a)*** The input-output pairs, with additive i.i.d. normal noise. ***(b)*** The maximum likelihood estimation (MLE) and maximum a

posteriori (MAP) point estimates. ***(c)*** The mean and 3 standard deviations interval of the posterior predictive distribution, using exact Bayesian inference.

So far, we operate under the assumption that a class of parametric functions (see *Eqn.* (2.8)) is given to us and we reviewed methods to learning their parameters that best explained the data. Next, we focus on a flexible, high-capacity class of functions, called neural networks.

2.2.2 Deep learning (DL)

We refer to SL with deep *neural networks* (NNs, McCulloch and Pitts, 1943; Rosenblatt, 1958; Fukushima, 1980; Hochreiter and Schmidhuber, 1997; LeCun et al., 1998; Krizhevsky et al., 2012; He et al., 2016b) as deep learning (DL). A L-layers deep NN comprises of the composition of L interleaving linear and non-linear activation layers:

$$f_{\text{NN}}(x; \theta_f) = W_L \nu \left(\cdots \nu \left(W_2 \nu \left(W_1 x + b_1 \right) + b_2 \right) \cdots \right) + b_L, \tag{2.19}$$

where ν is a parameter-free non-linear function, e.g., the sigmoid function, and the "learnable" model parameters are the weights and biases, i.e., $\theta_f \triangleq [W_1, b_1, \cdots, W_L, b_L]$.

Gradient-based optimisation. In contrast to the simple linear models in *Example* 2.2, there are no closed-form solutions for the MLE and MAP estimates in *Eqns.* (2.15, 2.16) for a NN. Instead, we can use first-order gradient-based optimisation (Boyd et al., 2004), such as stochastic gradient descent (SGD), to efficiently optimise them:

$$\theta_f' \leftarrow \theta_f - \eta \nabla_{\theta_f} \mathcal{L}(\theta_f, \mathcal{D}) = \theta_f - \eta \frac{\partial \mathcal{L}}{\partial f} \frac{\partial f}{\partial \theta_f}, \tag{2.20}$$

where the *loss function* $\mathcal{L}(\theta_f, \mathcal{D})$ is the negative log-likelihood (resp. with a regulariser) for the MLE (resp. MAP) estimate,[†] and $\eta \in \mathbb{R}^+$ is the learning rate. For a non-trivial hypothesis space Θ_f, the loss function $\mathcal{L}$ is not convex w.r.t. the function parameters θ_f and therefore the iterative parameter updates in *Eqn.* (2.20) converge[‡] to a local minimum. However, it has been conjectured (Choromanska et al., 2015) and shown empirically that SGD converge to high quality local minima and therefore *Eqn.* (2.20) is a sensible and scalable learning algorithm. Next, we see how to efficiently calculate the $\partial f / \partial \theta_f$ term.

Back-propagation. Reverse-mode automatic differentiation techniques (Rall, 1981) and, in particular, the *back-propagation* algorithm (Rumelhart et al., 1986) can be used for efficiently calculating the gradient of the NN output w.r.t. its parameters, i.e.,

$$\frac{\partial f}{\partial \theta_f} \triangleq \left. \frac{\partial \hat{y}}{\partial \theta_f} \right|_{\hat{y} = f(x; \theta_f)}. \tag{2.21}$$

[†]Generalises to other functions too, as long as the partial derivative $\partial \mathcal{L} / \partial f$ exists and it's finite.
[‡]Convergence is guaranteed for some principled adaptive learning rates (Robbins and Monro, 1951).

Next, we review methods for ignorance quantification that scale to deep NNs.

2.2.3 Ignorance-aware deep neural networks

Exact Bayesian inference for the parameters of a non-trivial NNs is intractable. Approximate Bayesian inference algorithms have been used instead, such as the Laplace's method (MacKay, 1992; Tishby et al., 1989; Rue et al., 2009; Ritter et al., 2018), variational inference (Hinton and Van Camp, 1993; Graves, 2011; Blundell et al., 2015; Kingma et al., 2015; Gal and Ghahramani, 2016; Louizos and Welling, 2017; Farquhar et al., 2020) and expectation propagation (Minka, 2001; Hernández-Lobato and Adams, 2015).

Modern NNs have up to hundreds of billions of parameters, making the exact computation of the posterior predictive distribution for even a single input in *Eqn.* (2.14) practically impossible. In practice, we approximate the calculation of the integral with Monte Carlo (MC) sampling from the (approximate) posterior distribution over the model parameters:

$$p(Y = y|x; \mathcal{D}^{\text{train}}) \approx \frac{1}{K} \sum_{k=1}^{K} p(Y = y|x; \hat{\theta}_{f,k}), \quad \text{s.t.} \quad \left\{\hat{\theta}_{f,k}\right\}_{k=1}^{K} \sim p(\Theta_f|\mathcal{D}^{\text{train}}). \tag{2.22}$$

Certain methods aim to directly draw approximate samples from the posterior over model parameters $\hat{\theta}_{f,1}, \hat{\theta}_{f,2}, \ldots, \hat{\theta}_{f,K}$ (Chen et al., 2014), embracing the fact that an accurate representation of the posterior over model parameters $p(\Theta_f|\mathcal{D}^{\text{train}})$ is intractable both to obtain and to use, e.g., for computing the posterior predictive or other statistics.

Ensemble methods. One simple and scalable family of methods in DL for approximate sampling from the posterior are the (deep) ensembles (Lakshminarayanan et al., 2017; Osband et al., 2016). A deep ensemble comprises of K *components* (a.k.a. *members*), which are K NNs with separate parameters, trained either independently or jointly (Lakshminarayanan et al., 2017; Pearce et al., 2020). Independent training is a lot more scalable since it can be perfectly parallelised in hardware but it is less principled (Wenzel et al., 2020). It has been argued that the diversity of independently trained ensemble members is important for approximating the posterior predictive distribution (Lakshminarayanan et al., 2017; Osband et al., 2018; Fort et al., 2019; Wilson and Izmailov, 2020) and, to that end, various noise injections techniques *per member* have been used to achieve it.

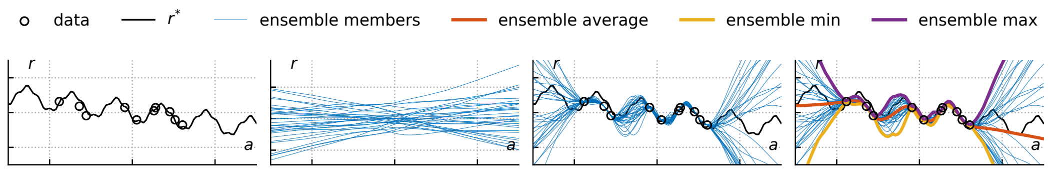
> ***Remark 2.7*** (general implementation strategy for deep ensembles)***:*** In this thesis, unless explicitly stated otherwise, we independently train deep ensembles members with MAP inference by: (i) randomising the parameters initialisation per member (Lakshminarayanan et al., 2017; Izmailov et al., 2021); (ii) using a different subset of data via bootstrapping for each member (Tibshirani, 1996; Osband et al., 2016); and (iii) adding structured noise (e.g., randomised priors, Osband et al., 2018).

Intuitively, we expect the ensemble members' predictions to "agree" on the in-distribution data. Moreover, due to the aforementioned noise injection mechanisms, the ensemble members are expected to generalise differently in OOD regimes and, therefore, their predictions to "disagree" there. *Figure 2.5* confirms this intuition in a simple regression problem.

Empirical validation. Deep ensembles are the default approach to ignorance quantification in DL not only due to their simplicity in their implementation but also due to their empirically validated superior performance, compared to the "more principled" approximate inference approaches (Ovadia et al., 2019). Despite being "less Bayesian", deep ensembles are shown to lead to better (i) ignorance quantification (Wilson and Izmailov, 2020); and (ii) predictive performance with orders of magnitude improved sample and hence computational efficiency (Ashukha et al., 2020).

> ***Remark 2.8*** (deep ensembles for ignorance quantification by default)***:*** For the remaining of this thesis, we are using deep ensembles as the main mechanism for ignorance quantification for NNs, with the training strategy outlined in *Remark* 2.7. The key contributions lie in to *how these ignorance estimates are used for sequential decision-making via planning* and not how they are obtained. We expect our contributions to be compatible with future improvements in ignorance quantification.

> ***Example 2.3*** (deep ensembles for ignorance-aware reward modelling)***:*** Consider the problem setting of *Example* 2.2, i.e., SL with the data in *Figure 2.5a* but instead of polynomial functions, we use 2-layer multi-layer perceptrons (MLPs) with 1024 hidden units as the function class to approximate f^* (see *Eqn.* (2.17)). We follow the training strategy in *Remark* 2.7 for an ensemble of $K = 32$ members with the AdamW (Kingma and Ba, 2014; Loshchilov and Hutter, 2017) optimiser with a linearly scheduled learning rate that decays from 4e-4 to 0 after 10,000 steps.



(a) Noisy data *(b)* At initialisation *(c)* After training *(d)* KEs

Figure 2.5: Ensemble of deep neural networks (NNs) for ignorance quantification. ***(a)*** The action→reward pairs, with additive i.i.d. normal noise. ***(b)*** The diversely initialised ensemble members. ***(c)*** The ensemble members' predictions at the end of training, where the members "agree" in-distribution, i.e., close to the data, and disagree in OOD inputs and hence forming useful ignorance estimates. ***(d)*** Knowledge equivalents (KEs) for ignorance-aware agents (see §2.1.2) using ensemble statistics. Note that an ignorance-seeking (resp. -averse) agent that maximises the ensemble max (resp. min) would pick an action far from (resp. close to) the observed data.

Next, we present a sequential decision-making setting, in which the agent has access to expert demonstrations and hence it can use SL methods to solve it.

2.2.4 Imitation learning (IL)

In many real-world settings, such as autonomous driving, expert demonstrations are available (Sun et al., 2020) or can be efficiently gathered (Behbahani et al., 2019). Imitation learning (IL, Widrow, 1964; Pomerleau, 1989; Atkeson and Schaal, 1997) is a framework for learning sequential decision-making policies when expert demonstrations are available:

$$\mathcal{D} \triangleq \left\{ \left(s^{(n)}, a^{(n)} \right) \right\}_{n=1}^{N}, \quad \text{s.t.} \quad a^{(n)} \sim \pi^{*}(\cdot | s^{(n)}). \tag{2.23}$$

The goal in IL is to match the expert policy that generated the high-quality demonstrations (Pomerleau, 1991; Heskes, 1998; Ng et al., 2000; Abbeel and Ng, 2004). The dominant approaches to IL are (i) behaviour cloning (BC, Widrow, 1964; Pomerleau, 1989); and (ii) inverse reinforcement learning (IRL, Kalman, 1964; Russell, 1998).

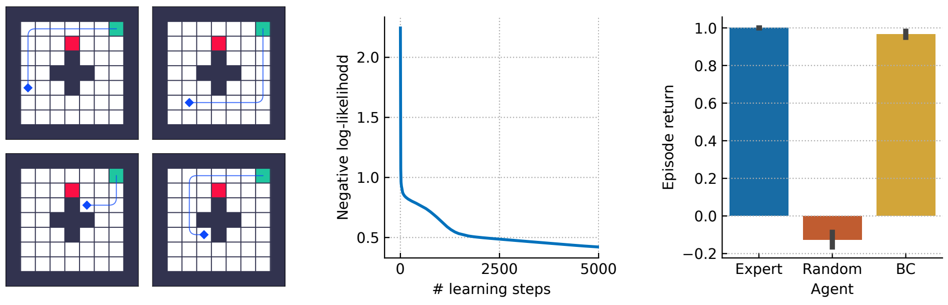
Behaviour cloning. The agent tries to identify, i.e., *behaviourally clone* (BC), the expert policy mapping[†] $\pi^{*} : \mathbb{S} \to \mathbb{A}$ from the expert demonstrations $\mathcal{D}$, e.g., using SL methods. Both point estimation methods, such as MLE and MAP inference (Widrow, 1964; Pomerleau, 1989; Heskes, 1998; Billard et al., 2008; Argall et al., 2009; Codevilla et al., 2018) and (approximate) Bayesian inference (Menda et al., 2019) have been used for BC with NNs.

Example 2.4 (behaviour cloning (BC) an expert gridworld policy)***:*** Consider a 7×7 gridworld environment and access to expert demonstrations, depicted in *Figure 2.6a*. The goal of the agent is to navigate to the (green) goal cell without falling into the (red) trap cell. When the goal (respectively, trap) cell is reached, the episode terminates, and the agent receives $+1$ (respectively -1) reward and the environment resets, randomly placing the agent in an empty cell.

The agent learns to act from the expert demonstrations via BC. It approximates the optimal policy with a 2-layers deep multi-layer perceptron (MLP, Rosenblatt, 1958; Ivakhnenko and Lapa, 1966) neural network, trained with SGD (see (2.20)) and on the negative log-likelihood (see *Eqn.* (2.15)), illustrated in *Figure 2.6b*.

Figure 2.6c shows that the BC agent effectively approximates the expert policy, performing on par with it and clearly overperforming a baseline random agent.

[†]In §2.3, we provide a formalisation of policies as mappings from a state to distribution over actions.



(a) Expert demonstrations ***(b)*** Loss minimisation ***(c)*** Performance

Figure 2.6: Learning to act from expert demonstrations in a gridworld via behaviour cloning (BC) with a multi-layer perceptron (MLP) neural network policy. ***(a)*** Example trajectories of the expert agent ◆, which navigates to the goal 🟩 while avoiding the trap 🟥. ***(b)*** The loss function (see *Eqn.* (2.15)) decreases as more gradient update (i.e., learning) steps (see 2.20) are performed. ***(c)*** The performance, quantified as the episode return (see *Eqn.* (2.30)), of the BC agent, compared to the expert agent and an agent that selects actions uniformly randomly.

Inverse reinforcement learning. The agent tries to identify the *reward function* $\mathrm{r}^* : \mathbb{S}\times\mathbb{A} \rightarrow \Delta(\mathbb{R})$ from the expert demonstrations $\mathcal{D}$, using function approximation techniques. Ng et al. (1999) showed that there are infinitely many reward functions, under which the demonstrator policy is optimal, and since, numerous constraints have been proposed to select among the available options (Abbeel and Ng, 2004; Ratliff et al., 2006; Ramachandran and Amir, 2007; Neu and Szepesvári, 2009; Ziebart et al., 2008; Ni et al., 2021).

While the agent's main objective is *not* to learn an accurate model of the "true" reward function but to find a policy for taking good actions in the environment, there may be good reasons to do it as a means to an end. In particular, learning a reward model should be preferred over learning the direct mapping of states to actions via BC when (i) the number of available expert demonstrations is small to accurately (globally) identify the optimal policy; and (ii) the reward function is easier to identify from the demonstrations (Ng et al., 2000; Finn et al., 2016). In §4, we also show how to use IRL for making effective use of *sub-optimal* demonstrations in multi-task settings, which would not be possible with BC.

Next, we review the sequential decision-making problem setting and solution methods that make use of the reward function for learning policies.

2.3 Reinforcement learning (RL)

In this section, we review data-driven sequential decision-making under uncertainty, extending the formulations from §2.1 and making use of the learning algorithms from §2.2.

2.3.1 Agent-environment interface & definitions

We study a decision-maker (i.e., *agent*), acting in a stochastic environment by sequentially choosing actions over a sequence of discrete time steps, indexed by t, to maximise an objective (i.e., *utility*). The agent exchanges symbols with the environment. At any discrete time step $t \geqslant 0$, the agent receives the (fully observable) environment *state* $s_t \in \mathbb{S}$, takes an action $a_t \in \mathbb{A}$ and this interaction repeats, giving rise to a state-action sequence:

$$S_0 A_0 S_1 A_1 S_2 A_2 \cdots S_\tau A_\tau \cdots \triangleq (S_t, A_t)_{t \geqslant 0}. \tag{2.24}$$

We consider[†] states that satisfy the Markov property, as in *Example* 2.5:

$$p(S_{t+1} | (S_\tau, A_\tau)_{\tau \leqslant t}) = p(S_{t+1} | S_t, A_t), \forall \tau. \tag{2.25}$$

We can model the agent-environment interaction as a controlled Markov process (CMP):

$$\mathcal{C} \triangleq \langle \mathbb{S}, \mathbb{A}, p, \rho_0 \rangle \,, \tag{2.26}$$

where $\mathbb{S}$ and $\mathbb{A}$ represent the state and action spaces, respectively, ρ_0 the initial state distribution and $p : \mathbb{S} \times \mathbb{A} \to \Delta(\mathbb{S})$ is the transition dynamics, i.e., $S_0 \sim \rho_0$ and $S_{t+1} \sim p(\cdot | s_t, a_t)$.

Example 2.5 (controlled Markov process (CMP) 1D gridworld)***:*** For instance, consider a navigation agent in an 1D gridworld environment, as depicted in *Figure 2.7*.

The state of the environment is the position of the agent in the gridworld, i.e., $s \in \{0, 1, 2, 3\} = \mathbb{S}$ and the available actions are $a \in \{\texttt{LEFT}, \texttt{RIGHT}\} = \mathbb{A}$. The `RIGHT` action deterministically moves the agent one cell to the right, while the `LEFT` action with probability 0.9 moves the agent one cell to the left and leave its position unchanged otherwise. Therefore, the environment states satisfy the Markov property *Eqn.* (2.25) and hence the agent-environment interaction can be modelled as a CMP.

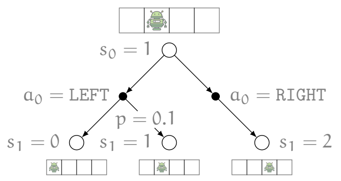


Figure 2.7: An agent navigates in an 1D gridworld environment, modelled as a controlled Markov process (CMP). The white nodes represent the environment states and the black nodes the agent actions.

[†]We focus on the simpler fully observable setting to most clearly communicate the contributions of this work. In later chapters, we show empirically that the results extend to the k-order MDPs (Puterman, 2014) and *partially observed Markov decision processes* (POMDPs, Kaelbling et al., 1998) models.

Task. In this work, we study goal-directed agents, which take actions in an environment to solve tasks. A *task* is formulated as a Markov decision process (MDP, Bellman, 1957b):

$$\mathcal{M} \triangleq \langle \mathbb{S}, \mathbb{A}, p, \rho_0, r \rangle \overset{(2.26)}{=} \langle \mathcal{C}, r \rangle \triangleq \mathcal{C}^r, \tag{2.27}$$

extending the CMP model with the (stochastic) reward function $r : \mathbb{S} \times \mathbb{A} \to \Delta(\mathbb{R})$.

> ***Remark 2.9*** (multi-task setting with shared dynamics)***:*** The CMP-based formulation of a task, as in *Eqn.* (2.27), is useful in multi-task settings, in which an agent has to solve multiple MDPs with the same state, action spaces and state-transition dynamics but different reward functions (§4; Dayan, 1993; Filos et al., 2021).

Extending the decision-making setup of §2.1 to multiple steps, at any discrete time step $t \geqslant 0$, the agent is in state $s_t \in \mathbb{S}$, takes an action $a_t \in \mathbb{A}$, according to a policy $\pi : \mathbb{S} \to \Delta(\mathbb{A})$, then receives reward $R_{t+1} \sim r(\cdot|s_t, a_t) \in \mathbb{R}$ and transitions to the state $S_{t+1} \sim p(\cdot|s_t, a_t)$. For brevity, the "true" environment (world) model is denoted by $m^* \triangleq (p, r)$ and we write:

$$S_{t+1}, R_{t+1} \sim m^*(\cdot, \cdot|s_t, a_t). \tag{2.28}$$

When it is clear from the context, we omit the subscript time index notation and use instead the "prime" notation for successive time steps, i.e., $S', R' \sim m^*(\cdot, \cdot|s, a)$.

Objective. The agent's goal is to find a policy π^* that maximises the *value*[†] of states:

$$\pi^* \in \arg\max_{\pi} \underbrace{\mathbb{E}[v^{\pi}(S)|S \sim \rho_0]}_{\triangleq J(\pi)} \tag{2.29a}$$

$$\text{s.t.} \quad v^{\pi}(s) \triangleq \mathbb{E}[\sum_{t \geqslant 0} \gamma^t R_{t+1} | S_0 = s, A_t \sim \pi(\cdot|S_t), (S_{t+1}, R_{t+1}) \sim m^*(\cdot, \cdot|S_t, A_t)], \tag{2.29b}$$

$$= \mathbb{E}[\sum_{t \geqslant 0} \gamma^t R_{t+1} | S_0 = s; \pi, m^*] \tag{2.29c}$$

$$= \mathbb{E}_{\pi, m^*}[\sum_{t \geqslant 0} \gamma^t R_{t+1} | S_0 = s], \tag{2.29d}$$

where $\gamma \in [0, 1)$ is the discount factor and $\mathbb{E}_{\pi, m^*}[\cdot]$ denotes the expectation over the trajectories induced by running policy π in the environment m^*, starting from state s. We refer to the argument of the expectation in *Eqn.* (2.29d) as the *return* random variable:

$$G^{\pi}_s \triangleq \sum_{t \geqslant 0} \gamma^t R_{t+1} | S_0 = s; \pi, m^* \Rightarrow v^{\pi}(s) \overset{(2.29d)}{=} \mathbb{E}_{\pi, m^*}[G^{\pi}_s | S_0 = s]. \tag{2.30}$$

[†]We focus on the standard, infinite horizon, geometric discounting return setting. The results should be extendable to other settings, e.g., finite horizon undiscounted return. This results in an expected utility (EU) agent, i.e., which maximises a risk-neutral certainty equivalent (CE), following *Remark* 2.3.

Discounting. The discount factor determines the present value of future rewards: immediate rewards are valued more than future rewards. A reward received k time steps in the future is worth $\gamma^{k-1}(< 1)$ times what it would be worth if it was received immediately.

Example 2.6 (Markov decision process (MDP) 1D gridworld)***:*** Consider the sequential decision-making problem setting in *Figure 2.8*.

The agent navigates the modified 1D gridworld environment of *Figure 2.7*, in which it receives a positive reward $+1$ when it enters the most right cell (goal), i.e., $s = 3$, and reward 0 otherwise. The interaction (episode) terminates upon reaching the goal state. Therefore, the dynamics of the environment are unchanged from the CMP in *Figure 2.7*, the reward function satisfies *Eqn.* (2.28) and hence the agent-environment interaction can be modelled as a MDP. For any (non-zero) discount factor, i.e., $\gamma \in (0, 1)$, it can be shown that the optimal policy is:

$$\pi^*(A = \texttt{RIGHT}|s) = 1, \forall s \in \mathbb{S},$$

and its (optimal) value, i.e., $v^{\pi^*} \triangleq v^*$, is:

$$\begin{aligned} v^*(s = 0) &= \gamma^2 \\ v^*(s = 1) &= \gamma \\ v^*(s = 2) &= 1 \\ v^*(s = 3) &= 0. \end{aligned}$$

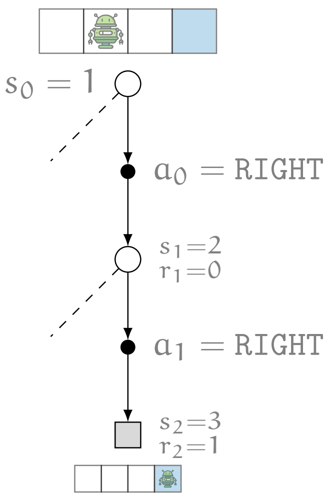


Figure 2.8: An agent navigates in an 1D gridworld environment to reach a rewarding goal, modelled as a Markov decision process (MDP). The white nodes represent the environment states, the black nodes the agent actions and the grey square nodes the terminal environment states, i.e., goal.

Intuitively, the value function $v^\pi(s)$ answers to the question: 'How good is it to be in state s when following policy π?'. It allows us to compare states, e.g., state s_A should be preferred over s_B under policy π if and only if (iff) $v^\pi(s_A) > v^\pi(s_B)$, but it does not offer us a way to search over actions and, by extension, improved policies to solve *Eqn.* (2.29a).

Action values. Consider a policy that first selects action a and thereafter follows policy π, which we call *one-step lookahead* policy w.r.t. π. Its value function, for all actions $a \in \mathbb{A}$, is termed the *action-value function* (a.k.a. Q-function), and it is given by:

$$q^\pi(s, a) \triangleq \mathbb{E}_{\pi,m^*}[\sum_{t \geqslant 0} \gamma^t R_{t+1} | S_0 = s, A_0 = a], \forall s \in \mathbb{S}, a \in \mathbb{A} \tag{2.31a}$$

$$= \mathbb{E}_{\pi,m^*}[R_1 + \gamma \sum_{t \geqslant 1} \gamma^t R_{t+1} | S_0 = s, A_0 = a] \tag{2.31b}$$

$$= \mathbb{E}_{\pi,m^*}[R_1 + \gamma v^\pi(S_1) | S_0 = s, A_0 = a], \tag{2.31c}$$

where *Eqn.* (2.31c) connects the action value q^π to the (state) value v^π, and vice versa:

$$v^\pi(s) \overset{(2.29d)}{\triangleq} \mathbb{E}_{\pi,m^*}[\sum_{t \geqslant 0} \gamma^t R_{t+1} | S_0 = s] \tag{2.32a}$$

$$= \mathbb{E}_{\pi,m^*}[R_1 + \gamma v^\pi(S_1) | S_0 = s] \tag{2.32b}$$

$$\overset{(2.31)}{=} \mathbb{E}_{A \sim \pi(\cdot|s)}[q^\pi(s, A)]. \tag{2.32c}$$

Intuitively, the action-value function $q^\pi(s, a)$ answers to the question: 'How good is it to take action a in state s and then follow policy π?', enabling us to *locally* search for the value maximising actions and, by extension, policies, towards solving *Eqn.* (2.29a). We can show (Bellman, 1957a) that the *greedy* w.r.t. the action-value function policy π_{greedy} is guaranteed to be a strict improvement over the policy π (unless already optimal) i.e.,

$$\pi_{\text{greedy}}(A = a|s) \triangleq \begin{cases} 1, \text{if } a = \arg\max_{b \in \mathbb{A}} q^\pi(s, b) \\ 0, \text{otherwise} \end{cases},^{\dagger} \tag{2.33a}$$

$$\text{then} \quad v^\pi(s) \leqslant v^{\pi_{\text{greedy}}}(s), \forall s \in \mathbb{S}. \tag{2.33b}$$

We can generalised the concept of action-value functions to *multi-step* lookahead policies:

$$q^\pi(s, \mathbf{a}^{0:k-1}) \triangleq \mathbb{E}_{\pi,m^*}[\sum_{t \geqslant 0} \gamma^t R_{t+1} | S_0 = s, A_{0:k-1} = \mathbf{a}^{0:k-1}], \tag{2.34}$$

where $\mathbf{a}^{0:k-1} \triangleq (a^0, a^1, \cdots, a^{k-1})$ is a k-step (open-loop) action sequence (i.e., *plan*). Note that for $k = 1$ in *Eqn.* (2.34) we obtain the action-value function of *Eqn.* (2.31).

The reinforcement learning *problem*. The environment model m^* is unknown to the agent and therefore it cannot directly compute π^* from *Eqn.* (2.29). Instead, the agent has access to samples of agent-environment interactions, generated by *some* agent(s), including itself, which it leverages for finding the optimal policy. This setting, i.e., sequential decision-making in an unknown environment, is termed the reinforcement learning (RL, Sutton and Barto, 2018) problem and it is the main focus of this thesis.

Data. The agent learns from state-reward-action sequences, a.k.a. *experience* or *rollouts*:

$$\mathcal{B} \triangleq \left\{ \left(s_t^{(n)}, r_t^{(n)}, a_t^{(n)} \right)_{t \geqslant 0} \right\}_{n=1}^{N}, \quad \text{s.t.} \quad \begin{cases} s_{t+1}^{(n)}, r_{t+1}^{(n)} \sim m^*(\cdot, \cdot | s_t^{(n)}, a_t^{(n)}) \\ a_t^{(n)} \sim \pi_\beta(\cdot | s_t^{(n)}) \end{cases}, \tag{2.35}$$

[†]Without loss of generality, we assume that the action-value per-state has a unique maximiser, e.g., we can enforce it by deterministically breaking ties (Sutton and Barto, 2018).

where, crucially, in contrast to the IL setting in *Eqn.* (2.23), the actions $\{a_t^{(n)}\}_{t,n}$ can be sampled from *any* (possibly sub-optimal) *behavioural policy* π_β. We refer to rollouts, and as a consequence to algorithms, as *on-policy* when the behavioural policy is the agent's policy, i.e., $\pi_\beta = \pi_{\text{agent}}$, and as *off-policy*, otherwise (Sutton and Barto, 2018).

Next, we present data-driven solution methods for the RL problem for learning policies, values or/and world models, making use of the supervised learning (SL) algorithms from §2.2.

2.3.2 Markov decision process (MDP) solvers

For many real-world problems, such as balloons navigation (Bellemare et al., 2020) and the natural sciences (Degrave et al., 2022), expert demonstrations (see *Eqn.* (2.23)) may be expensive or impossible to gather. Instead, the specification of a reward function can be (more) tractable. Reinforcement learning (RL, Sutton and Barto, 2018) is a framework for learning sequential decision-making policies when a reward function can be specified (or efficiently learned) and the problem of interest can be formulated as a MDP.

> ***Remark 2.10*** (intuition behind most reinforcement learning (RL) algorithms)***:*** A RL algorithm makes use of the reward information in rollouts (see *Eqn.* (2.35)) to decide which actions to do more in the future (*positively reinforced*) and which ones to do less (*negatively reinforced*). In particular, it reinforces actions that maximise the expected future (discounted) cumulative rewards, i.e., the agent's value (see 2.29).

Targets. In §2.2, we presented how to learn the parameters of an unknown function with SL, provided (possibly) noisy input-output sampled pairs from it. SL cannot be used out of the box to solve the RL problem since we do not have access to samples from, e.g., the optimal (action-)value function or the optimal policy. Instead, we need to form *estimates* of these quantities from the provided or generated rollouts. We refer to these estimates as value and policy *targets* and use the hat notation, i.e., $v^\pi \approx \hat{v}$ and $\pi' \approx \hat{\pi}$.

Function approximation for reinforcement learning. For most practical settings, the state and action spaces are continuous or so large that it is not possible to represent learned policies or values without parametric (i.e., fixed size) models. To that end, in this thesis, we use parametric function approximators (see §2.2). In particular, we denote with the explicit parameter subscript or semicolon notation the parametric approximations, e.g., v_{θ_v} is a deep neural network (NN) with parameters θ_v trained to approximate the (empirical) mapping $s \mapsto \hat{v}$ and, as a consequence, $v_{\theta_v} \approx \hat{v} \approx v^\pi$. We refer to this setting as *deep RL* to highlight the use of deep NNs with RL agents (Mnih et al., 2013).

Value learning

Let parametric (action-)value function approximator $\nu_{\theta_\nu} : \mathbb{S} \to \mathbb{R}$ (resp. $q_{\theta_q} : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$). As a reminder, see *Eqns.* (2.30, 2.29d), the value of a policy π is the expected discounted cumulative rewards, a.k.a. return. In the most interesting settings, the exact computation of this expectation is intractable without access to the environment model m^* or for large state and/or action spaces. To that end, different estimators are used instead, to obtain value targets for states from $\mathcal{B}$, i.e.,$(s_t^{(n)}, \hat{\nu}_t^{(n)})_{t,n} \sim \texttt{value\_estimator}(\mathcal{B})$. Hence the value function parameters θ_ν are trained to fit the (estimated) value targets, i.e.,

$$\mathcal{L}_\nu(\theta_\nu) \triangleq \sum_{t,n} \|\nu_{\theta_\nu}(s_t^{(n)}) - \hat{\nu}_t^{(n)}\|, \tag{2.36}$$

which can be minimised via gradient-based optimisation, in the case of deep RL:

$$\theta'_\nu \leftarrow \theta_\nu - \eta \nabla_{\theta_\nu} \mathcal{L}_\nu(\theta_\nu), \tag{2.37}$$

where $\eta \in \mathbb{R}^+$ is the learning rate (a.k.a. *step size*).

Next, we present common in the literature value estimation methods.

Monte Carlo value estimation. A simple method for estimating expected values is Monte Carlo (MC) integration. In particular, let rollouts $\mathcal{B}$, generating by π and m^*, then:

$$\hat{\nu}_{\text{MC}}(s) \triangleq \sum_{n,t} \frac{\mathbb{1}[s_t^{(n)} = s] g_t^{(n)}}{\sum_{n,t} \mathbb{1}[s_t^{(n)} = s]} \approx \nu^\pi(s), \quad \text{s.t.} \quad g_t^{(n)} \overset{(2.30)}{=} \sum_{\tau \geqslant t} \gamma^{\tau - t} r_t^{(n)}, \tag{2.38}$$

where $\mathbb{1}[x = y]$ is the indicator function, i.e., equal to 1 when $x = y$ and 0 otherwise. The MC value estimator can be used for estimating the action value $q^\pi(s, a)$, too, by replacing in *Eqn.* (2.38) the indicator function $\cancel{\mathbb{1}[s_t^{(n)} = s]}$ with $\mathbb{1}[s_t^{(n)} = s, a_t^{(n)} = a]$:

$$\hat{q}_{\text{MC}}(s, a) \triangleq \sum_{n,t} \frac{\mathbb{1}[s_t^{(n)} = s, a_t^{(n)} = a] g_t^{(n)}}{\sum_{n,t} \mathbb{1}[s_t^{(n)} = s, a_t^{(n)} = a]} \approx q^\pi(s, a). \tag{2.39}$$

Despite its simplicity, MC value estimation has some major weaknesses (Sutton and Barto, 2018): it is (i) a high variance estimator; (ii) sample inefficient since all the samples for which the indicator function is 0 do not contribute, either directly or indirectly, to the value estimation; and (iii) incompatible with the infinite horizon discounted setting since it is not possible to compute the return $g_t^{(n)}$ in *Eqn.* (2.38) when $\tau \to \infty$ and therefore a truncated return is used in practice, i.e., $\tau < T$ for some finite T, introducing bias.[†]

[†]However, for $\tau - t >> 1/(1-\gamma)$, this bias can be trivial but it is often not in practice (Sutton, 1988).

Temporal difference learning. A class of value learning algorithms that trade-off variance and sample inefficiency with bias is called temporal difference (TD) learning methods (Sutton, 1988). TD learning exploit the recursive nature of the value function:

$$\begin{aligned}
v^{\pi}(s) &\overset{(2.29d)}{\triangleq} \mathbb{E}_{\pi,m^*}[\sum_{t\geqslant 0} \gamma^t R_{t+1} | S_0 = s] && (2.40a)\\
&= \mathbb{E}_{\pi,m^*}[R_1 + \gamma \underbrace{\sum_{t\geqslant 1} \gamma^t R_{t+1}}_{v^{\pi}(S_1)} | S_0 = s] && (2.40b)\\
&= \mathbb{E}_{\pi,m^*}[R_1 + \gamma v^{\pi}(S_1) | S_0 = s] && (2.40c)\\
q^{\pi}(s,a) &\overset{(2.31c)}{\triangleq} \mathbb{E}_{\pi,m^*}[R_1 + \gamma q^{\pi}(S_1, A_1) | S_0 = s, A_0 = a], && (2.40d)
\end{aligned}$$

and employ the stochastic approximation method (Robbins and Monro, 1951) to estimate the expectations in *Eqn.* (2.40) with a single sampled transition (s, a, r', s', a'):[†]

$$\hat{v}_{\mathrm{TD}(0)}(s) \triangleq r' + \gamma \hat{v}_{\mathrm{TD}(0)}(s') \quad (2.41a)$$

$$\hat{q}_{\mathrm{TD}(0)}(s,a) \triangleq r' + \gamma \hat{q}_{\mathrm{TD}(0)}(s', a'). \quad (2.41b)$$

We note that the estimate of the value at the state s depends on the value of the subsequent state s', a method that we refer to as *bootstrapping* (Sutton and Barto, 2018). Due to the bootstrapping, the TD estimators are biased. We can design TD estimators with lower bias, in the expense of variance, by noting that the value functions in *Eqn.* (2.40) can be re-written as a sum of k reward terms and a γ^k-discounted value (bootstrap) term:

$$v^{\pi}(s) \overset{(2.29d)}{\triangleq} \mathbb{E}_{\pi,m^*}[\sum_{t\geqslant 0}^{k} (\gamma^t R_{t+1}) + \gamma^k v^{\pi}(S_k) | S_0 = s]. \quad (2.42)$$

If we apply the stochastic approximation method of Robbins and Monro (1951) on *Eqn.* (2.42), we obtain the lower bias (higher variance) k-step TD estimator, given by:

$$\hat{v}_{k\text{-step TD}}(s) \triangleq \sum_{t\geqslant 0}^{k} (\gamma^t r_{t+1}) + \gamma^k \hat{v}_{k\text{-step TD}}(s_k). \quad (2.43)$$

Note that as $k \to \infty$, we recover the (single sample) MC value estimator. Therefore, we can treat k as a hyperparameter for trading off bias and variance: for $k = 1$ the bias is the highest and for $k \to \infty$ the variance is the highest. Sutton (1988) showed that convex combinations of TD learning estimators with different values of k are valid value estimators and the most commonly used one is TD(λ), i.e., a geometric weighted mixture

[†]In *Eqn.* (2.41b), we present the SARSA (Sutton, 1988) algorithm and therefore need the sampled action a'. There are others algorithms, e.g., expected SARSA (Van Seijen et al., 2009; Sutton and Barto, 2018), for which we only need (s, a, r', s') sampled transitions for forming TD(0) action-value targets.

of different k-step TD learning targets with hyperparameter $\lambda \in [0, 1]$.

Q-learning. Similar to TD learning algorithms for estimating the value of a policy, *Q-learning* (Watkins, 1989) can be used for computing the optimal action-value function q^* (the value function of the optimal policy π^*) from a sampled transition (s, a, r, s'):

$$\hat{q}_{\mathrm{QL}}(s, a) \triangleq r + \gamma \max_{a' \in \mathbb{A}} \hat{q}_{\mathrm{QL}}(s', a'). \tag{2.44}$$

Eqn. (2.29d) suggest that the reason for computing the value function of a policy is to help us find an improved policy. Next, we present MDP solvers that act on this observation.

Policy learning

Let parametric policy $\pi_{\theta_\pi} : \mathbb{S} \to \Delta(\mathbb{A})$, we review algorithms that update the policy parameters θ_π in order to maximise the *regularised* policy optimisation objective, given by:

$$J(\pi) = \mathbb{E}_{S \sim \rho_0}[v^\pi(S)] + \Omega(\pi), \tag{2.45}$$

a modification to the original RL objective in *Eqn.* (2.29) that includes the regulariser $\Omega : \Pi \to \mathbb{R}$, e.g., action entropy (Williams and Peng, 1991) for discouraging the collapse to a deterministic policy and hence hurt exploration or a trust region (Schulman et al., 2015a; Abdolmaleki et al., 2018) for avoiding overoptimisation. Sutton et al. (1999) showed that the *policy gradient* of the objective in *Eqn.* (2.45) w.r.t. θ_π is given by:

$$\left.\frac{\partial J}{\partial \theta_\pi}\right|_{S_0 = s} \triangleq \delta_{\mathrm{PG}}(s) = \mathbb{E}_{\pi_{\theta_\pi}}[(q^{\pi_{\theta_\pi}}(s, a) - b(s))\nabla_{\theta_\pi} \log \pi_{\theta_\pi}(a|s)] + \nabla_{\theta_\pi}\Omega(\pi_{\theta_\pi}), \tag{2.46}$$

where $b : \mathbb{S} \to \mathbb{R}$ is any (state-only) *baseline* function. The policy gradient[†] in *Eqn.* (2.46) can be used to improve the policy via gradient-based optimisation using rollouts $\mathcal{B}$:

$$\theta'_\pi \leftarrow \theta_\pi + \eta \sum_{n,t} \delta_{\mathrm{PG}}(s_t^{(n)}), \tag{2.47}$$

where $\eta \in \mathbb{R}^+$ is the learning rate (a.k.a. *step size*). There is adaptive step size scheme, for which $J(\theta'_\pi) \geqslant J(\theta_\pi)$ (Boyd et al., 2004). The computation of the policy gradient in *Eqn.* (2.46) requires an estimate of the action-value. Different estimators for the action-value function give rise to different policy gradient methods and we review a few next.

[†]Vieillard et al. (2020) refers to policy gradient methods as *indirect* policy optimisation methods since they do not directly fit the policy parameters towards an improved policy target, instead they follow the gradient that indirectly leads to an improvement. However, Ghosh et al. (2020) showed that there is an equivalent formulation of policy gradient methods as implicit direct methods. Therefore, in this thesis, we do not make any distinction between direct and indirect methods.

REINFORCE. The first policy gradient algorithm is Williams's (1992) REINFORCE, given by (i) a zero baseline function; and (ii) single sample MC action-value estimation:

$$\delta_{\text{REINFORCE}}(s_t^{(n)}) = g_t^{(n)} \nabla_{\theta_\pi} \log \pi_{\theta_\pi}(a_t^{(n)} | s_t^{(n)}). \tag{2.48}$$

Non-trivial baseline functions, e.g., a learned state-value function as well as more advanced control variates (Foerster et al., 2018; Weber et al., 2019), can be used to reduce the variance of the REINFORCE gradient estimator (Weaver and Tao, 2001). Williams and Peng (1991); Mnih et al. (2016) also used an action entropy regulariser, i.e., $\Omega(\pi) = \mathbb{E}_\pi[\log \pi]$.

Actor-critic. A learned action-value function (approximator), i.e., $q_{\theta_q} \approx q^{\pi_{\theta_\pi}}$, a.k.a. *critic*, in this context, can be used along with the learned policy π_{θ_π}, a.k.a. *actor*, in this context, to estimate the policy gradient in *Eqn.* (2.46). Sutton et al. (1999); Sutton and Barto (2018) termed this class of methods as *actor-critic* methods, which have been some of the most successful deep RL algorithms (Mnih et al., 2016; Schulman et al., 2015a, 2017; Espeholt et al., 2018; Abdolmaleki et al., 2018; Haarnoja et al., 2018; Hessel et al., 2021). In practice, most of these algorithms estimate and use the *advantage*, given by:

$$A^\pi(s, a) \triangleq q^\pi(s, a) - v^\pi(s) \overset{(2.32c)}{=} q^\pi(s, a) - \mathbb{E}_{A \sim \pi(\cdot|s)}[q^\pi(s, A)], \tag{2.49}$$

in place of the action-value minus the baseline term in *Eqn.* (2.46), i.e.,

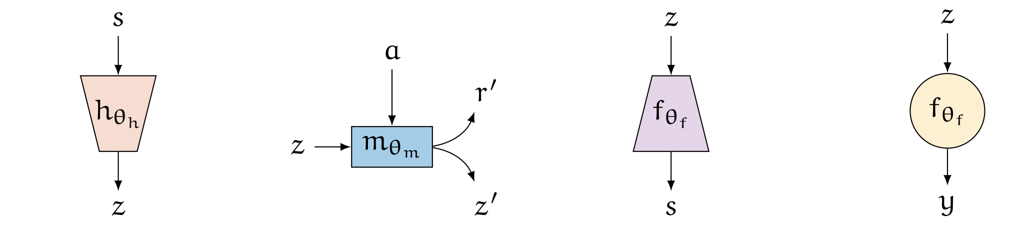
$$\delta_{\text{AC}}(s_t^{(n)}) = \hat{A}(s_t^{(n)}, a_t^{(n)}) \nabla_{\theta_\pi} \log \pi_{\theta_\pi}(a_t^{(n)} | s_t^{(n)}), \tag{2.50}$$

where the advantage estimate $\hat{A}$ is usually derived from a parametric (action-)value function approximator, e.g., using generalised advantage estimation (GAE, Schulman et al., 2015b; Kimura et al., 1998; Wawrzyński, 2009) — TD(λ) estimation for advantages.

So far the MDP solvers we have reviewed leverage the (real) rollouts $\mathcal{B}$ to learn directly value functions or policies or both. There are methods that also use model-generated rollouts for solving the MDP. Next we present algorithms for learning world models.

Model learning

Let learned (a.k.a. *world*) model $m_{\theta_m} : \mathbb{S} \times \mathbb{A} \to (\Delta(\mathbb{S}), \Delta(\mathbb{R}))$, with learnable parameters θ_m using rollouts $\mathcal{B}$ to approximate the "true" environment model m^* from *Eqn.* (2.28). Learned models can be useful to RL agents in various ways, such as: (i) action selection via planning (Richalet et al., 1978; Hafner et al., 2019b); (ii) representation learning (Schmidhuber, 1990; Jaderberg et al., 2016; Lee et al., 2019a; Guez et al., 2020; Hessel et al., 2021); (iii) planning for policy optimisation or value learning (Werbos, 1987; Sutton, 1991; Hafner et al., 2019b; Byravan et al., 2020); or (iv) a combination of all of them (Schrittwieser et al., 2020). In §5, we propose a novel use case for learned world



(a) Representation ***(b)*** Dynamics ***(c)*** Reconstruction ***(d)*** Partial

Figure 2.9: Latent world model sub-neural networks (NNs). ***(a)*** The representation network that embeds the observed environment state to a latent state (a.k.a. embedding). ***(b)*** The action-conditioned latent dynamics network. ***(c)*** The reconstruction network that decodes a latent state to an environment state. ***(d)*** The partial model's prediction network that predicts a partial view of the environment state, such as an estimate of the state value function or the optimal policy, from the latent state.

models, i.e., an ignorance quantification method for value functions, which relies on a single point estimate of a world model and a value function (Filos et al., 2022).

Reconstruction models. A popular, in the literature, class of model learning algorithms, rely on generative modelling (a.k.a. *reconstruction*) of observed states and rewards (Richalet et al., 1978; Sutton, 1991; Kumar and Varaiya, 2015; Sutton and Barto, 2018), originally rooted in system identification for optimal control (Bertsekas, 2012; Zhou et al., 1996). Given rollouts $(s, a, r', s') \sim \mathcal{B}$, the world model parameters θ_m are trained with SL algorithms, e.g., MLE or MAP, to approximate the empirical mapping $(s, a) \mapsto (s', r')$.

Latent variable models, e.g., action-conditioned hidden Markov models (Watter et al., 2015; Buesing et al., 2018; Hafner et al., 2019b), have been used to scale reconstruction-based methods to high-dimensional environments, e.g., with pixel observations. In particular, the underlying modelling assumption is that the state space $\mathbb{S}$ is compressible to a latent state space $\mathbb{Z}$, which also satisfies the Markov property in *Eqn.* (2.25). Using NNs, a (latent) world model is comprised of three sub-networks, illustrated in *Figure 2.9*: (i) the *representation* network that compresses environment states down to latent states (a.k.a. *embeddings*), i.e., $h_{\theta_h} : \mathbb{S} \to \mathbb{Z}$; (ii) the *reconstruction* network, which maps the latent states to environment states, i.e., $f_{\theta_f} : \mathbb{Z} \to \mathbb{S}$; and (iii) the latent dynamics network, i.e., $m_{\theta_m} : \mathbb{Z} \times \mathbb{A} \to (\Delta(\mathbb{Z}), \Delta(\mathbb{R}))$ or more explicitly $p_{\theta_m} : \mathbb{Z} \times \mathbb{A} \to \Delta(\mathbb{Z})$ and $r_{\theta_m} : \mathbb{Z} \times \mathbb{A} \to \Delta(\mathbb{R})$, where $(p_{\theta_m}, r_{\theta_m}) = m_{\theta_m}$. The h_{θ_h} and f_{θ_f} networks can also be stochastic, e.g., trained with variational inference (Watter et al., 2015; Hafner et al., 2019b).

To train the sub-networks we minimise the following loss w.r.t. the model parameters:

$$\mathcal{L}_{\text{rec-WM}}(\theta_h, \theta_m, \theta_f) \triangleq \mathcal{L}_{\text{s-rec}}(\theta_h, \theta_f) + \mathcal{L}_{\text{z-dynamics}}(\theta_m) + \mathcal{L}_{\text{r-dynamics}}(\theta_h, \theta_m), \quad (2.51)$$

where the constituent losses are given by:

$$\mathcal{L}_{\text{s-rec}}(\theta_h, \theta_f) \triangleq \|f_{\theta_f}(h_{\theta_h}(s)) - s\| \tag{2.52a}$$

$$\mathcal{L}_{z\text{-dynamics}}(\theta_m) \triangleq -\log p_{\theta_m}(\texttt{SG}[h_{\theta_h}(s')] | \texttt{SG}[h_{\theta_h}(s)], a) \tag{2.52b}$$

$$\mathcal{L}_{\text{r-dynamics}}(\theta_h, \theta_m) \triangleq -\log r_{\theta_m}(r' | h_{\theta_h}(s), a), \tag{2.52c}$$

where $\texttt{SG}[\cdot]$ denotes the "stop-gradient" operation, indicating that we treat its arguments as constants when computing derivatives. It is important to stop the gradients w.r.t. the representation network in $\mathcal{L}_{z\text{-dynamics}}$, detaching the representation learning from the latent dynamics learning problems (Ha and Schmidhuber, 2018; Hafner et al., 2019a), in order to avoid interference between the two losses. E.g., a trivial latent state transition model $h_{\theta_h}(s) = \mathbf{0}, \forall s \in \mathbb{S}$ minimises the $\mathcal{L}_{z\text{-dynamics}}$ but in expense of hurting $\mathcal{L}_{\text{s-rec}}$.

Despite their simplicity, in practice, reconstruction-based world models are shown to interfere with the policy and value learning, i.e., lacking a mechanism to prioritise modelling aspects of the state space and dynamics that impact the most for sequential decision-making (Sutton, 1995; Farahmand et al., 2017; Lambert et al., 2020; Grimm et al., 2020).

Partial models. An alternative to reconstruction-based world models, are *partial (world) models*, i.e., models which instead of reconstructing the full state, they reconstruct some function (i.e., partial view) of the state, such as estimates of the state-value function or the optimal policy (Sutton, 1995; Oh et al., 2017; Farquhar et al., 2017; Farahmand et al., 2017; Amos et al., 2018; Gregor et al., 2019; Schrittwieser et al., 2020; Hessel et al., 2021). Following Rezende et al.'s (2020) notation, we write y_t for the partial view of the state s_t and, instead of the reconstruction loss, the *prediction* network, see *Figure 2.9d*, is trained with:

$$\mathcal{L}_{\text{partial}}(\theta_f, \theta_h) \triangleq \|f_{\theta_f}(h_{\theta_h}(s_t)) - y_t\|. \tag{2.53}$$

Note that for $y_t = s_t$ we recover the reconstruction world model loss $\mathcal{L}_{\text{rec}}$ as a special case.

Multi-step models. So far we have focused on training one-step world models since, in theory, they are sufficient for planning (§2.3.3; Sutton, 1995; Sutton and Barto, 2018). However, in practice, we usually unroll the learned models for more than one steps, which leads to compounding model errors and hence poor performance (Sun et al., 2018). To that end, Amos et al. (2018); Luo et al. (2018); Guo et al. (2018); Gregor et al. (2019); Asadi et al. (2019); Schrittwieser et al. (2020); Hessel et al. (2021) train world models to make multi-step (open-loop) action-conditioned predictions, e.g., for some positive integer K:

$$\mathcal{L}_{\text{K-partial}}(\theta_h, \theta_m, \theta_f) \triangleq \sum_{k=0}^{K} \|f_{\theta_f}(z_t^k) - y_{t+k}\|, \tag{2.54}$$

where $(z_t^k, r_t^k) \sim m_{\theta_m}(\cdot, \cdot | z_t^{k-1}, a_{t+k-1})$, $z_t^0 = z_t = h_{\theta_h}(s_t)$ and z_t^k indicate the latent state reached after unrolling the model for k-step starting from latent state z_t and taking actions $(a_t, a_{t+1}, \dots, a_{t+k-1}) \triangleq a_{t:t+k-1}$. Note that since $\mathcal{L}_{\text{K-partial}}$ is a function of all the sub-network parameters, i.e., $(\theta_h, \theta_m, \theta_f)$, we can train a world model by just minimising $\mathcal{L}_{\text{K-partial}}$ (and $\mathcal{L}_{\text{r-dynamics}}$), e.g., eliminating the latent state dynamics loss $\mathcal{L}_{\text{z-dynamics}}$.

Example 2.7 (MuZero model learning)***:*** The MuZero (Schrittwieser et al., 2020) world model is a multi-step partial model, which uses estimates of the state-value function and optimal policy as the state's partial view, i.e., $y_t = (\hat{\pi}_t, \hat{v}_t)$, trained with:

$$\mathcal{L}_{\text{MuZero}}(\theta_h, \theta_m, \theta_f) \triangleq \sum_{k=0}^{K} (l_v^k + l_\pi^k + l_r^k), \tag{2.55}$$

where $l_v^k \triangleq \|v_{\theta_f}(z_t^k) - \hat{v}_{t+k}\|$, $l_\pi^k \triangleq \|\pi_{\theta_f}(z_t^k) - \hat{\pi}_{t+k}\|$ and $l_r^k \triangleq \|r_{\theta_m}(z_t^k) - r_{t+k}\|$.

2.3.3 Planning

Provided a (world) model of the environment, the agent can make counterfactual predictions and *plans* for either action selection (*planning for behaviour/acting*, Richalet et al., 1978) or policy/value learning (*planning for learning*, Sutton and Barto, 2018; Hamrick et al., 2020). Intuitively, we refer to planning as any computational process that refines policy and/or value estimates[†] $(\hat{v}, \hat{\pi})$ using a (learned) model of the environment $\hat{m}$, i.e.,

$$\texttt{planning}(\hat{v}, \hat{\pi}; \hat{m}) \mapsto (\bar{v}, \bar{\pi}), \quad \text{s.t.} \quad \begin{cases} \|\bar{v} - v^\pi\| \leqslant \|\hat{v} - v^\pi\| & (a) \\ v^{\bar{\pi}} \succeq v^{\hat{\pi}} & (b) \end{cases}. \tag{2.56}$$

We should interpret *Eqn.* (2.56) as a desideratum rather than a constraint since the validity of the inequalities highly depends on the quality of the model $\hat{m}$. We expect that a planning algorithm, if given the "true" environment model m^*, it should satisfy *Eqn.* (2.56).

On the one hand, when we plan for behaviour (Richalet et al., 1978), we act according to the refined policy $\bar{\pi}$, computed just for the state of interest s. The better the world model and the greater the compute budget, we can render significant improvement over the "baseline" policy $\hat{\pi}$ (Silver et al., 2016; Brown and Sandholm, 2019; Lerer et al., 2020).

On the other hand, when we plan for learning (Werbos, 1987; Sutton, 1991), we compute the refined predictions $(\bar{v}, \bar{\pi})$ for replayed states from the rollouts $\mathcal{B}$ and distil the improved via planning estimates to the parametric function approximators, e.g., by minimising:

$$\mathcal{L}_{(v,\pi)\text{-distillation}}(\theta_v, \theta_\pi) \triangleq \mathbb{E}_{s_t \sim \mathcal{B}}[\|\bar{v}_t - v_{\theta_v}(s_t)\| + \text{KL}[\bar{\pi}_t \| \pi_{\theta_\pi}(\cdot|s_t)]], \tag{2.57}$$

where KL$[\cdot\|\cdot]$ denotes the Kullback–Leibler (KL, Kullback and Leibler, 1951) divergence. It is also common for RL agents to plan for both behaviour and learning (Schrittwieser

[†]A similar formulation can be made using action value estimates $\hat{q}$, as in (Sutton, 1991).

et al., 2020). Next, we review planning algorithms that are used later in this thesis.

Dynamic programming. Bellman (1957b) exploited the recursive nature of the value and action-value functions (see *Eqns.* (2.40, 2.42)) and showed that the computation of the value of a policy (a.k.a. *policy evaluation*) and can be concisely formulated using Bellman evaluation operators. In particular, the *one-step* Bellman evaluation operator, applied on a state-(to-scalar) function $v \in \mathbb{V} \triangleq \{f : \mathbb{S} \to \mathbb{R}\}$ is formally defined, next.

> ***Definition 2.1*** (Bellman evaluation operator)***:*** Given the "true" environment model m^* and a policy π, the one-step Bellman evaluation operator $\mathscr{T}^{\pi} : \mathbb{V} \to \mathbb{V}$ is induced, and its application on a state-function $v \in \mathbb{V}$, for all $s \in \mathbb{S}$, is given by:
>
> $$\mathscr{T}^{\pi} v(s) \triangleq \mathbb{E}_{\pi, m^*} \left[R_1 + \gamma v(S_1) | S_0 = s \right]. \tag{2.58}$$

The Bellman operator, $\mathscr{T}^{\pi}$, can be self-cascaded, i.e., the k-times repeated application of the one-step Bellman operator gives rise to the k-step Bellman operator, i.e.,

$$(\mathscr{T}^{\pi})^k v \triangleq \underbrace{\mathscr{T}^{\pi} \cdots \mathscr{T}^{\pi}}_{\text{k-times}} v. \tag{2.59}$$

The Bellman evaluation operator, $\mathscr{T}^{\pi}$, is a contraction mapping (Bellman, 1957b; Bertsekas, 2012; Puterman, 2014), and its fixed point is the value of the policy π, i.e.,

$$\lim_{n \to \infty} (\mathscr{T}^{\pi})^n v = v^{\pi}, \forall v \in \mathbb{V}. \tag{2.60}$$

Therefore, the Bellman evaluation operators satisfy the desideratum (a) in *Eqn.* (2.56) and hence give rise to planning algorithms that, given m^*, improve the value estimates. Similarly we define Bellman optimality operators for the optimal value function.

> ***Definition 2.2*** (Bellman optimality operator)***:*** Given the "true" environment model m^*, the one-step Bellman optimality operator $\mathscr{T}^* : \mathbb{V} \to \mathbb{V}$ is induced, and its application on a state-function $v \in \mathbb{V}$, for all $s \in \mathbb{S}$, is given by:
>
> $$\mathscr{T}^* v(s) \triangleq \max_{a \in \mathbb{A}} \mathbb{E}_{m^*} \left[R_0 + \gamma v(S_1) | S_0 = s, A_0 = a \right]. \tag{2.61}$$

The Bellman optimality operator, $\mathscr{T}^*$, is also a contraction mapping (Bellman, 1957b; Bertsekas, 2012; Puterman, 2014), and its fixed point is the optimal value function, i.e.,

$$\lim_{n \to \infty} (\mathscr{T}^*)^n v = v^*, \forall v \in \mathbb{V}. \tag{2.62}$$

Model-induced Bellman operators. A model $\hat{m}$ and policy π induce a Bellman evaluation (resp. optimality) operator $\mathscr{T}^{\pi}_{\hat{m}}$ (resp.$\mathscr{T}^{*}_{\hat{m}}$) with a fixed point $v^{\pi}_{\hat{m}}$ (resp. $v^{*}_{\hat{m}}$).

Definition 2.3 (model-induced Bellman evaluation operator)***:*** Given world model $\hat{m}$ and a policy π, the one-step Bellman evaluation operator $\mathscr{T}^{\pi}_{\hat{m}} : \mathbb{V} \to \mathbb{V}$ is induced, and its application on a state-function $v \in \mathbb{V}$, for all $s \in \mathbb{S}$, is given by:

$$\mathscr{T}^{\pi}_{\hat{m}} v(s) \triangleq \mathbb{E}_{\pi,\hat{m}} \left[R_1 + \gamma v(S_1) | S_0 = s \right]. \tag{2.63}$$

Definition 2.4 (model-induced Bellman optimality operator)***:*** Given world model $\hat{m}$, the one-step Bellman optimality operator $\mathscr{T}^{*}_{\hat{m}} : \mathbb{V} \to \mathbb{V}$ is induced, and its application on a state-function $v \in \mathbb{V}$, for all $s \in \mathbb{S}$, is given by:

$$\mathscr{T}^{*}_{\hat{m}} v(s) \triangleq \max_{a \in \mathbb{A}} \mathbb{E}_{\hat{m}} \left[R_1 + \gamma v(S_1) | S_0 = s, A_0 = a \right]. \tag{2.64}$$

Similar to *Eqn.* (2.59), a k-step model-induced Bellman operator is given by:

$$(\mathscr{T}^{\pi}_{\hat{m}})^k v = \underbrace{\mathscr{T}^{\pi}_{\hat{m}} \cdots \mathscr{T}^{\pi}_{\hat{m}}}_{k\text{-times}} v. \tag{2.65}$$

In §5, we use the model-induced Bellman operator to construct multiple value estimates from a point estimate of a learned world model, which in turn use to quantify ignorance.

Despite their simplicity and theoretical guarantees, for most problems of interest, it is impractical to apply exactly Bellman operators since they involve the calculation of computationally intractable expectations. Instead, sampling-based approximations are used, which are reminiscent of temporal difference (TD) value estimators, introduced in §2.3.2, applied on model-generated rollouts (Sutton, 1991; Heess et al., 2015; Feinberg et al., 2018; Hafner et al., 2019b; Byravan et al., 2020). For instance, the single-sample, k-step model-based value estimation of the policy π, with prior value estimator $\hat{v}$, is given by:

$$\hat{q}^k(s, a) = \sum_{i=0}^{k-1} (\gamma^i r^{i+1}) + \gamma^k \hat{v}(s^k), \tag{2.66}$$

where $s^0 = s$, the model samples $(s^{i+1}, r^{i+1}) \sim \hat{m}(\cdot, \cdot | s^i, a^i)$ and actions $a^i \sim \pi(\cdot | s^i)$.

Open-loop lookahead planning. We can form model-based estimates of the value of the k-step lookahead policy, denotes as $\hat{q}(s, \mathbf{a}^{0:k-1})$ and defined in *Eqn.* (2.34), by replacing the policy-sampled actions in *Eqn.* (2.66) with the open-loop plan $\mathbf{a}^{0:k-1}$. Then the agent selects the plan that maximises the model-based action-value estimate, i.e.,

$$\hat{\mathbf{a}} \triangleq \arg\max_{\mathbf{a} \in \mathbb{A}^k} \hat{q}(s, \mathbf{a}), \tag{2.67}$$

In the case of deep RL with continuous actions, we can use gradient-based optimisation to solve the problem in *Eqn.* (2.67), a method that is termed *planning with value gradi-*

ents (Heess et al., 2015). With discrete actions, biased gradient estimators (Hafner et al., 2020) or zero-order optimisation methods (Hafner et al., 2019b) can be used.

Model predictive control. In open-loop lookahead planning for behaviour, it is common that the agent executes *only* the first action $\hat{a}^1$ from $\hat{\mathbf{a}}$ to the environment, observes the new environment state and reward and re-plans. This iterative process is also known as receding horizon planning and *model predictive control* (MPC, Richalet et al., 1978).

2.3.4 Overoptimising a learned objective

A recurring theme in a number of sequential-decision making settings and methods, have been the requirement to optimise some quantity that is the output of a learned model. A few examples are: (i) in behaviour cloning (BC, see §2.2.4), the agent selects the action that maximises the likelihood under a learned approximation of the expert policy; (ii) in Q-learning (see §2.3.2), the agent selects the action that maximises its learned action-value function approximation; (iii) in actor-critic methods (see §2.3.2), the agent updates its policy in such a way that it maximises its learned action-value (or advantage) function; and (iv) in planning with a learned model (see §2.3.3), the agent both selects actions and updates its policy based on rollouts that are purely generated from a learned model.

Naively treating these settings as regular optimisation problems and employing powerful solvers to the learned objectives,[†] can eventually damage the true objectives, a phenomenon that we refer to as *overoptimisation* (Gao et al., 2023). Overoptimisation can be seen as a special case of Goodhart's Law[‡] (Hoskin, 1996) or specification gaming (Krakovna et al., 2020). In the context of deep learning (DL), overoptimisation is popularised in the field of adversarial robustness (Chakraborty et al., 2018), in which it can be shown that by taking a regular pre-trained neural network (NN) and freezing its learned parameters, we can find, via gradient-descent or even zero-order optimisers, perturbations to the inputs that while qualitatively[§] trivial, they cause the model to generate arbitrarily bad predictions. In model-based RL, it is often the case that naively planning with a learned model leads to overoptimisation due to compounding model errors that lead to degenerate performance, despite the learned objective being successfully optimised (Janner et al., 2019).

A commonly proposed solution in the literature to the problem of overoptimisation of a learned objective is ignorance-aware optimisation (Deisenroth et al., 2014; Stadie et al., 2015; Rajeswaran et al., 2016; Pathak et al., 2017; Kahn et al., 2017; Chua et al., 2018; Kenton et al., 2019; Gleave and Irving, 2022). In particular, these methods modify the optimisation objective, factoring in ignorance estimates, e.g., avoid optimisation close to regions with high ignorance. The concept of knowledge equivalent (KE) from economics that

[†]A quantity that is derived from a learned model and is used as an optimisation target.
[‡] *"When a measure becomes a target, it ceases to be a good measure."*
[§]Most of the times there is a human-in-the-loop evaluation/verification for this.

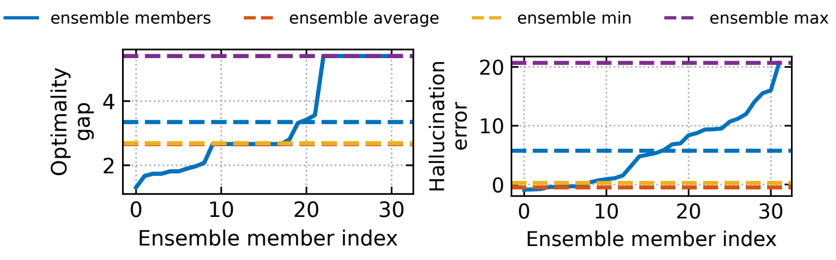
we introduce in §2.1.2 (see *Remark* 2.4) formalises this class of methods, and it will be our main approach for tackling overoptimisation of learned objectives in this thesis.

Example 2.8 (overoptimising a learned ignorance-aware reward model)***:*** We revisit the results of the *Example* 2.3. In particular, we study if optimising against the learned reward model(s) leads to overoptimisation and then how ignorance quantification and different KEs impact on the optimality gap and hallucination gap. The optimality gap is defined as the difference in "true" reward of the optimal action (which is unknown to the agent) and the action the agent selects according to its learned model, i.e., regret to the optimal policy. The hallucination gap is defined as the difference between the agent's belief about the reward of the selected action and its "true" reward.

As a reminder, we train a deep ensemble of reward models, $\{r_{\theta_k} : \mathbb{A} \to \mathbb{R}\}_{k=1}^{32}$, according to the training strategy in *Remark* 2.7 on the finite aciont$\to$reward pairs, depicted in *Figure 2.5a*. The per-ensemble member predictions and the corresponding KEs (e.g., ensemble average, minimum and maximum) are illustrated in *Figure 2.5d*.

The agent selects the action that maximises its reward model (or some KE in the case of ignorance-aware reward model). In *Figure 2.10a* (resp. 2.10b), we visualise in solid blue the optimality gap (resp. hallucination error) of the agent that acts greedily with respect to each of the $K = 32$ ensemble members (sorted from low to high for illustration purposes) and the average (across the ensemble members) optimality gap (resp. hallucination error) is depicted with the dashed blue line.

We observe that the ignorance-averse and -seeking KEs incur the lowest overoptimisation/error, even smaller than the average of the ensemble members, and the ignorance-seeking KE has the highest error. This observation justifies the use of ignorance-averse KEs for robustness and ignorance-seeking KEs for exploration (Savage, 1954).



(a) Optimality gap ***(b)*** Hallucination error

Figure 2.10: Overoptimisation with an ensemble of reward models. ***(a)*** The optimality gap, i.e., agent's regret to the optimal policy. ***(b)*** The hallucination error, i.e., difference between the agent's reward estimate for its action and its true value.

2.3.5 Ignorance-aware RL agents

In §2.3.4, we noted that numerous RL agents, such as Q-learning (see §2.3.2), actor-critic methods (see §2.3.2), agents that plan with a learned world model (see §2.3.3), are susceptible to the overoptimisation phenomenon. To that end, ignorance-aware variants of these methods can be found in the literature (Dearden et al., 1998, 1999; Osband et al., 2016; Kurutach et al., 2018), which often comprise of: (i) an ignorance quantification mechanism; and (ii) a KE as the learning/planning objective.

Ignorance quantification for RL. In tabular settings (i.e., environments with small and finite number of states and actions), exact Bayesian inference (see §2.2.1) can be used for quantifying the agents' ignorance about both their value function (Dearden et al., 1998) and world model (Dearden et al., 1999). However, in complex RL problems, since exact Bayesian inference is intractable, proxy signals are often used instead, including prediction error (Lopes et al., 2012), approximate state visitation counts (Bellemare et al., 2016) and disagreement of samples from either approximate posterior distributions over learned parameters (Blundell et al., 2015) or explicit ensembles of value functions (Osband et al., 2016; Lowrey et al., 2018) or world models (Chua et al., 2018; Sekar et al., 2020).

Ensemble-based RL methods. Deep ensembles (see §2.2.3) of learned (action-)value functions and world models can be used with an appropriately selected KE for: (i) stabilising learning (§2.3.4; Faußer and Schwenker, 2015; Anschel et al., 2017; Kalweit and Boedecker, 2017; Kurutach et al., 2018; Chua et al., 2018; Liang et al., 2022); (ii) ignorance-driven exploration (§2.1.2, Osband et al., 2016; Shyam et al., 2019; Pathak et al., 2019; Flennerhag et al., 2020; Ball et al., 2020; Sekar et al., 2020); and (iii) robustness to distribution shifts (Lowrey et al., 2018; Kenton et al., 2019; Agarwal et al., 2020).

Example 2.9 (deep ensemble of action-value functions and knowledge equivalent (KE) optimisation)***:*** Consider an ensembles of K action-value functions $\{q_{\theta_{q,k}}\}_{k=1}^{K}$, trained on some rollouts $\mathcal{B}$ with the training protocol in *Remark* 2.7. Then to stabilise learning, e.g., with Averaged Q-learning, Anschel et al. (2017) uses as value target the ensemble average (see *Eqn.* (2.5)) (ignorance-neutral) KE, given by:

$$\hat{q}_{\text{Averaged}-\text{QL}}(s, a) \triangleq r + \gamma \max_{a' \in \mathbb{A}} (\frac{1}{K} \sum_{k} q_{\theta_{q,k}}(s', a')). \tag{2.68}$$

For exploration (resp. robustness), we select actions by maximising the ignorance-seeking ensemble max (resp. ignorance-averse ensemble min) KE in *Eqn.* (2.7b) (resp. *Eqn.* (2.7a)), i.e., "optimism (resp. pessimism) in the face of ignorance":

$$\pi_{\text{explore}}(s) = \max_{a \in \mathbb{A}} \max_{k} q_{\theta_{q,k}}(s, a) \quad (\text{resp. } \pi_{\text{safe}}(s) = \max_{a \in \mathbb{A}} \min_{k} q_{\theta_{q,k}}(s, a)). \tag{2.69}$$

"It does not do to leave a live dragon out of your calculations, if you live near one."

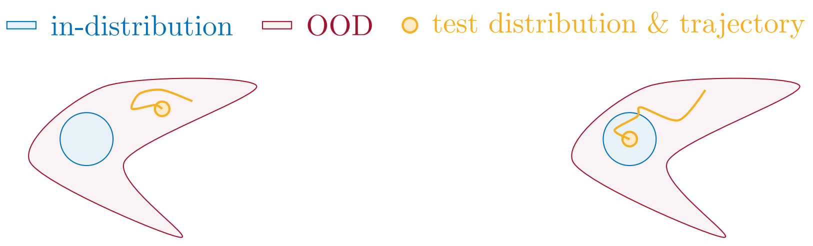
— John Ronald Reuel Tolkien (1892–1973)

3 Plan and adapt from expert demonstrations

Contents

Out-of-training distribution (OOD) scenarios are a common challenge for learning agents, which usually leads to arbitrary deductions and poorly informed decisions due to their failure to generalise (Sugiyama and Kawanabe, 2012; Amodei et al., 2016; Snoek et al., 2019). In §2.2, we defined OOD data as samples for which the independent and identically distributed (i.i.d.) assumption with respect to (w.r.t.) the training distribution does *not* hold, and we illustrated an example in *Figure 2.3*. In the context of imitation learning (IL, see §2.2.4), which is the focus of this chapter, and sequential decision-making, more broadly, there are two main reasons for which agents encounter OOD states, i.e., they undergo a *distribution shift*, visualised in *Figure 3.1*: (i) the agents are deployed in states that are far from the training distribution, as shown in *Figure 3.1a*; or (ii) the agents are deployed in in-distribution states but small deviations from the demonstrated expert be-

[*]This chapter is based on the (Filos et al., 2020) publication.



(a) Domain shift *(b)* Covariate shift

Figure 3.1: Sequential decision-making and out-of-training distribution (OOD) settings. The agent is trained on the in-distribution states and deployed on the test states, from which it follows a trajectory in OOD states. ***(a)*** The agent is deployed in novel/OOD scenarios, i.e., it undergoes a domain shift. ***(b)*** The agent is deployed in in-distribution actions but its model imperfections lead to compounding errors that finally push it in OOD states, i.e., it undergoes a covariate shift. We refer to both settings as distribution shift.

haviour, due to modelling errors, compound and drift the agents to OOD states, in which the agents make uninformed decisions that keep pushing them to OOD states, leading to a vicious circle (a.k.a. *"avalanche" phenomenon* or *covariate shift*) (Ross et al., 2011), depicted in *Figure 3.1b*. Therefore, IL agents, unless they have been trained with expert demonstrations that exhaustively cover nearly *all* the state-action space, they should be designed with the prospect of encountering distribution shifts upon deployment.

In principle, detection of and adaptation to OOD situations can mitigate their adverse effects. In this chapter, we highlight the limitations of current state-of-the-art density estimation approaches to imitation learning (IL) and we present a novel ignorance-aware algorithm for robust planning from expert demonstrations, called *robust imitative planning* (RIP). Our method relies on distinctly modelling the agent's ignorance, and the inherent randomness of the expert demonstrations, i.e., risk, as defined in *Remark* 2.1. Then the agent uses its ignorance quantification mechanism to detect potential distribution shifts and recover from some of them by penalising decisions with high ignorance, i.e., optimising a ignorance-averse knowledge equivalent (KE), as presented in §2.1.2.

Moreover, if the agent's ignorance is too great to suggest a safe course of action, it can instead query an expert for feedback, enabling sample-efficient online adaptation—an online, adaptive variant of our method we term *adaptive robust imitative planning* (AdaRIP).

We provide empirical evidence in both online experiments in a simulated autonomous driving environment (`CARLA`, Dosovitskiy et al., 2017) and the offline prediction `nuScenes` challenge (Caesar et al., 2019) that our proposed methods: (i) can scale to high-dimensional LIDAR observations; (ii) highlight the important role of ignorance-aware models for detecting, recovering from and adapting to OOD scenarios. The key contributions are:

List of contributions** §3* (plan and adapt from expert demonstrations):***

1. **Ignorance-aware planning from expert demonstrations:** We present an ignorance-aware agent that plans from expert demonstrations, called *robust imitative planning* (RIP). It comprises of a likelihood model, e.g., autoregressive normalising flows (Rezende and Mohamed, 2015), learned with maximum likelihood estimation (MLE) given the expert demonstrations that quantifies ignorance, e.g., deep ensemble (Lakshminarayanan et al., 2017). The principle of *"pessimism in the face of uncertainty (ignorance)"* is used for RIP and therefore the agent plans by maximising an ignorance-averse knowledge equivalent (KE), such as the ensemble min (see §2.1.2), allowing for robustness to and recovery from distribution shifts. A didactic example of RIP is depicted in *Figure 3.6*.
2. **Ignorance-driven sample-efficient adaptation:** Provided a mechanism for online feedback from an expert, the quantified ignorance is used to efficiently query the expert for feedback—only when insufficiently certain about what to do. We term this online and adaptive variant of RIP as *adaptive robust imitative planning* (AdaRIP) and demonstrate in a simulated autonomous driving environment that with a handful of targeted queries to an expert, it can successfully adapt and solve originally OOD tasks that RIP and baselines struggle with.
3. **Autonomous driving robustness benchmark:** We introduce an autonomous driving benchmark, called `CARNOVEL`, to assess the robustness of autonomous driving methods to a suite of OOD tasks. In particular, we evaluate them in terms of their ability to: (i) detect OOD events, measured by the correlation of infractions and model uncertainty; (ii) recover from distribution shifts, quantified by the percentage of successful manoeuvres in novel scenes and (iii) efficiently adapt to OOD scenarios, provided online supervision, assessed by their few-shot adaptation performance.

3.1 Background & problem setting

We model the agent's interaction with the environment as a Markov decision process (MDP, Puterman, 2014), i.e., $\mathcal{M} \triangleq \langle \mathbb{S}, \mathbb{A}, p, \rho_0, r \rangle$, as formulated in §2.3.1 and *Eqn.* (2.27). The agent does not get to observe the reward function, $r : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$, or sample from it, but has access to rollouts from an optimal policy, as defined in *Eqn.* (2.29a).

Learning from expert demonstrations. The agent's goal is to approximate the unknown expert demonstrator's policy, π^*, using imitation learning (IL, Widrow, 1964; Pomerleau, 1989) methods on the expert demonstrations, as described in §2.2.4. The agent is then evaluated: (i) offline for its predictions to a held-out set of expert demonstrations; (ii) online by interaction with an environment without online supervision; and (iii) with online expert supervision (i.e., fine-tuning on few-shot expert demonstrations).

Formally, the (finite number of N) expert demonstrations are given by:

$$\mathcal{D} \triangleq \left\{ \left(s^{(n)}, a^{(n)} \right) \right\}_{n=1}^{N}, \quad \text{s.t.} \quad a^{(n)} \sim \pi^*(\cdot | s^{(n)}). \tag{2.23}$$

Without loss of generality, we assume that the optimal policy is *stochastic*. For brevity, we denote T-step long expert plans with $\mathbf{a}$ and denote expert demonstrations with:

$$\mathcal{D} \triangleq \left\{ \left(s^{(i)}, \mathbf{a}^{(i)} \right) \right\}_{i=1}^{I}, \text{ s.t. } \left(s^{(i)}, \mathbf{a}^{(i)} \right) \triangleq \left(s_t^{(n)}, a_{t:t+T}^{(n)} \right), \text{ for some } I \in \mathbb{N}. \tag{3.1}$$

While, in general, for *any* MDP there is an optimal *deterministic* policy (Puterman and Shin, 1978; Sutton and Barto, 2018), it is too restrictive to make this assumption here since the gathering of the demonstrations is not under our control. For example, in *Figure 3.2*, we visualise the aggregated coordinates of expert driving demonstrations, used in §3.3. We observe that the trajectories followed by the expert are multi-modal, e.g., both right and left turns from (almost) the same starting positions, especially at intersection points.

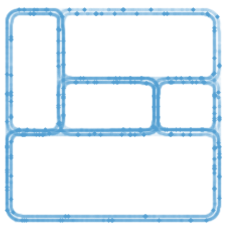


Figure 3.2: Expert demonstrations coverage.

Imitative model. We can perform state-conditioned density estimation (see *Eqn.* (2.11)) of the distribution over expert's future sequence of actions (i.e., plans), using a probabilistic model $q(\mathbf{A}|s; \theta)$, with learnable parameters $\theta \in \Theta$, trained on the expert demonstrations $\mathcal{D}$, e.g., via maximum likelihood estimation (MLE, *Eqn.* (2.15)):

$$\theta_{\text{MLE}} \triangleq \arg\max_{\theta} \mathbb{E}_{(s^{(i)}, \mathbf{a}^{(i)}) \sim \mathcal{D}} \left[\log q(\mathbf{A} = \mathbf{a}^{(i)} | s^{(i)}; \theta) \right]. \tag{3.2}$$

We refer to this class of models as *imitative models*, since we are borrowing Rhinehart et al.'s (2020) deep imitative model (DIM) neural network (NN) architecture. In particular, we represent $q(\mathbf{A}|s; \theta)$ as an autoregressive neural density estimator (Larochelle and Murray, 2011; Graves, 2013; Gregor et al., 2014), and the likelihood of a plan $\mathbf{a} \triangleq (a_0, a_1, a_2, \dots, a_T)$ in state s to come from an expert under the model is given by:

$$q(\mathbf{A} = \mathbf{a} | s; \theta) = \prod_{t=1}^{T} q(A_t = a_t | \mathbf{a}_{<t}, s; \theta), \tag{3.3}$$

where $\mathbf{a}_{<t} \triangleq (a_1, a_2, \dots, a_{t-1})$ is the sub-plan up to (without including) time-step t. For brevity, we refer to $q(\mathbf{A} = \mathbf{a}|s; \theta)$ as the *imitative score* of plan $\mathbf{a}$ under model θ.

We decompose the likelihood as a telescopic product,[†] and in the case of a continuous

[†]We consider here a causal factorisation of the joint distribution to have a fair comparison with the baselines which make a similar decision. Our method does *not* depend on this and hence any other valid,

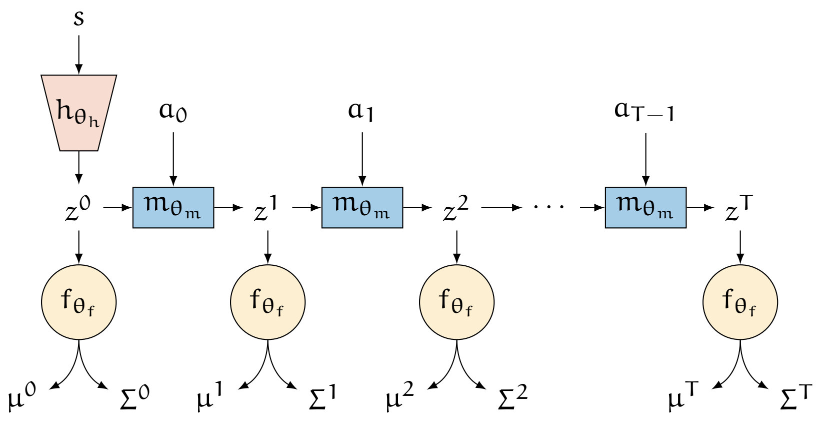


Figure 3.3: Imitative model neural network (NN) architecture for continuous actions. Following the terminology from §2.3.2, there are three sub-NNs: (i) the representation network h_{θ_h}; (i) the dynamics network m_{θ_m}; and (iii) the prediction network f_{θ_f}.

action space, which is the focus in our experiments in §3.3, the conditional densities are assumed to be normally distributed with predicted mean and covariance, i.e.,

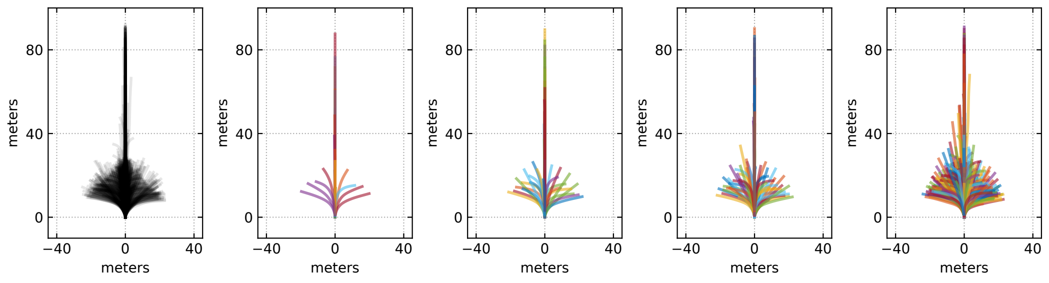
$$q(A_t = a_t|\mathbf{a}_{<t}, s; \theta) = \mathcal{N}(a_t; \mu(\mathbf{a}_{<t}, s; \theta), \Sigma(\mathbf{a}_{<t}, s; \theta)) \tag{3.4}$$

where $\mu(\cdot;\theta)$ and $\Sigma(\cdot;\theta)$ are the NN-based predictions for the mean and covariance.

The architecture of the NN for the imitative model is depicted in *Figure 3.3*. It comprises of three sub-networks, similar to the ones in §2.3.2 and *Figure 2.9* for the (partial) latent world models: (i) a representation network h_{θ_h} that embeds the (possibly) high-dimensional state s into an embedding z; (ii) an action-conditioned dynamics network m_{θ_m} that models the transitions in the embedding space; and (iii) a prediction network f_{θ_f} that outputs the estimates for the mean and the covariance for the continuous action. Unless stated otherwise, we use (i) a convolutional NN (MobileNetV2, Sandler et al., 2018) as the representation network; (ii) a recurrent NN (gated recurrent unit (GRU), Chung et al., 2014) as the dynamics network; and (iii) a multi-layer perceptron (MLP, see §2.2.2) as the prediction network. For brevity, we refer to the NN parameters as $\theta \triangleq (\theta_h, \theta_m, \theta_f)$ and write $q(\mathbf{A} = \mathbf{a}|s;\theta) = \prod_t q(A_t = a_t|\mathbf{a}_{<t}, s;\theta)$, where $q(A_t = a_t|\mathbf{a}_{<t}, s;\theta) = \mathcal{N}(a_t; \mu^t; \Sigma^t)$ and (μ^t, Σ^t) the prediction network outputs for unroll-step t.

Remark 3.1 (imitative model as multi-step action-value function)***:*** Functionally and intuitively, the imitative score, $q(\mathbf{A} = \mathbf{a}|s;\theta)$, resembles the optimal action-value of a multi-step lookahead policy, $q^*(s, \mathbf{a})$, as defined in *Eqn.* (2.34), since both evaluate the quality of a plan $\mathbf{a}$ executed from a given state s. In particular, $q(\mathbf{A} = \mathbf{a}|s;\theta)$

potentially anti-causal, factorisation is also compatible with it. For instance, we could reverse the "arrow of time", i.e., $q(\mathbf{a}|s;\theta) = \prod_{t=T}^{1} p(a_t|\mathbf{a}_{>t}, s;\theta)$, and leave the rest of the method unchanged.



(a) Trajectories *(b)* $K = 32$ *(c)* $K = 64$ *(d)* $K = 128$ *(e)* $K = 1024$

Figure 3.4: Trajectory library from `CARLA`'s autopilot demonstrations, constructed with K-means clustering on 4 seconds-long expert plans from the training dataset.

estimates the likelihood of the plan under the expert policy and $q^*(s, \mathbf{a})$ is the expected return of first executing the plan $\mathbf{a}$ and then following the expert policy. Although the likelihood and the reward are two quantities with different units, there are theoretical frameworks (Kappen et al., 2012; Levine, 2018) that connect the two. In this chapter, we do not explore further any theoretical connections between these two quantities but we exploit this observation by applying the $q^*(s, \mathbf{a})$-based planning algorithms and formulations from §2.3.3 to the imitative score, $q(\mathbf{A} = \mathbf{a}|s; \theta)$.

Planning. The learned imitative model can be used for action selection. In particular, we pick the *mode* of the state-conditioned distribution over plans, induced by the model, i.e.,

$$\hat{\mathbf{a}} \triangleq \arg\max_{\mathbf{a}} q(\mathbf{A} = \mathbf{a}|s; \theta) = \arg\max_{\mathbf{a}} \log q(\mathbf{A} = \mathbf{a}|s; \theta), \tag{3.5}$$

where, in practice, we maximise the log-likelihood to avoid any numerical instabilities.[†] As discussed in §2.3.3, for continuous action spaces, the optimisation problem in *Eqn.* (3.5), can be solved online with gradient-based optimisation since the gradient $\partial q / \partial \mathbf{a}$ can be computed with automatic differentiation techniques (§2.2.2; Rall, 1981). Only the first action $\hat{a}^1$ of the plan $\hat{\mathbf{a}}$ is executed as in model predictive control (MPC, see §2.3.3) and a new plan is recomputed upon observing the next environment state.

Trajectory library. In real-world applications, e.g., autonomous driving or more generally robotics, the on-device computational resources may not suffice for running online planning (Jund et al., 2021), either because it is too slow for real-time reaction, or the gradient calculation for *Eqn.* (3.5) does not fit the memory device memory. A scalable alternative that we consider in this chapter is to efficiently search for plans by restricting the search space to a *trajectory library* (Liu and Atkeson, 2009), $\mathbb{T}_{\mathbb{A}}$, i.e., a finite set of fixed (pre-computed) plans. In other words, we discretise the continuous (and infinite) space of plans to a finite number of plans, turning an optimisation problem of continuous

[†]We refer to both $q(\mathbf{A} = \mathbf{a}|s; \theta)$ and $\log q(\mathbf{A} = \mathbf{a}|s; \theta)$ as the imitative score, depending on the context.

variables into a search problem over a discrete space which can be easily parallelised:

$$\hat{\mathbf{a}}_{\text{TL}} \triangleq \arg\max_{\mathbf{a} \in \mathbb{T}_{\mathbb{A}}} \log q(\mathbf{A} = \mathbf{a} | s; \theta). \tag{3.6}$$

The computational cost gains come at the cost of approximation error. In this work, to construct the trajectory library $\mathbb{T}_{\mathbb{A}}$, we perform K-means clustering of the expert's plans from the training distribution and keep 64 of the centroids, as illustrated in *Figure 3.4*.

3.2 Methods

The main challenge of planning with an imitative model is the potential to overoptimise a learned objective, as discussed in §2.3.4, since the agent is explicitly maximising the likelihood of a plan under a learned state-conditioned imitative model, see *Eqns.* (3.5, 3.6). When we only train the imitative model on an offline and static dataset of demonstrations, overoptisation during planning is more likely (Ross et al., 2011; Levine et al., 2020)

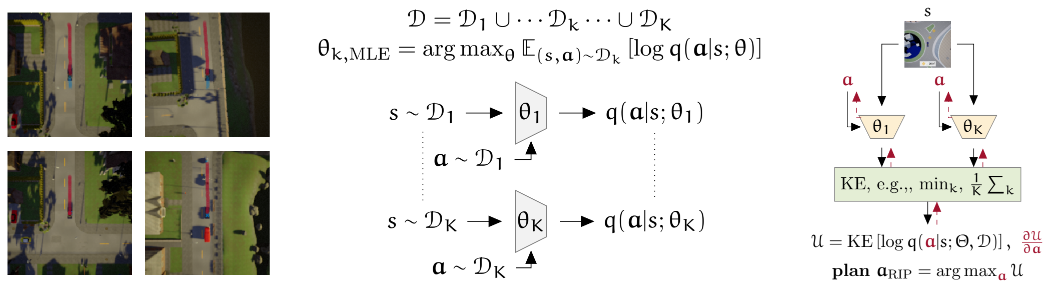
3.2.1 Robust imitative planning (RIP)

In this section, we present an agent that learns offline from expert demonstrations, as defined in §3.1, which can (i) effectively model stochastic expert demonstrations (i.e., aleatoric uncertainty); (ii) quantify ignorance (i.e., epistemic uncertainty) to allow detection of out-of-training distribution (OOD) situations; and (iii) avoid or recovers from distributions shifts via ignorance-based optimisation of a knowledge equivalent (KE). We call the proposed method *robust imitative planning* (RIP) and we illustrate it in *Figure 3.5*.

Formalisation. We treat the imitative model parameters θ as random variables (see §2.2.1), i.e., we place a prior distribution $p(\Theta)$ over possible imitative model parameters $\theta \in \Theta$, which induces a distribution over imitative scores $q(\mathbf{A} = \mathbf{a}|s; \Theta)$. Note the difference between the imitative score $q(\mathbf{A} = \mathbf{a}|s; \theta) \in \mathbb{R}$ and $q(\mathbf{A} = \mathbf{a}|s; \Theta) \in \Delta(\mathbb{R})$. The former is (deterministic) scalar quantity, i.e., the imitative score for a fixed (point estimate) θ, and the latter is a random variable induced by the model parameters random variable Θ. After observing the expert demonstrations $\mathcal{D}$, the posterior distribution over model parameters and imitative scores is $p(\Theta|\mathcal{D})$ and $q(\mathbf{A} = \mathbf{a}|s; \Theta, \mathcal{D})$, respectively, where:

$$q(\mathbf{A} = \mathbf{a}|s; \Theta, \mathcal{D}) = q(\mathbf{A} = \mathbf{a}|s; \Theta) p(\Theta|\mathcal{D}). \tag{3.7}$$

Remark 3.2 (posterior distribution over imitative scores captures only ignorance and not risk)***:*** $q(\mathbf{A} = \mathbf{a}|s; \theta)$ is a deterministic function of the parameters θ, i.e., con-


$\mathcal{D} = \mathcal{D}_1 \cup \cdots \mathcal{D}_k \cdots \cup \mathcal{D}_K$
$\theta_{k,\text{MLE}} = \arg\max_\theta \mathbb{E}_{(s,\mathbf{a})\sim\mathcal{D}_k}[\log q(\mathbf{a}|s;\theta)]$
$s \sim \mathcal{D}_1$
θ_1
$q(\mathbf{a}|s;\theta_1)$
$\mathbf{a} \sim \mathcal{D}_1$
$s \sim \mathcal{D}_K$
θ_K
$q(\mathbf{a}|s;\theta_K)$
$\mathbf{a} \sim \mathcal{D}_K$
s
$\mathbf{a}$
$\mathbf{a}$
θ_1
θ_K
KE, e.g.,, $\min_k$, $\frac{1}{K}\sum_k$
$\mathcal{U} = \text{KE}[\log q(\mathbf{a}|s;\Theta,\mathcal{D})]$, $\frac{\partial\mathcal{U}}{\partial\mathbf{a}}$
plan $\mathbf{a}_{\text{RIP}} = \arg\max_{\mathbf{a}} \mathcal{U}$

(a) Expert demonstrations ***(b)*** Ensemble training ***(c)*** Planning under ignorance

Figure 3.5: The robust imitative planning (RIP) agent. ***(a)*** Expert demonstrations. We assume access to states s and expert plans $\mathbf{a}$ pairs, i.e., $\mathcal{D} \triangleq \{(s^{(i)}, \mathbf{a}^{(i)})\}_i$, collected either in simulation (Dosovitskiy et al., 2017) or in real-world (Caesar et al., 2019; Sun et al., 2019; Houston et al., 2020). ***(b)*** Learning algorithm (see §3.2.1). We capture agent's ignorance by learning an ensemble of density estimators $\{q(\mathbf{a}|s;\theta_k)\}_{k=1}^K$, following the implementation of *Remark* 2.7. ***(c)*** Planning paradigm (see §3.2.1). The plan $\mathbf{a}_{\text{RIP}}$ that maximises an ignorance-averse knowledge equivalent (KE, see §2.1.2), e.g., the ensemble min, is selected. For continuous action spaces, the plans are efficiently calculated online with gradient-based optimisation (see §2.3.3) *through the learned likelihood models.*

ditioned on a value for $\Theta = \theta$, there is no risk (i.e., aleatoric uncertainty) about the imitative score—likelihood under imitative model with parameters θ. As a consequence, the posterior over imitative scores $q(\mathbf{A} = \mathbf{a}|s;\Theta,\mathcal{D})$ captures only ignorance.

Since the imitative score is a continuous variable (regardless of the action space), the variance of the posterior distribution can be a signal for quantifying ignorance (Gal, 2016), i.e.,

$$\text{Var}_{\text{RIP}}(\mathbf{a}, s, p(\Theta|\mathcal{D})) \triangleq \text{Var}_\Theta[\log q(\mathbf{A} = \mathbf{a}|s;\Theta)p(\Theta|\mathcal{D})]. \tag{3.8}$$

Assuming well-calibrated estimates, in-distribution state→plan pairs have low posterior variance and OOD ones high, as shown in §3.3 and depicted in *Figure 3.9*.

To incorporate the ignorance estimates into the planning objective, we select a knowledge equivalent (KE, see §2.1.2), which is a reduction of the posterior distribution over imitative scores down to an "optimisable" scalar value, i.e., the RIP selected plan is given by:

$$\mathbf{a}_{\text{RIP}}(s, p(\Theta|\mathcal{D}); \text{KE}) \triangleq \arg\max_{\mathbf{a}} \text{KE}[\log q(\mathbf{A} = \mathbf{a}|s;\Theta)p(\Theta|\mathcal{D})]. \tag{3.9}$$

The choice of the KE boils down to the designer's preferences (i.e., a *hyperparameter*) and directly affects the induced behaviour of the agent. There are ignorance-neutral KEs that do not take into account the shape of the posterior distribution and capture only the central tendency, such as the "model average" (MA Savage, 1954), given by *Eqn.* (2.5).

There are also ignorance-sensitive KE for either seeking or avoiding ignorance. In this chapter, we aim to build an agent that is robust to distribution shifts and hence we adhere to the principle of "pessimism in the face of ignnorance" (Wald, 1939; Markowitz, 1952; Savage, 1954) and hence use an ignorance-averse KE. In particular, we use the "worst-case model" (WCM; Wald, 1939),[†] given by *Eqn.* (2.7a) and adapted for our setting, i.e.,

$$\mathbf{a}_{\text{RIP-WCM}}(s, p(\Theta|\mathcal{D})) \triangleq \arg\max_{\mathbf{a}} \min_{\theta \in \text{support}(p(\Theta|\mathcal{D}))} [\log q(\mathbf{A} = \mathbf{a}|s; \theta)]. \tag{3.10}$$

Intuition. An imitative model is trained such that it outputs high imitative scores for state-conditioned plans that are likely to have been made by the expert demonstrator and low scores otherwise. We expect (and provide empirical evidence §3.3) that optimising against the learned imitative model, i.e., naively searching for the plan that maximise the imitative score, is susceptible to the overoptimisation problem (see §2.3.4). In particular, the model-generated imitative scores for OOD state→plan pairs can be arbitrarily wrongly estimates and hence they should not be trusted. To guard an agent from this pitfall, the agent should be equipped with mechanisms to: (i) detect OOD state→plan pairs; and (ii) treat them differently during planning, compared to in-distribution ones.

Ignorance quantification addresses the former since the ignorance estimate from a well-calibrated model is high (resp. low) in OOD (resp. in-distribution) model inputs, e.g., as measured by the variance of the posterior distribution of imitative models (§2.2.1; MacKay, 1992; Gal, 2016). KE-based planning provides a framework for the latter since it "re-assigns" new scores according to the posterior distribution over imitative scores and a decision-theoretic rule (§2.1.2; Wald, 1939; Savage, 1954; Kochenderfer et al., 2022).

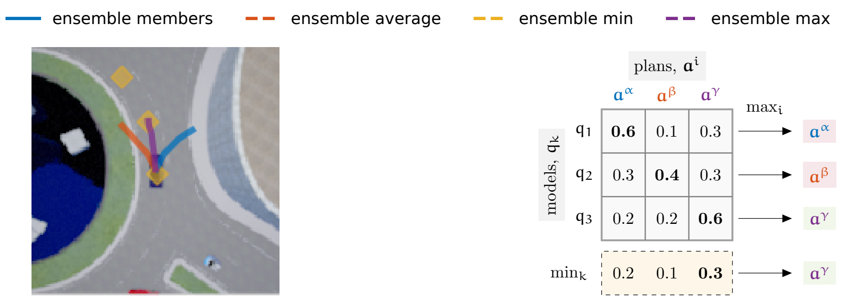
Any KE that assigns a low score to OOD plans fits our need for robustness to distribution shifts. The chosen "worst-case model" (WCM) KE in *Eqn.* (3.10) is ignorance-averse and hence satisfy the requirement. It assigns a pessimistic score to each action, equal to the lowest imitative score across all the *probable* imitative models under the posterior distribution. The in-distribution plans are reliably scored since only the imitative models that fit them accurately are in the support[‡] and all OOD plans get assigned low scores since with high probability there is a model in the posterior that assigns them low scores.

Example 3.1 (didactic example for robust imitative planning (RIP) in out-of-training distribution (OOD) driving scenario)***:*** Consider the OOD driving scenario in *Figure 3.6a*. To simplify the presentation, the agent selects between the plans $\mathbf{a}^{\alpha}, \mathbf{a}^{\beta}, \mathbf{a}^{\gamma}$, according to an ensemble of imitative models q_1, q_2, q_3. We collect the imitative scores in *Figure 3.6b*, where each column has the per-model scores. In this

[†]The WCM KE is incompatible with, by design, full-support posteriors over model parameters since it's over-pessimistic. Alternative, "softer" ignorance-averse KEs can be used instead, including the conditional value at risk (CVaR, Embrechts et al., 2013; Rajeswaran et al., 2016) that employs quantiles.

[‡]Assuming that the probability under the posterior for a bad fit is exactly 0.

visualisation, planning (see *Eqns.* (3.5, 3.9)) is equivalent to calculating the per-row arg max. Under models q_1 and q_2 the selected plans are the catastrophic trajectories $\mathbf{a}^{\alpha}$ and $\mathbf{a}^{\beta}$, respectively. In the last row in yellow, we compute the "worst-case model" (WCM) imitative scores, i.e., per-column minimum, given by *Eqn.* (3.10). Planning against the WCM knowledge equivalent (KE) proposes the safe plan $\mathbf{a}^{\gamma}$.


ensemble members
ensemble average
ensemble min
ensemble max
plans, $\mathbf{a}^i$
$\mathbf{a}^{\alpha}$ $\mathbf{a}^{\beta}$ $\mathbf{a}^{\gamma}$
$\max_i$
models, q_k
q_1 0.6 0.1 0.3 → $\mathbf{a}^{\alpha}$
q_2 0.3 0.4 0.3 → $\mathbf{a}^{\beta}$
q_3 0.2 0.2 0.6 → $\mathbf{a}^{\gamma}$
$\min_k$ 0.2 0.1 0.3 → $\mathbf{a}^{\gamma}$

(a) OOD scenario ***(b)*** Robust imitative planning (RIP)

Figure 3.6: Didactic example of robust imitative planning (RIP) agent in an out-of-training distribution (OOD) driving setting. ***(a)*** The bird-eye view of the scene and the candidate plans, i.e., $\mathbf{a}^{\alpha}, \mathbf{a}^{\beta}, \mathbf{a}^{\gamma}$. ***(b)*** The imitative scores for ensemble of imitative models q_1, q_2, q_3 and the actions each model would select and the RIP selected action under the "worst-case model" (WCM) knowledge equivalent (KE).

Practical implementation. In practice, it is intractable to perform exact Bayesian inference (see §2.2.1) for the parameters of the imitative model (Neal, 1995). Instead, we adopt approximate ignorance quantification strategies, such as deep ensembles (see §2.2.3; Lakshminarayanan et al., 2017). We consider an ensemble of K imitative models $\{q(\cdot|s; \theta_k)\}_{k=1}^{K}$, as given in §3.3, trained according to the guidelines in *Remark* 2.7 to improve the ignorance estimation, namely: (i) maximum a posteriori (MAP) inference, with a normally distributed prior over the parameters (i.e., L2 regularisation term); (ii) randomised parameter initialisation per ensemble member (Lakshminarayanan et al., 2017; Izmailov et al., 2021); and (iii) data bootstrapping (Osband et al., 2016). However, any (approximate) inference method to approximate or sample from the posterior $p(\Theta|\mathcal{D})$ would suffice. To demonstrate that our method is agnostic to deep ensembles, we use Monte Carlo (MC) dropout (Gal and Ghahramani, 2016) for some experiments in §3.3.2.

Then, we re-write *Eqns.* (3.8, 3.10) using the parameters of the deep ensemble $\{\theta_k\}_{k=1}^{K}$:

$$\mathrm{Var}_{\mathrm{RIP}}(\mathbf{a}, s, \{\theta_k\}_{k=1}^{K}) \triangleq \mathrm{Var}[\{\log q(\mathbf{A} = \mathbf{a}|s; \theta_k)\}_{k=1}^{K}] \tag{3.11a}$$

$$\mathbf{a}_{\text{RIP-WCM}}(s, \{\theta_k\}_{k=1}^{K}) \triangleq \arg\max_{\mathbf{a}} \min_{k} [\log q(\mathbf{A} = \mathbf{a}|s; \theta_k)], \tag{3.11b}$$

where *Eqn.* (3.11a) refers to the empirical variance across the ensemble scores. As discussed in §3.1, to solve *Eqn.* (3.11b), we either use online gradient-ascent through the learned ensemble of imitative models or search over a (pre-selected) trajectory library.

3.2.2 Adaptive robust imitative planning (AdaRIP)

In this section, we present an adaptive variant of RIP, called *adaptive robust imitative planning* (AdaRIP), which learns online from expert demonstrations that it selects to gather. It leverages its ignorance estimates to decide when to query the expert for feedback, avoiding this way potentially unsafe decisions and learning from this feedback. AdaRIP's pseudocode and the main differences with RIP are given in Algorithm 1.

Formalisation. AdaRIP has two learning phases: (i) offline learning (i.e., pre-training) from static expert demonstrations $\mathcal{D}$; and (ii) online learning (i.e., fine-tuning) from actively collected demonstrations $\mathcal{B}$. The former is identical to RIP's learning algorithm in §3.2.1 and the latter is specific to AdaRIP, during its interaction with the environment.

In particular, it is assumed that the agent has black-box access to an expert policy (i.e., *oracle*), π^*,[†] which it can choose to query at any state $s \in \mathbb{S}$ and receive an expert plan, i.e., $\mathbf{a}^* \sim \pi^*(\cdot|s)$. Similar to DAgger (Ross et al., 2011) and its variants (Zhang and Cho, 2016; Cronrath et al., 2018), the access to the oracle is limited and hence a mechanism to ensure sample-efficient use of it is required. To that end, AdaRIP queries the oracle on (believed) OOD states, i.e., states with high estimated posterior variance of imitative scores.

Specifically, for each state s: (i) we compute the RIP plan $\hat{\mathbf{a}}_{\text{RIP}}$ according to *Eqn.* (3.10); (ii) we estimate the posterior variance of the imitative score $\hat{\sigma}^2_{\text{RIP}}$ for the $(s, \hat{\mathbf{a}}_{\text{RIP}})$ pair; and (iii) if $\hat{\sigma}^2_{\text{RIP}} > \tau$, where $\tau \in \mathbb{R}^+$ is a pre-specified, designer-selected (i.e., hyperparameter) threshold, then (iv) we query the oracle at s, receive $\mathbf{a}^*$ and append it to the online demonstrations dataset $\mathcal{B}$. Finally, after observing $\mathcal{B}$, we update (in an online fashion) the posterior over imitative model parameters, obtaining $p(\Theta|\{\mathcal{D}, \mathcal{B}\})$, and then repeat.

Intuition. As *Figure 3.1* and the empirical evidence in §3.3.2 suggest, planning with an ignorance-averse KE against a static learned model is not always sufficient to recover from distribution shifts, especially domain shifts as the one in *Figure 3.1a*. Even with perfect ignorance estimates, the RIP agent knows that it does not know what to do but this is not enough to extract a good course of actions, i.e., ignorance quantification does not "magically" solve the problem of generalisation. If an online feedback mechanism is in place, the information that the agent should not trust its plans due to ignorance, is actionable. For instance, in a human-in-the-loop setup (Ross et al., 2011; Christiano et al., 2017), the agent can defer from making a decision under (extreme) ignorance and, instead, ask the human to act on its behalf. Therefore, the agent both avoids taking a potentially unsafe action that could not be tolerated in safety-critical settings, e.g., autonomous driving, and can learn from the demonstrated behaviour online (Duan et al., 2017), reducing its ignorance and allowing it cope with similar situations encountered later. Of course, there is no extra value added in querying the human in situations with low ignorance

[†]AdaRIP is also compatible with other feedback mechanisms, such as expert preferences (Christiano et al., 2017) or explicit reward functions (de Haan et al., 2019).

since that should[†] mean that the imitative model is already trained on similar situations.

Before taking an action, the AdaRIP agent: (i) plans according to the RIP strategy in *Eqn.* (3.10); and (ii) decides if its confidence in the plan is above some threshold and if so, it commits to it. Otherwise it invokes the online feedback mechanism. The acquired feedback drives the online model adaptation/fine-tuning.

Practical implementation. We build on the RIP's practical implementation. The learning and acing loops of the AdaRIP agent are described in pseudocode in Algorithm 1. We tune the threshold, τ, by performing an analysis similar to the one in *Figure 3.9*, see §3.3.1 for more details. For online fine-tuning with the $\mathcal{B}$ demonstrations, for each stochastic gradient descent (SGD) update, we have a mini-batch of mixed expert demonstrations—50% from the online demonstrations $\mathcal{B}$ and 50% from the offline demonstrations $\mathcal{D}$. Note that similar most methods for learning online/continually, our fine-tuning approach suffers from, e.g., catastrophic forgetting (French, 1999) and sample-inefficiency (Finn et al., 2017). In this chapter, our goal is only to demonstrate AdaRIP's efficacy to adapt under distribution shifts and hence we do not address either of these limitations. Future work lies in providing a practical, sample-efficient continual learning algorithm to be used in conjunction with the AdaRIP agent. Methods for efficient (e.g., few-shot or zero-shot) and safe adaptation (Finn et al., 2017; Zhou et al., 2019) are orthogonal to AdaRIP and hence any improvement in these fields could be directly used for AdaRIP.

3.3 Experiments

We consider a series of prediction and control experiments—with and without online expert feedback—to determine how effective is ignorance quantification and ignorance-aware planning when learning from expert demonstrations and acting under distribution shifts. We focus on autonomous driving since (i) there are open source real-world expert demonstrations (Caesar et al., 2019; Sun et al., 2020); (ii) realistic simulators (e.g., `CARLA`, Dosovitskiy et al., 2017); and (iii) current state-of-the-art imitation learning (IL) methods are benchmarked in such domains (Chen et al., 2019; Rhinehart et al., 2020).

Autonomous driving specific assumptions. For a fair comparison to the baselines, we make a few (benign) autonomous driving specific assumptions. In particular, the agent plans over sequences of relative (egocentric) (x, y) coordinates rather than low level actions, e.g., steering angle, throttle pressure, which allows it to be invariant to the physical properties of the controlled vehicle and allows us to use Additionally, for the online experiments it is assumed that high level goals are provided, e.g., "turn left at the traffic light", "at the roundabout take the second exit", "at the end of the road make a U turn" and the agent can factor them into its planning procedure. Formally:

[†]This does not necessarily hold true when we use in practice imperfect ignorance estimates.

Algorithm 1: Adaptive robust imitative planning (AdaRIP)

```
Input :
    $\mathcal{D}$  Expert demonstrations              $q_\theta$  Imitative model architecture
    $K$  Number of ensemble members                   $p(\Theta)$  Model parameters prior
    $\mathcal{B}$  Online demonstrations buffer       $\tau$  Variance threshold
Output :
    $\{\theta_k\}_{k=1}^{K}$  Parameters of ensemble members

  // Learning offline from expert demonstrations
1 for $k = 1 \dots K$ do
2     Bootstrap sample dataset $\mathcal{D}_k \overset{\text{boot}}{\sim} \mathcal{D}$
3     Sample model parameters from prior, $\theta_k \sim p(\Theta)$
4     Learn model with maximum a posteriori (MAP) with regulariser $\Omega$

          $\theta_k \leftarrow \arg\max_{\theta} \mathbb{E}_{(s,\mathbf{a})\sim\mathcal{D}_k} [\log q(\mathbf{A} = \mathbf{a}|s;\theta)] + \Omega(\theta)$    ▷ see Eqn. (2.16)

  // Agent-environment online interaction
5 $s \leftarrow$ env.reset()
6 while not done do
7     Plan with learned ignorance-aware model from state $s$

          $\hat{\mathbf{a}}_{\text{RIP}} \leftarrow \arg\max_{\mathbf{a}} \min_{k} [\log q(\mathbf{A} = \mathbf{a}|s;\theta_k)]$    ▷ see Eqn. (3.10)

      // Online adaptation
8     Calculate the predictive variance of the plan

          $\hat{\sigma}^2_{\text{RIP}} \leftarrow \text{Var}[\{\log q(\mathbf{A} = \hat{\mathbf{a}}_{\text{RIP}}|s;\theta_k)\}_{k=1}^{K}]$    ▷ see Eqn. (3.8)

      if $\hat{\sigma}^2_{RIP} > \tau$ then
9         $\mathbf{a}^* \leftarrow$ Query expert at $s$
10        $\mathcal{B} \leftarrow \mathcal{B} \cup (s, \mathbf{a}^*)$
11        Perform parameter update using online and offline demonstrations $\mathcal{B}$, $\mathcal{D}$
12    $s$, done $\leftarrow$ env.step($\mathbf{a}^*$)
```

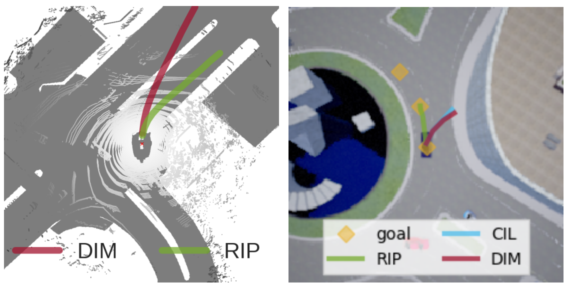
Assumption 3.1 (inverse dynamics)***:*** We assume that the physical properties of the controlled vehicle ϕ, e.g., mass, dimensions and minimum/maximum acceleration, are known and hence a low-level controller c that makes use of this information, e.g., proportional–integral–derivative controller (PID, Zhou et al., 1996) can be used for (deterministically) mapping "high-level" plans $\mathbf{a}$ in relative (x, y) coordinates to low-level controls $\mathbf{u}$, e.g., steering, accelerating and breaking:

$$\mathbf{u} = c(\mathbf{a}; \phi). \tag{3.12}$$

Assumption 3.2 (global planner)***:*** We assume access to a global navigation system that we can use to specify goals $g \in \mathcal{G}$ is the form of (x, y) coordinates or navigation command, such as "turn left/right" "at the roundabout take the second exit" etc.

These are benign assumptions for autonomous driving (Rhinehart et al., 2020). If required, these quantities can also be learned from data, and are typically easier to learn than π^*.

Environments. For the offline prediction experiments, we use the `nuScenes` open source dataset (Caesar et al., 2019) for autonomous driving, collected by human expert drivers. To assess agents in this setting, we use the metrics from the ICRA 2020 `nuScenes` prediction challenge. For the online control[†] experiments, we use the `CARLA` autonomous driving simulator (Dosovitskiy et al., 2017). In particular, to make sure that agents experience challenging OOD driving scenarios in this environment and are evaluated on their robustness to distribution shifts, we introduce a benchmark, called `CARNOVEL`. Offline expert demonstrations[‡] from `Town01` are provided for training. Then, the agents are evaluated on a suite of OOD navigation tasks, including but not limited to roundabouts, challenging non-right-angled turns and hills, none of which are experienced during training since these are from other `CARLA` towns, i.e., `Town03`, `Town04` and `Town05`. The complete `CARNOVEL` suite of tasks are given in §A.1 and a handful of example are depicted in *Figure 3.8*



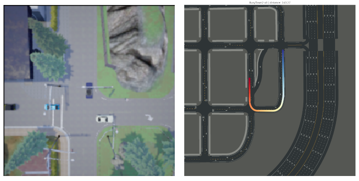
(a) `nuScenes` *(b)* `CARNOVEL`

Figure 3.7: RIP's robustness to OOD scenarios, compared to CIL (Codevilla et al., 2018) and DIM (Rhinehart et al., 2020). Both baselines get out of the road and hit the barriers while RIP follows a safe trajectory.

[†]We use the terms "prediction" and "control" as in (Sutton and Barto, 2018).

[‡]We use the `CARLA` rule-based autopilot (Dosovitskiy et al., 2017) as the expert demonstrator.

(a) `AbnormalTurns4-v0`


(b) `BusyTown2-v0`

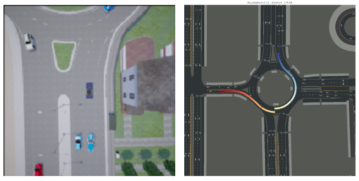

(c) `Roundabouts1-v0`

Figure 3.8: The spawn location and the route for completing example `CARNOVEL` task.

Table 3.1: Robust imitative planning (RIP) variants used for the ablation study.

Variants	**Description**
DIM	Ignorance-unaware, i.e., point estimate of imitative model (Rhinehart et al., 2020).
RIP-BCM	Ignorance-seeking KE, i.e., $\max_{k}$, optimism in the face of ignorance
RIP-MA	Ignorance-neutral KE, i.e., $\mathbb{E}_{k}$, ensemble average
RIP-WCM-TL	Trajectory library-based (Liu and Atkeson, 2009) variant of RIP-WCM, see *Eqn.* (4.2)

Metrics. Since we are studying navigation tasks, agents should be able to reach safely pre-specified destinations. As also done in previous work (Codevilla et al., 2018; Rhinehart et al., 2020; Chen et al., 2019), the *infractions per kilometre* metric (i.e., violations of rules of the road and accidents per kilometre traveled) measures how safe the agent navigates. The *success rate* measures the percentage of successful navigations to the destination, without any infraction. However, these standard metrics do not directly reflect the methods' performance under distribution shifts. As a result, we introduce three new metrics to quantify performance in out-of-training distribution (OOD) tasks:

1. **Detection score**: The correlation of infractions and model's uncertainty termed *detection score* is used to measure a method's ability to predict the OOD scenes that lead to catastrophic events. As discussed by Michelmore et al. (2018), we look at time windows of 4 seconds (Taoka, 1989; Coley et al., 2009). A method that can detect potential infractions should have high detection score.
2. **Recovery score**: The percentage of successful manoeuvres in novel scene—where the ignorance-unaware methods fail—is used to quantify a method's ability to recover from distribution shifts. A method that is oblivious to novelty should have 0 recovery score, but positive otherwise.
3. **Adaptation score**: The improvement in success rate as a function of number of online expert demonstrations is used to measure a method's ability to adapt efficiently online. A method that can adapt online should have a positive adaptation score.

Baselines. For both prediction and control experiments, we consider two types of baseline methods: (i) state-of-the-art methods in each setting; and (ii) variants of our RIP

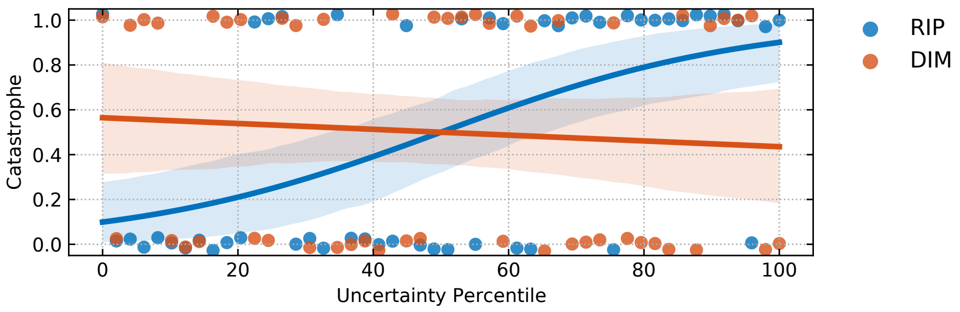


Figure 3.9: Uncertainty estimators as indicators of catastrophes on `CARNOVEL`. We collect 50 scenes for each model that led to a crash, record the uncertainty 4 seconds (Taoka, 1989) before the accident and assert if the uncertainties can be used for detection. RIP's (ours) predictive variance (in blue, see *Eqn.* (3.8)) serves as a useful detector, while DIM's (Rhinehart et al., 2020) negative log-likelihood (in orange) cannot be used for detecting catastrophes. We assume that the main reason for catastrophes is distribution shifts.

agent to ablate the importance of our design decisions, summarised in *Table 3.1*.[†]

For the prediction experiments, we compare against the state-of-the-art methods in the `nuScenes` dataset, the: (i) Multiple-Trajectory Prediction (**MTP**, Cui et al., 2019); (ii) **MultiPath** (Chai et al., 2019); and (iii) **CoverNet** (Phan-Minh et al., 2019), all of which score a (fixed) set of trajectories, i.e., a trajectory library (Liu and Atkeson, 2009).

For the control experiments, we compare against the state-of-the-art methods in the `CARLA` autonomous driving simulator, the: (i) conditional imitation learning (**CIL**, Codevilla et al., 2018), which is a discriminative behavioural cloning method that conditions its predictions on contextual information (e.g., LIDAR) and high-level commands (e.g., "turn left", "go straight"); (ii) learning by cheating (**LbC**, Chen et al., 2019), which is a method that builds on CIL and uses (cross-modal) distillation of privileged information (e.g., game state, rich, annotated bird-eye-view observations) to a sensorimotor agent; and (iii) LbC agent that uses *privileged* information directly (i.e., teacher), which we term **LbC-GT** and is only for reference since it is not a fair comparison against the LIDAR-based methods.

3.3.1 Detecting distribution shifts

We have argued for the role of ignorance in detecting distribution shifts in §2.1.2, which motivated the design of our RIP agent in this chapter. Therefore, we expect the following hypotheses to hold, which we test empirically. **H1**: Risk (i.e., aleatoric uncertainty) quantification cannot be used for detecting distribution shifts since it aims to capture the inherent stochasticity of the expert demonstrators' behaviour rather than novelty. **H2**: Ignorance (i.e., epistemic uncertainty) is an effective signal for detecting distribution shifts.

[†]We treat deep imitative model (**DIM**, Rhinehart et al., 2020) as a variant of RIP as discussed in §3.2.1.

Control. We benchmark in `CARNOVEL` one ignorance-unaware baseline method, DIM (Rhinehart et al., 2020) and our ignorance-aware RIP. We collect 100 scenes, recording estimates for their risk and ignorance, respectively. In 50 of them, after 4 seconds the agent crashed (Taoka, 1989) and in the other 50, the agents successfully completed the tasks. We assert if the recorded uncertainty estimates can be used for detecting that a catastrophe is about to occur—a proxy for whether the agent is experiencing a distribution shift. The results are illustrated in *Figure 3.9*. RIP's (ours) predictive variance *Eqn.* (3.8)—implemented as the ensemble variance—serves as a useful distribution shift detector, i.e., with high probability above the 80% percantile a crash occurs, while DIM's negative log-likelihood is not predictive of whether a crash is about to take place. These results support hypotheses **H2** and **H1**, respectively.

Having verified that ignorance is a useful signal for detecting distribution shifts, next, we study ways to integrate ignorance estimates into planning and assess their robustness to (i.e., ability to recover from) distribution shifts in both prediction and control settings.

3.3.2 Recovering from distribution shifts

The main motivation behind the ignorance-averse knowledge equivalent (KE) used in RIP is to enable robustness and recovery from distribution shift, implementing the principle of "pessimism in the face of ignorance". To this end, we investigate the following hypotheses. **H3**: Ignorance-aware agents perform qualitatively and quantitatively different in OOD scenarios from ignorance-unaware ones. **H4**: RIP's explicit mechanism for recovery from distribution shifts leads to improved performance in OOD scenes. **H5**: The role and choice of the knowledge equivalent (KE) is non-trivial—RIP is *not* "just an ensemble".

Prediction. *Table 3.2* gives the performance of the RIP agent, its variants and state-of-the-art methods in the `nuScenes` challenge dataset. In more detail, we use the provided train–val–test splits from (Phan-Minh et al., 2019), for towns `Boston` and `Singapore`. For all methods we use $\mathsf{N} = 50$ trajectories, and in case of both DIM and RIP, we only optimise the "imitation prior" *Eqn.* (3.9)), since goals are not provided, running N planning procedures with different random initialisations. We observe that the ignorance-aware agents, i.e., RIP variants that take ignorance into account by optimising a KE, perform consistently better than the ignorance-unaware baselines, backing up **H3**. Also, the ignorance-averse agents, i.e., RIP-WCM with continuous and RIP-WCM-TL with discretised actions, outperform the ignorance- oblivious and seeking methods, confirming **H4** and **H5**.

Control. *Table 3.3* summarises the results in the `CARNOVEL` autonomous driving control benchmark. All methods are trained with offline expert demonstrations from `CARLA Town01`. We perform 10 trials per `CARNOVEL` task with randomised initial simulator state (same for all agents). The ignorance-aware agents, i.e., RIP-WCM and RIP-MA, outperform the ignorance-unaware and ignorance-seeking methods, corroborating

Table 3.2: We evaluate different autonomous driving prediction methods in terms of their robustness to and recovery from distribution shifts from the `nuScenes` ICRA 2020 challenge (Phan-Minh et al., 2019). We use the provided train–val–test splits and report performance on the test (i.e., out-of-sample) scenarios. A "$\clubsuit$" indicates methods that use LIDAR observation, as in (Rhinehart et al., 2019), and a "$\diamond$" methods that use bird-view privileged information, as in (Phan-Minh et al., 2019). A "$\star$" indicates that we used the results from the original paper, otherwise we used our implementation. Standard errors are in grey (via bootstrap sampling). The **outperforming** method is in bold.

	`Boston`			`Singapore`		
	minADE$_1$ ↓	minADE$_5$ ↓	minFDE$_1$ ↓	minADE$_1$ ↓	minADE$_5$ ↓	minFDE$_1$ ↓
Methods	(2073 scenes, 50 samples, open-loop plans)			(1189 scenes, 50 samples, open-loop plans)		
MTP$^{\diamond\star}$ (Cui et al., 2019)	4.13	3.24	9.23	4.13	3.24	9.23
MultiPath$^{\diamond\star}$ (Chai et al., 2019)	3.89	3.34	9.19	3.89	3.34	9.19
CoverNet$^{\diamond\star}$ (Phan-Minh et al., 2019)	3.87	2.41	9.26	3.87	2.41	9.26
DIM$^{\clubsuit}$ (Rhinehart et al., 2020)	3.64±0.05	2.48±0.02	8.22±0.13	3.82±0.04	2.95±0.01	8.91±0.08
RIP-BCM$^{\clubsuit}$ (ablation, *Table 3.1*)	3.53±0.04	2.37±0.01	7.92±0.09	3.57±0.02	2.70±0.01	8.39±0.03
RIP-MA$^{\clubsuit}$ (ablation, *Table 3.1*)	3.39±0.03	2.33±0.01	7.62±0.07	3.48±0.01	2.69±0.02	8.19±0.02
RIP-WCM-TL$^{\clubsuit}$ (ablation, *Table 3.1*)	**3.31**±0.03	**2.32**±0.00	**7.49**±0.05	**3.44**±0.01	**2.69**±0.01	**8.10**±0.04
RIP-WCM$^{\clubsuit}$ (ours, §3.2.1)	**3.29**±0.03	**2.28**±0.00	**7.45**±0.05	**3.43**±0.01	**2.66**±0.01	**8.09**±0.04

H3 and **H5**, respectively. On top of the results in *Table 3.3*, the illustrated examples in *Figure 3.7* support **H4**.

Despite RIP's improvement over current state-of-the-art methods with 97.5% success rate and 0.26 infractions per driven kilometre, the safety-critical nature of the task mandates higher performance. Towards this goal, next, we assess AdaRIP's, the online adaptive variant of RIP in a few-shot imitation learning setting, where the agent decides when to query the expert.

3.3.3 Adapting to distribution shifts

In the previous sections, we have shown that ignorance is a useful signal for detecting distribution shifts (**H2**) and predictive of catastrophic outcomes (see *Figure 3.9*). When online expert demonstrations are available, AdaRIP uses its quantified ignorance to efficiently query the expert to avoid potential catastrophes. As a result, we expect the following hypothesis to be true, which we investigate experimentally. **H6**: With more online expert demonstrations, AdaRIP's performance improves.

Control. We evaluate AdaRIP on `CARNOVEL` tasks, where the `CARLA` autopilot (Dosovitskiy et al., 2017) is queried for demonstrations online when the predictive variance (see *Eqn.* (3.8)) exceeds a threshold, chosen according to RIP's detection score, (see *Figure 3.9*) in a hold-out dataset. AdaRIP's performance on the most challenging `CARNOVEL` tasks is summarised in *Figure 3.10*, where, the success rate improves as the

Table 3.3: We evaluate different autonomous driving methods in terms of their robustness to and recovery from distribution shifts, in our new benchmark, `CARNOVEL`. All methods are trained on `CARLA Town01` using imitation learning with expert demonstrations from the `CARLA` autopilot (Dosovitskiy et al., 2017). A "$*$" indicates methods that use first-person camera view, as in (Chen et al., 2019), a "♣" methods that use LIDAR observation, as in (Rhinehart et al., 2020) and a "◇" methods that use the ground truth game engine state, as in (Chen et al., 2019). A "★" indicates that we used the reference implementation from the original paper, otherwise we used our implementation. For all the scenes we chose pairs of start-destination locations and ran 10 trials with randomised initial simulator state for each pair (same for all methods). Standard errors are in grey (via bootstrap sampling). The **outperforming** method is in bold.

	`AbnormalTurns`			`BusyTown`		
Methods	Success ↑ (7 × 10 scenes, %)	Infra/km ↓ (×1e−3)	Distance ↑ (m)	Success ↑ (11 × 10 scenes, %)	Infra/km ↓ (×1e−3)	Distance ↑ (m)
CIL♣★ (Codevilla et al., 2018)	65.71±07.37	07.04±05.07	128±020	05.45±06.35	11.49±03.66	217±033
LbC$*$★ (Chen et al., 2019)	00.00±00.00	05.81±00.58	208±004	20.00±13.48	03.96±00.15	374±016
LbC-GT◇★ (Chen et al., 2019)	02.86±06.39	**03.68**±00.34	217±033	65.45±07.60	02.59±00.02	400±006
DIM♣ (Rhinehart et al., 2020)	74.28±11.26	05.56±04.06	108±017	47.13±14.54	08.47±05.22	175±026
RIP-BCM♣ (ablation, *Table 3.1*)	68.57±09.03	07.93±03.73	096±017	50.90±20.64	03.74±05.52	175±031
RIP-MA♣ (ablation, *Table 3.1*)	**84.28**±14.20	07.86±05.70	102±015	**64.54**±**23.25**	05.86±03.99	170±033
RIP-WCM♣ (ours, §3.2.1)	**87.14**±14.20	**04.91**±03.60	102±021	**62.72**±05.16	**03.17**±02.04	167±021

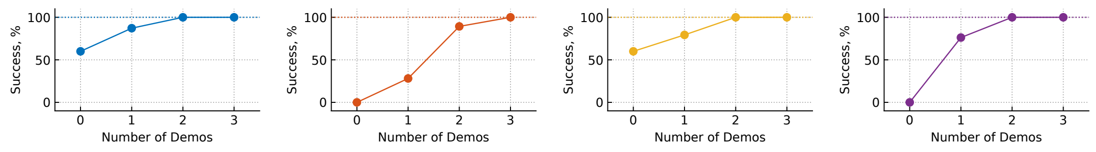
	`Hills`			`Roundabouts`		
Methods	Success ↑ (4 × 10 scenes, %)	Infra/km ↓ (×1e−3)	Distance ↑ (m)	Success ↑ (5 × 10 scenes, %)	Infra/km ↓ (×1e−3)	Distance ↑ (m)
CIL♣★ (Codevilla et al., 2018)	60.00±29.34	04.74±03.02	219±034	20.00±00.00	**03.60**±03.23	269±021
LbC$*$★ (Chen et al., 2019)	50.00±00.00	01.61±00.15	541±101	08.00±10.95	03.70±00.72	323±043
LbC-GT◇★ (Chen et al., 2019)	05.00±11.18	03.36±00.26	312±020	00.00±00.00	06.47±00.99	123±018
DIM♣ (Rhinehart et al., 2020)	70.00±10.54	06.87±04.09	195±012	20.00±09.42	06.19±04.73	240±044
RIP-BCM♣ (ablation, *Table 3.1*)	75.00±00.00	05.49±04.03	191±013	06.00±09.66	06.78±07.05	251±027
RIP-MA♣ (ablation, *Table 3.1*)	**97.50**±07.90	**00.26**±00.54	196±013	**38.00**±06.32	05.48±05.56	271±047
RIP-WCM♣ (ours, §3.2.1)	**87.50**±13.17	**01.83**±01.73	191±006	**42.00**±06.32	04.32±01.91	217±030

number of online demonstrations increases, validating **H6**. We also illustrate in §A.2 pre- and post- adaptation plans executed by AdaRIP.

3.4 Related work

The literature review for imitation learning (IL), knowledge equivalents (KEs), ignorance quantification in deep learning (DL) and deep ensembles can be found in §2. In this section, we review autonomous driving-specific related work to better contextualise our contributions in this domain, since our experiments are solely in it.

Learning from expert autonomous driving demonstrations. A plethora of expert driving demonstrations has been used for IL (Caesar et al., 2019; Sun et al., 2019;



(a) `AbnormalTurns4-v0` *(b)* `BusyTown2-v0` *(c)* `Hills1-v0` *(d)* `Roundabouts1-v0`

Figure 3.10: Adaptation scores of AdaRIP (see §3.2.2) on `CARNOVEL` tasks that RIP-WCM and RIP-MA (see §3.2.1) do worst. We observe that as the number of online expert demonstrations increases, the success rate improves thanks to online model adaptation.

Kesten et al., 2019) since a model mimicking expert demonstrations can simply learn to stay in "safe", expert-like parts of the state space and no explicit reward function need be specified. On the one hand, behaviour cloning (BC) approaches (Liang et al., 2018; Sauer et al., 2018; Li et al., 2018; Codevilla et al., 2018, 2019; Chen et al., 2019) fit command-conditioned discriminative sequential models to expert demonstrations, which are used in deployment to produce expert-like trajectories. On the other hand, Rhinehart et al. (2020) proposed command-*unconditioned* expert trajectory density models which are used for planning trajectories that both satisfy the goal constraints and are likely under the expert model. However, both of these approaches fit point-estimates to their parameters, thus do not quantify their ignorance (see *Remark* 2.6), as explained next. This is especially problematic when estimating what an expert would or would not do in *unfamiliar*, OOD scenes. In contrast, our methods, RIP and AdaRIP, does quantify ignorance in order to both improve planning performance (see §3.3.2) and triage situations in which an expert should intervene (see §3.3.3).

Current autonomous driving benchmarks. We are interested in the control problem, where autonomous driving agents get deployed in reactive environments and make sequential decisions. The `CARLA` Challenge (Ros et al., 2019; Dosovitskiy et al., 2017; Codevilla et al., 2019) is an open-source benchmark for control in autonomous driving. It is based on 10 traffic scenarios from the NHTSA pre-crash typology (National Highway Traffic Safety Administration, 2007) to inject challenging driving situations into traffic patterns encountered by autonomous driving agents. The methods are only assessed in terms of their generalization to weather conditions, the initial state of the simulation (e.g., the start and goal locations, and the random seed of other agents.) and the traffic density (i.e., empty town, regular traffic and dense traffic).

Despite these challenging scenarios selected in the `CARLA` Challenge, the agents are allowed to train on the same scenarios in which they evaluated, and so *the robustness to distributional shift is not assessed.* Consequently, both Chen et al. (2019) and Rhinehart et al. (2020) manage to solve the `CARLA` Challenge with almost 100% success rate, when trained in `Town01` and tested in `Town02`. However, both methods score *almost* 0% when evaluated in `Roundabouts` due to the presence of OOD road morphologies, as discussed in §3.3.

3.5 Conclusion & discussion

Summary of contributions. In this chapter, we studied imitation learning (IL) agents in out-of-training distribution (OOD) tasks (i.e., under distribution shifts). We introduced an ignorance-aware agent, called robust imitative planning (RIP), which can detect and recover from distribution shifts, as shown experimentally in a real-world prediction task, `nuScenes`, and a driving simulator, `CARLA`. We presented an adaptive variant, called adaptive robust imitative planning (AdaRIP), which uses RIP's ignorance estimates to efficiently query the expert for online feedback and adapt its model parameters online. We also introduced and open-sourced an autonomous driving control benchmark, termed `CARNOVEL`, to assess the robustness of driving agents to a suite of OOD tasks.

Insights & lessons learned. In our experiments in §3.3, we showed that ignorance (i.e., epistemic uncertainty) is an effective signal for distribution shifts detection, while risk (i.e., aleatoric uncertainty) is *not*, since it only captures the inherent stochasticity of the expert data. Moreover, ignorance-aware agents exhibit distinctively different behaviour in familiar and novel situations, while ignorance-unaware agents act the same, e.g., boldly and over-confidently, even when wrong. Also, optimising for a decision-theoretic ignorance-sensitive knowledge equivalent (KE), and the careful selection of it, enables ignorance-aware agents to avoid falling into their own pitfalls–overoptimisation.

Limitations & next steps. IL agents, although power, they miss out the opportunity to learn from sub-optimal demonstrations. This would enable them to massively scale the sources of data they can consume to improve (see §4). Moreover, the generic ignorance quantification strategy that we used for (Ada)RIP, i.e., deep ensembles, do not leverage the sequential nature of the problem setting. Future work lies in developing architectures and ignorance quantification algorithms that target sequential-decision making (see §5).

Outlook. Expert (human) demonstrations are one of the dominant data sources for training modern machine learning (ML) models and agents (Deng et al., 2009; Krizhevsky et al., 2012; Lin et al., 2014; Silver et al., 2016; Cordts et al., 2016; Vinyals et al., 2019; Radford et al., 2019; Sun et al., 2020). In an ever-changing environment as the real-world, any agent trained on this static data, despite its size, it is bound to experience novel, out-of-training distribution (OOD) scenarios and forced to make decisions under ignorance. Inspired by *Figure 3.1*, we argued that whether a learning agent experiences a distribution shift in deployment is not a matter of if, but when. As a consequence, the agent's designer should be developing the agent with this perspective in mind, not necessarily trying to anticipate the distribution shifts and accommodate these particular instances but instead acknowledge that there will be settings that the agent cannot take any meaningful decision due to its lack of knowledge (i.e., ignorance) and instead it should either act conservatively, i.e., avoid taking a leap to the unknown (as RIP does), or be equipped with a mechanism to use efficiently an "I don't know" action, in which case, some expert takes over (as AdaRIP does).

"Adaptability is not imitation. It means power of resistance and assimilation."

— Mahatma Gandhi (1869–1948)

4 Plan and adapt from sub-optimal demonstrations

Contents

Generally capable data-driven agents should be able to learn from any kind of data, making minimal assumptions about its generative process. The harder and more specialised the tasks we expect agents to fulfil, the fewer the chances we can find high quality expert demonstrations for training imitation learning (IL, see §2.2.4 and §3) agents. Also, reinforcement learning (RL, see §2.3) agents do not possess a mechanism for learning without reward information, limiting their scope to tasks for which a reward function can be specified (Russell, 2019), or off-policy rollouts (see §2.35) with reward samples.

Nonetheless, we often have access to *no-reward demonstrations*, i.e., data from the interaction of other agents with the environment of interest, without any reward information.

[*]This chapter is based on the (Filos et al., 2021) publication.

Our main modelling assumption in this chapter is that agents are typically goal-directed—sometimes by definition (Franklin and Graesser, 1996). While their goals can be different, they often depend on shared salient features of the environment, and may be able to interact with and affect the environment in similar ways. Humans and other animals make ready use of these similarities to other agents while learning (Henrich, 2017; Laland, 2018). We can observe the goal-directed behaviours of other humans, and combine these observations with our own experiences, to quickly learn how to achieve our own goals. If RL agents could similarly interpret the behaviour of others, they could learn more efficiently, relying less on solitary trial and error.

To this end, we formalise and address a problem setting in which an agent (the 'ego-agent') is given access to observations and actions drawn from the experiences of other goal-directed agents interacting with the same environment, but pursuing distinct goals. These observed trajectories are unlabelled in the sense that they lack the goals or rewards of the other agents, i.e., behavioural data of mixed and unknown quality. This type of data is readily available in many real-world settings, either from (i) observing other agents acting simultaneously with the ego-agent in the same environment (i.e., *social learning*); or (ii) multi-task demonstrations collected independently from the ego-agent's experiences. Consider autonomous driving as a motivating example: the robot car can observe the decisions of many nearby human drivers with various preferences and destinations, or may have access to a large offline dataset of such demonstrations. Because the other agents are pursuing their own varied goals, it can be difficult to directly use this information with conventional (single-task) IL (see §2.2.4), i.e., behaviour cloning (BC, Widrow, 1964) methods or inverse reinforcement learning (IRL, Ng et al., 2000; Ziebart et al., 2008).

To effectively use the no-reward demonstrations, we turn to the framework of successor features. This allows us to disentangle shared features and dynamics of the environment from agent-specific rewards and policies. We propose a multi-task inverse reinforcement learning (IRL) algorithm, called *inverse temporal difference* (ITD) *learning*, that learns shared state features, alongside per-agent successor features and preference vectors, purely from demonstrations without reward labels. We further show how to seamlessly integrate ITD with learning from online environment interactions, arriving at a novel algorithm for RL with demonstrations, called $\Psi\Phi$-*learning* ($\Psi\Phi$L, pronounced 'Sci-Fi').

We provide empirical evidence in a set of gridworld environments, a traffic-flow simulator (Leurent, 2018), and a task from the `Procgen` suite (Cobbe et al., 2020) that our proposed methods, i.e., ITD and $\Psi\Phi$L: (i) can scale to high-dimensional pixel observations; (ii) lead to faster from scratch and few-shot learning than vanilla RL (Mnih et al., 2015; Schulman et al., 2017), IL (Reddy et al., 2019; Ho and Ermon, 2016), and auxiliary-task baselines (Hernandez-Leal et al., 2017); and (iii) learn useful feature representations that are predictive of other agents' behaviour. The key contributions are:

List of contributions §4 (plan and adapt from sub-optimal demonstrations)***:***

1. **Offline multi-task IRL:** We propose a multi-task inverse reinforcement learning (IRL) algorithm, called *inverse temporal difference* (ITD) *learning.* Using only no-reward demonstrations, we learn shared state features, alongside per-agent successor features and inferred preferences. The reward functions can be straightforwardly computed from these learned quantities. We show empirically that ITD achieves superior or comparable performance to prior methods.
2. **RL with no-reward demonstrations:** By combining ITD with learning from environment interactions, we arrive at a novel algorithm for RL with unlabelled demonstrations, called ΨΦ*-learning* (ΨΦL, pronounced 'Sci-Fi'). ΨΦ-learning is compatible with sub-optimal demonstrations. It treats the demonstrated trajectories as being soft-optimal under *some* task and employs ITD to recover successor features for the demonstrators' policies. ΨΦ-learning inherits the unbiased, asymptotic performance of RL methods while leveraging the provided demonstrations with ITD. When the goals of any of the demonstrators are even partially aligned with the ΨΦ-learner, this enables much faster learning than solitary RL. Otherwise, when the demonstrations are not useful or even misleading, it gracefully falls back to standard RL, unlike naïve behaviour cloning or IRL.
3. **Few-shot adaptation with task inference:** Taking full-advantage of the successor features framework, our ΨΦ-learner can even adapt zero-shot to new goals it has never seen or experienced during training, but which are partially aligned with the demonstrated goals. This is possible due to the disentanglement of representations into task-specific features (i.e., preferences) and shared state features. We can efficiently update the task-specific preferences and rely on generalised policy improvement for safe policy updates. We derive worst-case bounds for the performance of ΨΦL in zero-shot transfer to new tasks.

4.1 Background & problem setting

We consider a world that can be represented as a controlled Markov process (CMP), i.e., $\mathcal{C} \triangleq \langle \mathbb{S}, \mathbb{A}, p, \rho_0 \rangle$, as formulated in §2.3.1. Then, as per *Eqn.* (2.27), a *task* is formulated as a Markov decision process (MDP, Puterman, 2014), characterised by a reward function, $r : \mathbb{S} \times \mathbb{A} \to \Delta(\mathbb{R})$, i.e., $\mathcal{M} \triangleq \langle \mathcal{C}, r \rangle$. As *Remark* 2.9 suggests, this formalisation allows us to treat tasks with the same dynamics (state, action spaces, initial state distribution and state-transition dynamics) but different reward functions in a unified notation.

We adapt the notation for the value functions (see *Eqns.* (2.29d, 2.31)), accordingly, i.e.,

$$v^{\pi,r}(s) \triangleq \mathbb{E}[\sum_{t \geqslant 0} \gamma^t R_{t+1} | S_0 = s, A_t \sim \pi(\cdot|S_t), (S_{t+1}, R_{t+1}) \sim m^*(\cdot, \cdot|S_t, A_t)] \quad (4.1a)$$

$$= \mathbb{E}[\sum_{t\geqslant 0} \gamma^t R_{t+1} | S_0 = s; \pi, \mathcal{C}] \tag{4.1b}$$

$$= \mathbb{E}[\sum_{t\geqslant 0} \gamma^t R_{t+1} | S_0 = s; \pi, \mathcal{C}] \tag{4.1c}$$

$$= \mathbb{E}_{\pi,\mathcal{C}}[\sum_{t\geqslant 0} \gamma^t R_{t+1} | S_0 = s] \tag{4.1d}$$

$$q^{\pi,r}(s,a) \triangleq \mathbb{E}_{\pi,\mathcal{C}}[\sum_{t\geqslant 0} \gamma^t R_{t+1} | S_0 = s, A_0 = a], \tag{4.1e}$$

where $\mathbb{E}_{\pi,\mathcal{C}}[\cdot]$ denotes expected value when following policy π in an environment $\mathcal{C}$.

Reinforcement learning with no-reward demonstrations. We are interested in settings in which, in addition to an environment $\mathcal{C}$, the agent also has access to *demonstrations without rewards.* The demonstrations are generated by other agents, whose goals and levels of expertise are unknown, and who have no incentive to educate the controlled agent. We will refer to the controlled agent, i.e., reinforcement learner, as the 'ego-agent' and to the agents that generated the demonstrations as 'other-agents'. We denote the demonstrations with $\mathcal{D} = \{\xi^{(1)}, \xi^{(2)}, \dots, \xi^{(N)}\}$, where the n-th trajectory is:

$$\xi^{(n)} \triangleq (s_0^{(n)}, a_0^{(n)}, \dots, s_T^{(n)}, a_T^{(n)}; k) \tag{4.2}$$

is generated by the k-th agent. Note that each trajectory does include an identifier of the agent that generated it. The ego-agent also gathers its own experience by interacting with the environment, collecting rollouts $\mathcal{B} = \{(s, a, s', r')\}$. Due to the lack of reward annotations in $\mathcal{D}$, and the fact that the data may be irrelevant to the ego-agent's task, it is not trivial to combine demonstrations from $\mathcal{D}$ with the ego-agent's experience $\mathcal{B}$.

Behaviour cloning as an auxiliary loss. The no-reward demonstrations $\mathcal{D}$ can be used as an auxiliary loss for representation learning (Hernandez-Leal et al., 2019). For instance, consider a Q-learning agent (§2.3.2; Watkins, 1989) with a representation network h_{θ_h} and action-value network q_{θ_q} as in *Figure 4.1* (Mnih et al., 2013, 2015). These two neural networks (NNs) are trained with the Q-learning loss (see *Eqn.* (2.44)) on the rollouts $\mathcal{B}$. On top of these, a policy network π_{θ_π} is trained with the behaviour cloning (BC) loss on the no-reard demonstrations $\mathcal{D}$. The gradients from the BC loss flow also through the representation network h_{θ_h} and hence we say that they shape the representation of the Q-learner, i.e.,

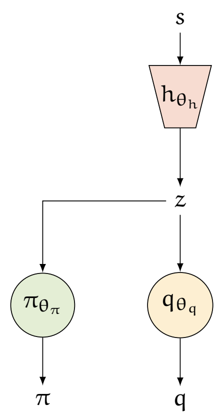


Figure 4.1: Q-learning with a behaviour cloning (BC) auxiliary loss.

$$\mathcal{L}_{\text{BC-AUX}}(\theta_h, \theta_\pi) \triangleq \mathbb{E}_{(s,a)\sim\mathcal{D}}[-\log \pi(A = a | h(s; \theta_h); \theta_\pi]. \tag{4.3}$$

Next, we review the framework of successor features, cumulants and generalised policy updates, which are the core building blocks for the methods we introduce in §4.2

Cumulants and preferences. The CMP-based formulation of tasks (see §4.1 and *Remark* 2.9) allows us to capture the notation that a set of tasks may differ in their reward function but they share the same environment. Consider $L \in \mathbb{N}$ tasks, with reward functions $\{r^l\}_{l=1}^L$, then the L tasks (i.e., MDPs) can be written in vector form, i.e.,

$$\begin{bmatrix} \mathcal{M}^1 \\ \mathcal{M}^2 \\ \vdots \\ \mathcal{M}^L \end{bmatrix} \overset{(2.27)}{=} \begin{bmatrix} \langle \mathcal{C}, r^1 \rangle \\ \langle \mathcal{C}, r^2 \rangle \\ \vdots \\ \langle \mathcal{C}, r^L \rangle \end{bmatrix} = \langle \mathcal{C}, \begin{bmatrix} r^1 \\ r^2 \\ \vdots \\ r^L \end{bmatrix} \rangle = \langle \mathcal{C}, \Phi^T \begin{bmatrix} \mathbf{w}^1 \\ \mathbf{w}^2 \\ \vdots \\ \mathbf{w}^L \end{bmatrix} \rangle, \tag{4.4}$$

where for the last equality we used the decomposition of the rewards in *cumulants* Φ and *preferences vectors* $\{\mathbf{w}^l\}_{l=1}^L$ as done by Dayan (1993); Barreto et al. (2017), i.e.,

$$r^{\mathbf{w}}(s, a) \triangleq \Phi(s, a)^\top \mathbf{w}, \tag{4.5}$$

where $\Phi(s, a), \mathbf{w} \in \mathbb{R}^d$. As *Eqn.* (4.4) suggests, the cumulants Φ are *task-agnostic* (i.e., shared among all L tasks) and the preferences are task-specific (i.e., one per task).

Remark 4.1 (cumulants and preference in layman's terms)***:*** We can think of cumulants as linear basis functions for the rewards, capturing shared salient features between the different tasks. The preference vectors are then the coefficients for mixing the different components of the (cumulants) basis to compose rewards functions.

Example 4.1 (didactic example for cumulants and preferences)***:*** For instance, consider a set of different tasks in the gridworld environment in *Figure 4.2*.

Let three 3 tasks, with rewards:

- r^1: 1 when entering a green cell, -1 when entering a red cell and 0 otherwise.
- r^2: 2 when entering a red cell, -1 when entering a yellow cell and 0 otherwise.
- r^3: 1 when entering a yellow cell, -3 when entering a green cell and 0 otherwise.

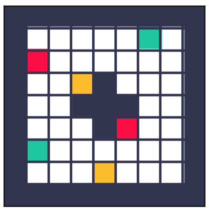


Figure 4.2: `CoinGrid`

The reward functions can be decomposed into cumulants:

$$\Phi = [r_{\text{green}}, r_{\text{red}}, r_{\text{yellow}}], \tag{4.6}$$

and then the corresponding preference vectors are:

$$\mathbf{w}^1 = [1, -1, 0], \quad \mathbf{w}^2 = [0, 2, -1], \quad \mathbf{w}^3 = [-3, 0, 1]. \tag{4.7}$$

The choice of decomposing the rewards into a dot product between a task-agnostic term (cumulants) and a task-specific (preferences vector) term, rather than any other non-linear decomposition, lies in the fact that we want exploit the linearity property of the value function (w.r.t. the reward function)[†] (Puterman, 2014) to speed up policy evaluation.

Successor features. Given *Eqn.* (4.5), the action-value of a policy π is given by:

$$q^{\pi,\mathbf{w}}(s, a) \triangleq \mathbb{E}_{\pi,\mathcal{C}}[\sum_{t\geqslant 0} \gamma^t r^{\mathbf{w}}(S_t, A_t) | S_0 = s, A_0 = a] \tag{4.8a}$$

$$\overset{(4.5)}{=} \mathbb{E}_{\pi,\mathcal{C}}[\sum_{t\geqslant} \gamma^t \Phi(S_t, A_t)^\top \mathbf{w} | S_0 = s, A_0 = a] \tag{4.8b}$$

$$= \mathbb{E}_{\pi,\mathcal{C}}[\sum_{t\geqslant 0} \gamma^t \Phi(S_t, A_t) | S_0 = s, A_0 = a]^\top \mathbf{w}. \tag{4.8c}$$

We refer to the red term in *Eqn.* (4.8c) as *successor features* (SFs, Barreto et al., 2017), i.e.,

$$\Psi^\pi(s, a) \triangleq \mathbb{E}_{\pi,\mathcal{C}}[\sum_{t\geqslant 0} \gamma^t \Phi(S_t, A_t) | S_0 = s, A_0 = a] \tag{4.9a}$$

$$q^{\pi,\mathbf{w}}(s, a) \overset{(4.8c)}{=} \Psi^\pi(s, a)^\top \mathbf{w}. \tag{4.9b}$$

Remark 4.2 (successor features (SFs) in layman's terms)***:*** SFs are to cumulants what value functions are to rewards. In other words, successor features (of the policy π) are the expected discounted sum of cumulants collected by unrolling policy π in an environment $\mathcal{C}$. In particular, the i-th component of $\Psi^\pi(s, a)$ gives the expected discounted sum of $\Phi(s, a)$'s i-th component, when starting from state s, taking action a and then following policy π. Intuitively, cumulants Φ can be seen as a vector-valued reward function (see *Eqn.* (4.4)) and SFs Ψ^π the corresponding vector-valued state-action value function for policy π.

Similar to *Eqn.* (2.40) for the value functions, the SFs can be recursively defined:

[†]The value in *Eqn.* (2.29d) is the expectation (linear map) of a weighted sum of rewards (linear maps).

$$\begin{aligned}
\Psi^{\pi}(s,a) &\overset{(4.9a)}{\triangleq} \mathbb{E}_{\pi,\mathcal{C}}[\sum_{t\geqslant 0}\gamma^t\Phi(S_t,A_t)|S_0=s] && (4.10a)\\
&= \mathbb{E}_{\pi,\mathcal{C}}[\Phi_1+\gamma\underbrace{\sum_{t\geqslant 1}\gamma^t\Phi(S_t,A_t)}_{\Psi^{\pi}(S_1,A_1)}|S_0=s,A_0=a] && (4.10b)\\
&= \mathbb{E}_{\pi,\mathcal{C}}[\Phi_1+\gamma\Psi^{\pi}(S_1,A_1)|S_0=s,A_0=a]. && (4.10c)
\end{aligned}$$

As a result, temporal difference (TD, see 2.3.2) learning methods can be used for learning SFs. For instance, a function approximator with parameters θ_{Ψ^π} can be trained from one-step transitions $(s,a,s',a') \sim \mathcal{B}$ by minimising the (one-step) TD loss:

$$\mathcal{L}_{\text{TD-}\Psi}(\theta_{\Psi^\pi}) \triangleq \mathbb{E}_{(s,a,s',a')\sim\mathcal{B}}\|\Psi(s,a;\theta_{\Psi^\pi})-\Phi(s,a)-\gamma\,\texttt{SG}[\Psi(s',a';\theta_{\Psi^\pi})]\|, \quad (4.11)$$

where $\texttt{SG}[\cdot]$ is a stop-gradient operation for semi-gradient TD updates (see §2.3.2).

Generalised policy evaluation. *Eqn.* (4.9) suggests that SFs are task-agnostic. In particular, given Ψ^π, the value of π for a new preference $\mathbf{w}'$ can be easily computed with generalised policy evaluation (GPE, Barreto et al., 2017), i.e., $\Psi^\pi(s,a)^\top\mathbf{w}' = q^{\pi,\mathbf{w}'}(s,a)$.

Example 4.2 (didactic example for generalised policy evaluation (GPE))***:*** Consider the tasks from *Example* 4.1 and a policy π_{GY}, which navigates the gridworld to the nearest green or yellow cell without stepping on the red cells.

To evaluate the policy on all 3 tasks (rewards) from *Example* 4.1, there are options: (i) use policy evaluation methods from §2.3.2, i.e., compute from scratch the value of the policy π_{GY} for each reward function; and (ii) calculate only once the SFs of the policy, $\Psi^{\pi_{\text{GY}}}$, and then perform GPE with the corresponding preference vectors, i.e.,

$$\begin{aligned}
q^{\pi_{\text{GY}},\mathbf{w}^1}(s,a) &= \Psi^{\pi_{\text{GY}}}(s,a)^\top\mathbf{w}^1 && (4.12a)\\
q^{\pi_{\text{GY}},\mathbf{w}^2}(s,a) &= \Psi^{\pi_{\text{GY}}}(s,a)^\top\mathbf{w}^2 && (4.12b)\\
q^{\pi_{\text{GY}},\mathbf{w}^3}(s,a) &= \Psi^{\pi_{\text{GY}}}(s,a)^\top\mathbf{w}^3. && (4.12c)
\end{aligned}$$

Note that GPE scales a lot more gracefully than policy evaluation from scratch as the number of tasks grows. However, GPE relies on the assumption that it is possible to efficiently and effectively compute or estimate the SFs of the policy.

Planning with successor features. Fast policy evaluation thanks to GPE, and by extension SFs, can accelerate value-based planning. First, we note a theoretical result:

Theorem 4.1 (generalised policy improvement (GPI, Barreto et al., 2017))***:*** Let a set of policies $\Pi = \{\pi^1, \dots, \pi^K\}$ and let $\{q^{\pi^1, r}, \dots, q^{\pi^K, r}\}$ the corresponding action-value functions for reward function r. Next, we can show that the GPI policy, i.e.,

$$\pi_{\mathrm{GPI}}(s) \triangleq \arg\max_{a \in \mathbb{A}} \max_k q^{\pi^k, r}(s, a), \tag{4.13}$$

is a strict (unless already optimal) improvement over the set Π, i.e.,

$$q^{\pi_{\mathrm{GPI}}, r}(s, a) \geqslant \max_k q^{\pi^k, r}(s, a), \tag{4.14}$$

for any $s \in \mathbb{S}$ and $a \in \mathbb{A}$, where $q^{\pi_{\mathrm{GPI}}, r}$ is the action value of the GPI policy.

Proof. See (Barreto et al., 2017). □

Intuitively, *Theorem 4.1* states that, provided the action-value functions of a set of policies, we can extract a policy that dominates all of them. Also, we saw that GPE is an efficient mechanism for evaluating a policy in a number of tasks. Next, we combine these two.

Lemma 4.1 (generalised policy improvement (GPI) with successor features (SFs))***:*** Given the SFs of a set of policies, i.e., $\{\Psi^{\pi^1}, \dots, \Psi^{\pi^K}\}$, we can apply GPI to derive a new (improved) policy π_{GPI} whose performance on a task with preference vector $\mathbf{w}$ is no worse that the performance of any of $\pi \in \Pi$ on the same task, given by:

$$\pi_{\mathrm{GPI}}(s) \triangleq \arg\max_{a \in \mathbb{A}} \max_k \Psi^{\pi^k}(s, a)^\top \mathbf{w}. \tag{4.15}$$

Note that in *Eqn.* (4.15) the $\max_k$ can be thought as a policy selection/planning operation. In other words, it selects the policy that for a given state-action (s, a) pair and task $\mathbf{w}$ has the highest evaluation, and then performs an improvement step upon it (since the $\max_a$).

Example 4.3 (didactic example for generalised policy improvement (GPI))***:*** Consider the 3 tasks from *Example* 4.1 and corresponding policies π^1, π^2 and π^3, which are optimal for tasks $\mathbf{w}^1$, $\mathbf{w}^2$ and $\mathbf{w}^3$, with SFs Ψ^{π^1}, Ψ^{π^2} and Ψ^{π^3}, respectively.

Let a new task in which the agent receives -1 reward for stepping on any of the coloured cells, i.e., $\mathbf{w}^{\mathrm{new}} = [-1, -1, -1]$. Following any of the policies π^1, π^2 and π^3 leads to some negative reward since all three policies are seeking at least one colour.

Nonetheless, the GPI policy in *Eqn.* (4.15) for $\mathbf{w}^{\mathrm{new}}$ is optimal since it avoids green (resp. red, yellow) because for state-action pairs that lead to a green (resp. red, yellow) cell the $\max_k$ is won by $k = 3$ (resp. $k = 1$, $k = 2$).

4.2 Methods

In the context of learning from sub-optimal (no-reward) demonstrations, which is the focus of this chapter, if we could estimate the SFs of the demonstrators (a.k.a. other agents), we could utilise them for improving the ego-agent's policy with GPI. However, to do so with conventional methods, we would require access to their rewards, cumulants and/or preferences. In our setting, we can only observe their sequence of states and actions.

4.2.1 Inverse temporal difference (ITD) learning

In this section, we present an offline multi-task inverse reinforcement learning (IRL, §2.2.4) method for learning (i) cumulants; (ii) per-agent successor features; and (iii) corresponding agent preferences from no-reward demonstration, as defined in §4.1. We call the proposed method *inverse temporal difference* (ITD) learning and summarise it in Algorithm 2.

Formalisation. Let function approximators for learanble cumulants, preferences vectors and SFs with parameters θ_{Φ}, $\{\mathbf{w}^k\}_{k=1}^K$ and $\{\theta_{\Psi^k} \equiv \theta_{\Psi^k}\}_{k=1}^K$, respectively. We model the (other) agents that generated the no-reward demonstrations $\mathcal{D}$ as soft-optimal for an *unknown* task, i.e., the k-th agent's policy π^k is soft-optimal for the task $\mathbf{w}^k$ and is given by:

$$\begin{aligned}
\pi^k(A = a|s) &= \frac{\exp(q^{\pi^k}(s,a))}{\sum_{b\in\mathbb{A}} \exp(\exp(q^{\pi^k}(s,b))}, && (4.16a)\\
&= \texttt{softmax}[q^{\pi^k}(s,\cdot)][a] && (4.16b)\\
&\overset{(4.9b)}{=} \texttt{softmax}[\Psi^{\pi^k}(s,\cdot)^\top \mathbf{w}^k][a], && (4.16c)
\end{aligned}$$

where $\texttt{softmax}[f(\cdot)][i]$ is the i-th element of the $\texttt{softmax}$-normalised $f(\cdot)$ 1D-array. We choose to represent the policy π^k as a function of the SFs Ψ^{π^k} and preferences vector $\mathbf{w}^k$, to learn this quantities from the no-reward demonstrations via BC, i.e., minimising the loss:

$$\mathcal{L}_{\text{BC}}(\theta_{\Psi^k}, \mathbf{w}^k) \triangleq \mathbb{E}_{(s,a)\sim\mathcal{D}^k}[-\log \texttt{softmax}[\Psi(s,\cdot;\theta_{\Psi^k})^\top \mathbf{w}^k][a]], \tag{4.17}$$

where $\mathcal{D}^k$ are the demonstrations, generated by the k-th (other) agent, i.e., $\mathcal{D} \supset \mathcal{D}^k \sim \pi^k$.

Our key insight is that the cumulants should be TD-consistent with all agents' successor features, as of *Eqns.* (4.10, 4.11) and hence we can train the θ_{Φ} accordingly. This procedure inverts[†] the standard TD learning framework for SFs where they are trained to be consistent with a *fixed* (unlearned) cumulant function Φ. Instead, we first train the SFs and preferences vectors to 'explain' the other agents' behaviour with the BC loss *Eqn.* (4.17), and then train θ_{Φ}, θ_{Ψ^k} to be (self-)consistent by minimising the ITD loss, given by:

[†]Hence the name *inverse TD learning.*

$$\mathcal{L}_{\mathrm{ITD}}(\theta_\Phi, \theta_{\Psi^k}) \triangleq \mathbb{E}_{(s,a,s',a')\sim\mathcal{D}^k}\|\Psi(s,a;\theta_{\Psi^k}) - \Phi(s,a;\theta_\Phi) - \mathtt{SG}[\gamma\Psi(s,a;\theta_{\Psi^k})]\|. \quad (4.18)$$

After training, by the definitions of cumulants, preferences vectors and SFs from 4.1, we recover: (i) action-value functions that can be used for imitating the demonstrator, i.e.,

$$q^{\pi^k}(s,a;\theta) = \Psi(s,a;\theta_{\Psi^k})^\top \mathbf{w}^k \quad (4.19)$$

and (ii) an explicit reward function for each agent, i.e.,

$$r^k(s,a;\theta) = \Phi(s,a;\theta_\Phi)^\top \mathbf{w}^k, \quad (4.20)$$

where $\theta \triangleq (\theta_\Phi, \{\mathbf{w}^k\}_{k=1}^K, \{\theta_{\Psi^k}\}_{k=1}^K)$ are all the learnable parameters. Moreover, we can formally show that the learned reward functions are 'valid' in the IRL sense.

Theorem 4.2 (validity of the inverse temporal difference (ITD) learning minimisers)***:*** The minimisers of $\mathcal{L}_{\mathrm{BC}}$ and $\mathcal{L}_{\mathrm{ITD}}$ are potentially-shaped cumulants that explain the reward-free demonstrations.

Proof. See §B.3. □

Intuition. We treat the demonstrators as (soft-)expert agents for some unknown tasks, whose reward functions can be decomposed into cumulants and preferences, as described in §4.1. In other words, while these agents may pursuit distinct objectives there is some 'behavioural basis' that can be inferred from the no-reward demonstrations.

To discover this basis, we use the fact that: (i) we can parametrise the function approximations to the demonstrators' policies with action-value functions and hence preferences vectors and SFs, which are trainable via BC; and (ii) the learned cumulants must be TD-consistency with *each and every* demonstrators' learned SFs.

Example 4.4 (single-task inverse reinforcement learning (IRL) with inverse temporal difference (ITD))***:*** To gain more intuition about the ITD algorithm, consider the simpler case of performing IRL with demonstrations from a single demonstrator.

This obviates the need for a representation of preferences, so we can use $\mathbf{w} = 1$. In this case Ψ is the action-value function q and Φ is simply the reward function r. Minimising *Eqn.* (4.17) reduces to finding a action-value function whose `softmax` gives the observed policy, and minimising *Eqn.* (4.18) finds a scalar reward that explains the action-value function. Our more general formulation, with cumulants Φ in place of a scalar reward, allows us to perform ITD learning on demonstrations from many policies, and to efficiently transfer to new tasks.

Algorithm 2: Inverse temporal difference (ITD) learning

```
Input :
    $\mathcal{D} = \{(s_1, a_1, \ldots, a_T; k)\}_{k=1}^{K}$   No-reward demonstrations
    $\lambda_{\mathbf{w}}$                                      $\mathcal{L}_1$ loss coefficient
Output :
    $\theta_\Phi$                          Parameters of cumulants network
    $\{\theta_{\Psi^k}\}_{k=1}^{K}$        Parameters of successor features approximators
    $\{\mathbf{w}^k\}_{k=1}^{K}$           Preferences vectors for the $K$ agents
// Initialisations
1 Initialise parameters $\theta_\Phi, \{\theta_{\Psi^k}, \mathbf{w}^k\}_{k=1}^{K}$
2 while budget do
3     Sample trajectories $\{\xi^{(n)} = (s_1^{(n)}, a_1^{(n)}, \ldots, s_T^{(n)}, a_T^{(n)}; k^{(n)})\}_{n=1}^{N} \sim \mathcal{D}$
      // Behaviour cloning (BC) loss minimisation
4     Calculate $\mathcal{L}_{\text{BC}}(\theta_{\Psi^k}, \mathbf{w}^k)$ on samples $\{\xi^{(n)}\}_{n=1}^{N}$      ▷ see Eqn. (4.17)
5     $\theta_{\Psi^k} \overset{\alpha}{\leftarrow} \nabla_{\theta_{\Psi^k}} \mathcal{L}_{\text{BC}}(\theta_{\Psi^k}, \mathbf{w}^k)$      ▷ update Ψs
6     $\mathbf{w}^k \overset{\alpha}{\leftarrow} \nabla_{\mathbf{w}^k} \left(\mathcal{L}_{\text{BC}}(\theta_{\Psi^k}, \mathbf{w}^k) + \lambda_{\mathbf{w}} \|\mathbf{w}^k\|_1\right)$      ▷ update ws
      // Inverse temporal difference (ITD) loss minimisation
7     Calculate $\mathcal{L}_{\text{ITD}}(\theta_\Phi, \theta_{\Psi^k})$ on samples $\{\xi^{(n)}\}_{n=1}^{N}$      ▷ see Eqn. (4.18)
8     $\theta_\Phi \overset{\alpha}{\leftarrow} \nabla_{\theta_\Phi} \mathcal{L}_{\text{ITD}}(\theta_{\Psi^k}, \theta_\Phi)$      ▷ update Φ
9     $\theta_{\Psi^k} \overset{\alpha}{\leftarrow} \nabla_{\theta_{\Psi^k}} \mathcal{L}_{\text{ITD}}(\theta_{\Psi^k}, \theta_\Phi)$      ▷ update Ψs
```

Practical implementation. In practice, we found that a sparsity prior, i.e., $\mathcal{L}_1$ regularisation loss, on preferences $\mathbf{w}^k$ is also key to promote disentangled cumulant dimensions (see *Figure 4.7*). In particular, we found the $\mathcal{L}_1$ loss made the algorithm more robust to the choice of dimension of Φ (see *Figure B.7*), but did not substantially affect the overall performance otherwise. Moreover, target networks (Mnih et al., 2013, 2015) are used for the ITD loss to further stabilise learning.

4.2.2 ΨΦ-learning with no-reward demonstrations

In this section, we present a RL agent that can leverage sub-optimal demonstrations to accelerate its learning, combining the ITD multi-task IRL algorithm (see §4.2.1) with a novel ignorance-averse GPI (see §2.1.2 and *Lemma 4.1*) planning mechanism. We call the proposed method ΨΦ*-learning* (ΨΦL) and summarise it in Algorithm 3.

Formalisation. ΨΦ-learning is an off-policy algorithm based on Q-learning (Watkins and Dayan, 1992) and leverages the no-reward demonstrations using ITD. The action-value

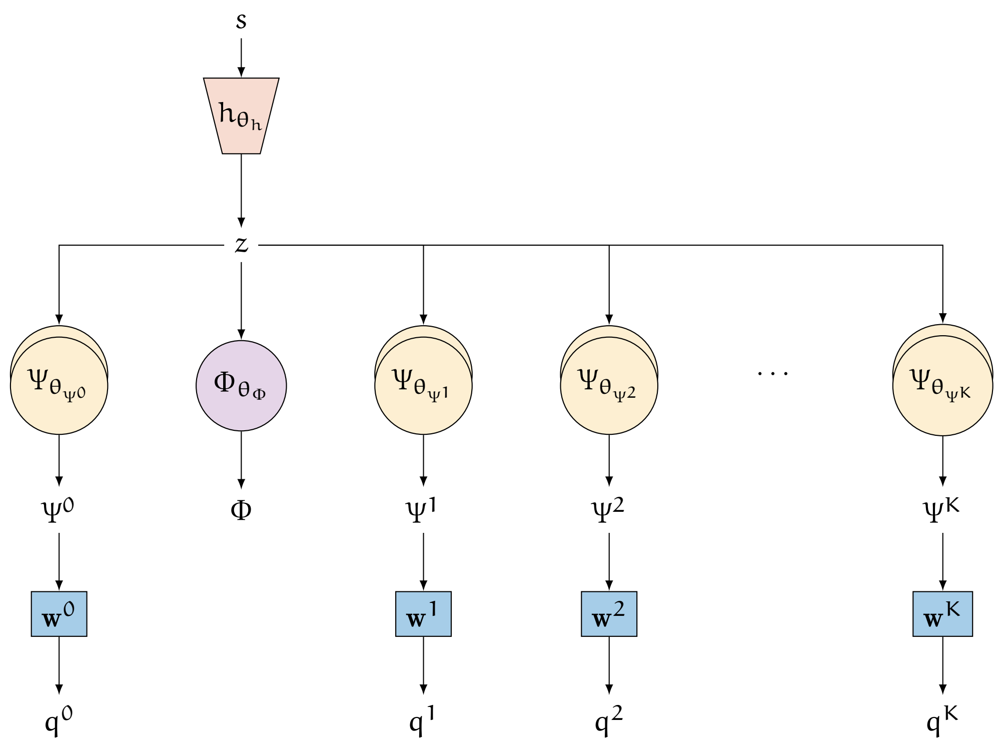


Figure 4.3: The $\Psi\Phi$-learning ($\Psi\Phi$L) agent's neural network (NN) architecture. It comprises of (i) a representation network h_{θ_h} that receives learning signal from all the losses, shaping a shared representation. (ii) a cumulants network Φ_{θ_Φ}; (iii) successor features (SFs) approximators for each of the demonstrators $\{\Psi_{\theta_{\psi k}}\}_{k=1}^{K}$ and the 'ego' reinforcement learning (RL) agent $\Psi_{\theta_{\psi 0}}$; and corresponding (iv) preferences vectors $\{\mathbf{w}^k\}_{k=0}^{K}$. Ensembles of two SFs approximators (per-agent) are used for combatting the overoptimisation of learned reward functions phenomenon (see §2.10).

function is represented with successor features, Ψ^0, and preferences, $\mathbf{w}^0$, as in *Eqn.* (4.9b).[†] The ego-agent interacts with the environment, storing its rollouts in a replay buffer, $\mathcal{B}$. On top of the ITD losses from §4.2.1, the $\Psi\Phi$-learner optimises: (i) a reward prediction loss; and (ii) TD-learning losses for its action value and SFs, given by:

$$\mathcal{L}_r(\theta_\Phi, \mathbf{w}^0) \triangleq \mathbb{E}_{(s,a,r')\sim\mathcal{B}} \|\Phi(s,a;\theta_\Phi)^\top \mathbf{w}^0 - r'\| \tag{4.21a}$$

$$\mathcal{L}_q(\theta_{\psi^0}) \triangleq \mathbb{E}_{(s,a,s',r')\sim\mathcal{B}} \|\Psi(s,a;\theta_{\psi^0})^\top \mathbf{w}^0 - r' - \texttt{SG}[\gamma \max_{a'} \Psi(s',a';\tilde{\theta}_{\psi^0})^\top \mathbf{w}^0]\| \tag{4.21b}$$

$$\mathcal{L}_{\text{TD-}\Psi}(\theta_{\psi^0}) \triangleq \mathbb{E}_{(s,a,s',a')\sim\mathcal{B}} \|\Psi(s,a;\theta_{\psi^0}) - \Phi(s,a;\theta_\Phi) - \gamma\, \texttt{SG}[\Psi(s',a';\theta_{\psi^0})]\|. \tag{4.21c}$$

Overall, the $\Psi\Phi$-learner minimises the $\mathcal{L}_{\Psi\Phi\text{L}}$ loss on rollouts $\mathcal{B}$ and demonstrations $\mathcal{D}$:

$$\mathcal{L}_{\Psi\Phi\text{L}}(\theta) \triangleq \textstyle\sum_{k=1}^{K}(\mathcal{L}_{\text{BC}}(\theta_{\psi^k}, \mathbf{w}^k) + \mathcal{L}_{\text{ITD}}(\theta_\Phi, \theta_{\psi^k})) + \mathcal{L}_r(\theta_\Phi, \mathbf{w}^0) + \mathcal{L}_q(\theta_{\psi^0}) + \mathcal{L}_{\text{TD-}\Psi}(\theta_{\psi^0}). \tag{4.22}$$

[†]We reserve the index $k = 0$ for the ego-agent.

Provided the SFs estimates $\{\theta_{\Psi^k}\}_{k=0}^{K}$ and the inferred ego preferences $\mathbf{w}^0$, we can use the GPI (see *Lemma 4.1*)) to obtain a policy π^{GPI}, which strictly dominates all policies, including the ego-agent's and the other agents', *for the ego-agents task*.

We observe in §4.3 (see **H3**) that naively applying GPI on all these learned SFs leads to instabilities and overoptimisation of learned objectives (i.e., SFs-induced action value functions). We address this problem by employing the lessons from the previous chapter, §3. In particular, we quantify the ignorance (i.e., epistemic uncertainty) about SFs and augment the GPI planning step with an ignorance-averse knowledge equivalent (KE). An ensemble of M successor features approximators (per-agent) $\{\{\theta_{\Psi^k_m}\}_{m=1}^{M}\}_{k=1}^{K}$ are learned (see *Figure 4.3*) and the ΨΦ-learner selects actions according to:

$$\pi^{\text{ego}}(s) \triangleq \arg\max_{a \in \mathbb{A}} \max_{k \in [K]} \min_{m \in [M]} \Psi(s, a; \theta_{\Psi^k_m})^\top \mathbf{w}^0. \tag{4.23}$$

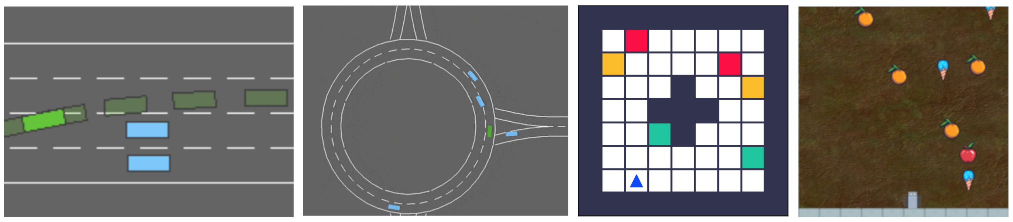
Intuition. The ΨΦ-learner projects all the data, rollouts and no-reward demonstrations, and its predictions in a unified space, the one of cumulants and preferences. This enables flow of information from the different data sources. In other words, we share the *same* cumulants between the ITD learning from other agents and the ego-learning, so that they span the joint space of reward functions. This can be also seen as a representation learning method, where by enforcing all agents, including the ego-agent, to share the same Φ, we transfer information about salient features of the environment from learning about one agent to benefit learning about all agents. This is, by extension, true all for the preferences.

Then, the ΨΦ-learner exploits the generalised policy updates from §4.1, i.e., GPE and GPI, to accelerate its online learning. Although *Lemma 4.1* suggests that the GPI is a strict improvement, it does not account for approximation errors. To avoid any potential overoptimisation problems, we adopt the principle of "pessimism in the face of ignorance".

Practical implementation. We use a NN function approximator and share representation between all the predictors. Its architecture is illustrated in *Figure 4.3*. To capture the ignorance in SFs to avoid overoptimisation of the learned action-values (see *Eqn.* (4.23)), we use ensembles of two components, trained according to *Remark* 2.8. Similar to §4.2.1, for any TD-like loss, we use target networks to stabilise learning.

Algorithm 3: $\Psi\Phi$-learning ($\Psi\Phi$L)

```
   Input :
       $\mathcal{D} = \{(s_1, a_1, \dots, a_T; k)_{k=1}^{K}\}$    No-reward demonstrations
       $\lambda_{\mathbf{w}}$                                  $\mathcal{L}_1$ loss coefficient
       $M$                                     Number of ensemble members
   Output :
       $\theta_{\Phi}$                     Parameters of cumulants network
       $\{\mathbf{w}^k\}_{k=0}^{K}$               Preferences vectors
       $\{\{\theta_{\Psi^k_m}\}_{m=1}^{M}\}_{k=0}^{K}$     Parameters of successor features approximators
    // Initialisations
 1 Initialise parameters $\theta_{\Phi}, \{\{\theta_{\Psi^k_m}\}_{m=1}^{M}, \mathbf{w}^k\}_{k=0}^{K}$ and buffer $\mathcal{B} = \{\}$
 2 while budget do
       // Agent-environment online interaction
 3     $s \leftarrow$ env.reset(), $t \leftarrow 0$
 4     while not done do
 5         Infer ego-agent's task with online linear regression
               $\mathbf{w}^0 \leftarrow \arg\min_{w} \mathbb{E}_{(s,a,r')\sim\mathcal{B}} \|\Phi(s, a; \theta_{\Phi})^{\top} w - r'\|$      ▷ see Eqn. (4.21a)
           Plan with ignorance-averse generalised policy improvement (GPI)
               $a^* \leftarrow \arg\max_{a\in\mathbb{A}} \max_{k\in[K]} \min_{m\in[M]} \Psi(s, a; \theta_{\Psi^k_m})^{\top} \mathbf{w}^0$      ▷ see Eqn. (4.23)
           $s', r', \text{done} \leftarrow$ env.step($a^*$)
           // Book-keeping
 6         $\mathcal{B} \leftarrow \mathcal{B} \cup (s, a^*, r', s')$, $s' \leftarrow s$, $t \leftarrow t + 1$
 7         Append online demonstrations in $\mathcal{D}$                  ▷ optional
       // Learning from sub-optimal (no-reward) demonstrations
 8     $\theta_{\Phi}, \{\theta_{\Psi^k_m}, \mathbf{w}^k\}_{k=1}^{K} \leftarrow \text{ITD}\left(\mathcal{D}, \lambda_{\mathbf{w}}, \theta_{\Phi}, \{\theta_{\Psi^k}, \mathbf{w}^k\}_{k=1}^{K}\right)$      ▷ see Algorithm 2
       // Learning from ego-experience (RL)
 9     Sample SARS transitions $\{b_n = (s^{(n)}, a^{(n)}, r'^{(n)}, s'^{(n)})\}_{n=1}^{N} \sim \mathcal{B}$
       // Reward loss minimisation
10     Calculate $\mathcal{L}_r(\theta_{\Phi}, \mathbf{w}^0)$ on samples $\{b_n\}_{n=1}^{N}$            ▷ see Eqn. (4.21a)
11     $\theta_{\Phi} \xleftarrow{\alpha} \nabla_{\theta_{\Phi}} \mathcal{L}_r(\theta_{\Phi}, \mathbf{w}^0)$                              ▷ update $\Phi$
       // Temporal difference (TD) losses minimisation
12     Calculate $\mathcal{L}_q(\theta_{\Psi^0})$ on samples $\{b_n\}_{n=1}^{N}$             ▷ see Eqn. (4.21b)
13     Calculate $\mathcal{L}_{\text{TD-}\Psi}(\theta_{\Psi^0})$ on samples $\{b_n\}_{n=1}^{N}$         ▷ see Eqn. (4.21c)
14     $\theta_{\Psi^0} \xleftarrow{\alpha} \nabla_{\theta_{\Psi^0}} \left(\mathcal{L}_q(\theta_{\Psi^0}) + \frac{1}{|\Psi|}\mathcal{L}_{\text{TD-}\Psi}(\theta_{\Psi^0})\right)$            ▷ update $\Psi^0$
```



(a) `Highway` *(b)* `Roundabout` *(c)* `CoinGrid` *(d)* `FruitBot`

Figure 4.4: Environments studied in this chapter. Environments (a-b) are multi-agent environments in which the ego-agent must learn online from other agents, and learn to navigate around other agents in the environment (see §4.3.1). Environments (c-d) are single-agent. In environment (c) we test whether the ego-agent can learn offline from a set of demonstrations previously collected by other agents (see §4.3.4). Environment (d) is used to test whether our method can scale to more complex, high-dimensional tasks (see §4.3.1).

4.3 Experiments

We conduct a series of experiments to determine how well ΨΦ-learning (ΨΦL) functions as an RL, IRL, imitation learning, and transfer learning algorithm.

Baselines. We benchmark against the following methods: (i) **DQN** (Mnih et al., 2013), (ii) **Behaviour cloning (BC)** is a simple imitation learning method in which we learn $p(a|s)$ via supervised learning on the demonstration data, (iii) **DQN+BC-AUX** is the DQN agent with an additional behaviour cloning (BC) auxiliary loss (Hernandez-Leal et al., 2019), (iv) **GAIL** is the Generative Adversarial Imitation Learning (Ho and Ermon, 2016), which uses a GAN-like approach to approximate the expert policy, and (v) **SQILv2** is Soft Q Imitation Learning (Reddy et al., 2019), a recently proposed imitation technique that combines imitation learning and RL, and works in the absence of rewards. For high-dimensional environments, we replace DQN with **PPO**, Proximal Policy Optimization (Schulman et al., 2017). Both DQN and PPO are trained to optimise environment reward through experience, and do not have access to other agents' experiences.

Environments. Experiments are conducted using four environments, shown in *Figure 4.4*. We cover a broad range of problem setting, including both multi-agent and single-agent environments, as well as learning online during RL training, or offline from previously collected demonstrations.

`Highway` (Leurent, 2018) is a multi-agent autonomous driving environment in which the ego-agent must safely navigate around other cars and reach its goal. The other agents follow near-optimal scripted policies for various goals, depending on the scenario. In the single-task scenario (`SingleHighway`), other agents have the same objective as the ego-agent, so their experience is directly relevant. In the adversarial task

(`AdversarialHighway`), the other agents do not move, and the ego-agent has to accelerate and go to a particular lane while avoiding other vehicles. Finally, in the multi-task scenario (`MultiHighway`), the other agents and ego-agent have different preferences over target speed, preferred lane, and following distance. We consider the multi-task scenario to be the most realistic and representative of real highway driving with human drivers. In addition to highway driving, we also study the more complex `Roundabout` task. `Roundabout` is inherently multi-task, in that other agents randomly exit either the first or second exit, while the ego-agent must learn to take the third exit.

`CoinGrid` is a single-agent grid-world, environment containing goals of different colours. We collect offline trajectories of pre-trained agents with preferences for different goals. red. During training, the ego-agent is only rewarded for collecting a subset of the possible goals. We can then test how well the ego-agent is able to transfer to a goal that was never experienced during training (§4.3.4). This environment also enables learning easily interpretable preference vectors, allowing us to visualize how well our method works as an IRL method for inferring rewards (§4.3.2).

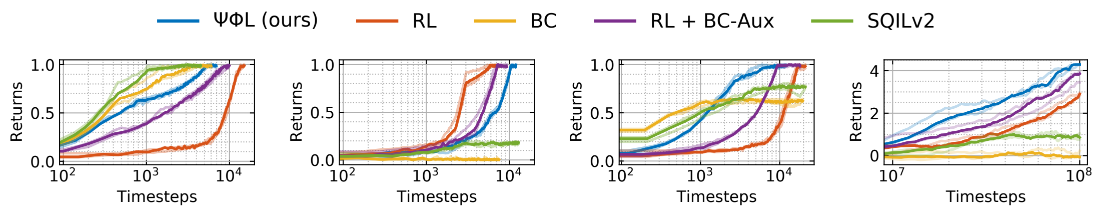
`FruitBot` is a high-dimensional, procedurally generated, single-agent environment from the OpenAI ProcGen (Cobbe et al., 2020) suite. We use `FruitBot` to test whether ΨΦ-learning can scale up to more complex RL environments, requiring larger deep neural network architectures that learn directly from pixels. The agent must navigate around randomly generated obstacles while collecting fruit, and avoiding other objects and walls. To create a multi-task version of `FruitBot`, we define additional tasks which vary agents' preferences over collecting objects in the environment, and train PPO baselines on these task variants. The ego-agent observes the states and actions of these trained agents playing the game in parallel with its own interactions.

4.3.1 Accelerating RL with no-reward demonstrations

This section addresses three hypotheses: **H1**: When the unlabelled demonstrations are relevant, ΨΦ-learning can accelerate or improve performance of the ego-agent when learning with online RL; and **H2**: If the demonstrations are irrelevant, biased, or are generated by sub-optimal demonstrators, ΨΦ-learning can perform at least as well as standard RL. **H3**: Ignorance-averse planning is essential when learning from static no-reward demonstrations.

Figure 4.5 shows the results of ΨΦ-learning and the baselines in the `Highway` and `FruitBot` environments. In `SingleHighway` (*Figure 4.5a*), when other agents' experience is entirely relevant to the ego-agent's task, imitation learning methods like BC and SQILv2 learn fastest. DQN learns slowly because it does not use the other agents' experience. However, ΨΦ-learning achieves competitive results, out-performing DQN+BC-Aux (Hernandez-Leal et al., 2019; Ndousse et al., 2020).

In `AdversarialHighway` (*Figure 4.5b*), the other agents' behaviors are irrelevant for



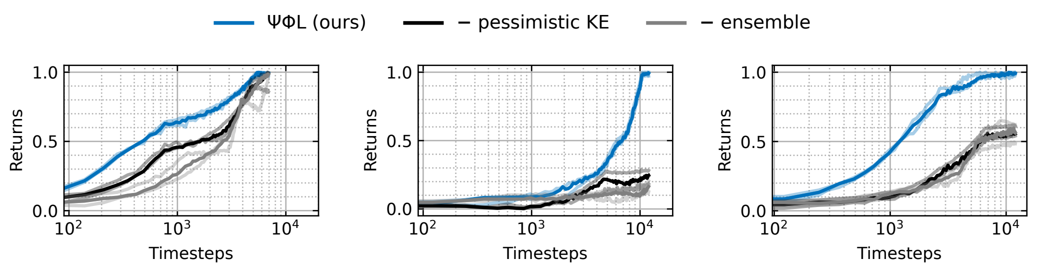
(a) `SingleHighway` *(b)* `AdversarialHighway` *(c)* `MultiHighway` *(d)* `FruitBot`

Figure 4.5: Learning curves for ΨΦ-learning (ΨΦL) and baselines in three tasks in the multi-agent `Highway` environment ***(a-c)***, and in single-agent `FruitBot` ***(d)***. Tasks ***(a)*** and ***(b)*** represent extreme cases where either reinforcement learning (RL) or imitation learning (IL) is irrelevant. In `SingleHighway` ***(a)***, other agents have the same task as the ego-agent, so IL excels. In `AdversarialHighway` ***(b)***, other agents exhibit degenerative behaviour, so IL performs extremely poorly and traditional RL (DQN) excels. In both of these extreme cases, ΨΦ-learning achieves good performance, showing it can flexibly reap the benefits of either IL or RL as appropriate. `MultiHighway` ***(c)*** is most realistic; here, other agents have varied preferences and goals that may or may not relate to the ego-agent's task. ΨΦ-learning clearly outperforms baseline techniques. Similar results are shown in `FruitBot` ***(d)***, showing that ΨΦ-learning scales well to high-dimensional environments, consistently outperforming baselines like PPO and SQIL. We plot mean performance over 3 runs and individual runs are transparent.

the ego-agent's task, so imitation learning (BC and SQILv2) performs poorly, while traditional RL techniques (DQN and DQN+BC-Aux) perform best. The performance of ΨΦ-learning does not suffer like other imitation learning methods; instead, it retains the performance of standard RL (**H2**). ΨΦ-learning can flexibly reap the benefits of either imitation learning or RL, depending on what is most beneficial for the task.

The `MultiHighway` (*Figure 4.5c*) is the most realistic autonomous driving task, in which other agents navigate the highway with varying driving styles. Here, ΨΦ-learning clearly out-performs all other methods, suggesting it can leverage information about other agents' preferences in order to learn the underlying task structure of the environment, acclerating performance on the ego-agent's RL task (**H1**). `FruitBot` (*Figure 4.5d*) gives consistent results, showing that ΨΦ-learning scales well to high-dimensional, single-agent tasks while still outperforming BC, SQIL, PPO, and PPO+BC-Aux.

Figure 4.6 shows the results of ΨΦ-learning and ignorance-unaware variants, aiming to assess the impact of ignorance quantification and pessimism (ignorance-averse KE in *Eqn.* (4.23)). We observe that not using an ignorance-averse (pessimistic) KE (a.k.a. "- pessimistic KE") is as bad as not using an ensemble at all (a.k.a. "- ensemble"). In particular, in `SingleHighway` (*Figure 4.6a*), they learn learn slower than ΨΦL, and in `AdversarialHighway` (*Figure 4.6b*) and `MultiHighway` (*Figure 4.6c*), they fail to learn completely. Therefore, we can conclude that ignorance quantification and an ignorance-



(a) `SingleHighway` *(b)* `AdversarialHighway` *(c)* `MultiHighway`

Figure 4.6: Learning curves for ΨΦ-learning (ΨΦL) and ignorance-unaware variants in three tasks in the multi-agent `Highway` environment ***(a-c)***. The "- pessimistic KE" variant learns ensembles of cumulants and successor features (SFs) but uses only on for action selection, i.e., $m = 1$ in *Eqn.* (4.23). The "- ensemble" variant does not learn ensembles at all. The performance of the ignorance-unaware variants is relatively poor.

averse KE are vital for ΨΦL's performance (**H3**).

4.3.2 Inferring reward functions

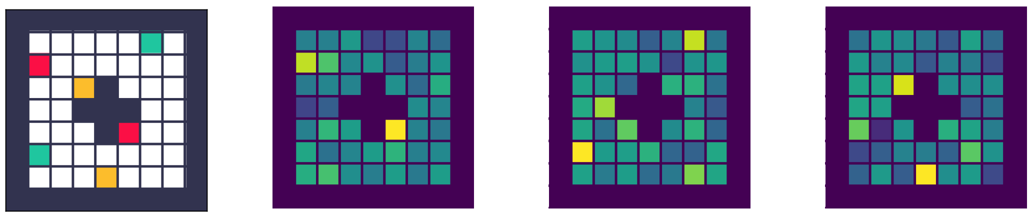
We now test hypothesis **H4**: ITD is an effective IRL method, and can accurately infer other agents' rewards. We present a quantitative and qualitative study of the rewards for other agents that are inferred by ITD, as well as the learned cumulants and preferences. Here, we focus solely on offline IRL and use only ITD to learn from offline reward-free demonstrations, without any ego-agent experience.

To quantitatively evaluate how well ITD can infer rewards, we train an RL agent on the inferred reward function, and compare the performance to other imitation learning and IRL methods. *Table 4.1* gives the performance in terms of normalised returns on all three environments. Using ITD to infer rewards results in significantly higher performance than BC and SQIL, in two environments, and competitive performance in `FruitBot`. We note that unlike ΨΦ-learning, BC and SQIL directly learn a policy from demonstrations, and do not actually infer an explicit reward function. In contrast, GAIL does infer an explicit reward function, and ITD gives consistently higher performance than GAIL in all three environments. These results demonstrate that ITD is an effective IRL technique (**H4**).

Qualitatively, we can evaluate how well the cumulants inferred by ITD in the `CoinGrid` environment span the space of possible goals. We compute the learned cumulants $\hat{\Phi}(s)$ for each square s in the grid. *Figure 4.7a* shows the original `CoinGrid` game, and *Figures 4.7b-4.7d* shows the first three dimensions of the learned cumulant vector, $\hat{\Phi}_1$-$\hat{\Phi}_3$ (the rest are given in *Figure B.6*). We find that $\hat{\Phi}_1$ is most active for red coins, $\hat{\Phi}_2$ for green, and $\hat{\Phi}_3$ for yellow. Clearly, ITD has learned cumulant features that span the space of goals for this game. See §B.4 for more details and visualisations of the learned rewards and preferences.

Table 4.1: We evaluate how well inverse temporal difference (ITD) learning is able to infer the correct reward function by training an RL agent on the inferred rewards, and comparing this to alternative imitation learning methods in three environments. All methods are trained on expert demonstrations. A "$\diamondsuit$" indicates methods that infer an *explicit* reward function and then use one of DQN or PPO to train an RL agent, depending on the environment. A "$\clubsuit$" indicates methods that directly learn a policy from demonstrations. A "$\dagger$" indicates methods that use privileged task id information for handling multi-task demonstrations. We report mean and standard error of *normalised returns* over 3 runs, where higher-is-better and the performance is upper bounded by 1.0, reached by the same RL agent, trained with the ground truth reward function.

Methods	`Roundabout`$^{\text{DQN}}$	`CoinGrid`$^{\text{DQN}}$	`FruitBot`$^{\text{PPO}}$
BC$^{\dagger\clubsuit}$ (Pomerleau, 1989)	0.81±0.02	0.69±0.06	**0.37**±0.02
SQIL$^{\dagger\clubsuit}$ (Reddy et al., 2019)	0.85±0.02	0.64±0.05	**0.35**±0.03
GAIL$^{\dagger\diamondsuit}$ (Ho and Ermon, 2016)	0.77±0.07	0.73±0.02	0.31±0.02
ITD$^{\diamondsuit}$ (ours, §4.2.1)	**0.92**±0.01	**0.77**±0.03	**0.35**±0.04



(a) `CoinGrid` *(b)* $\hat{\Phi}_1$ *(c)* $\hat{\Phi}_2$ *(d)* $\hat{\Phi}_3$

Figure 4.7: Qualitative evaluation of the learned cumulants in the `CoinGrid` task. Cumulants $\hat{\Phi}_1$, $\hat{\Phi}_2$, and $\hat{\Phi}_3$ seem to capture the red, green, and yellow blocks, respectively. Therefore, linear combinations of the learned cumulants can represent arbitrary rewards in the environment, which involve stepping on the coloured blocks.

4.3.3 Predicting other agents' behaviour

Here, we investigate hypothesis **H5**: $\Psi\Phi$-learning works as an effective imitation learning method, allowing for accurate prediction of other agents' actions. To test this hypothesis, we train other agents in the Roundabout environment, then split no-reward demonstrations from these agents into a train dataset (80%) and a held-out test dataset. We use the train dataset to run ITD, which means that we use the data to learn both Φ and the Ψ and $\mathbf{w}$ for other agents. Because it is specifically designed to accurately predict other agents' actions, we use BC as the baseline. We compare this to using only ITD, and using the full $\Psi\Phi$-learning algorithm including ITD *and* learning from RL and experience to update the shared cumulants Φ.

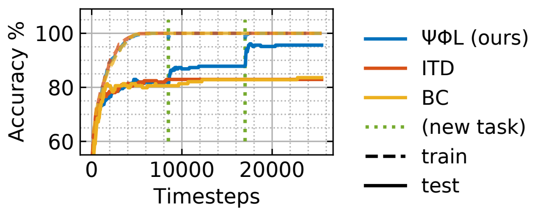


Figure 4.8: Test accuracy in predicting other agents' actions. The shared cumulants Φ for modelling others- and ego- reward functions allow our ΨΦ-learning (ΨΦL) to improve its ability to predict others' actions by experiencing new ego-tasks. Pure behaviour cloning (BC) and our ITD learning IRL methods achieve high train accuracy but they do not have a mechanism for utilising RL experience to improve their generalisation to the test set as new tasks are provided.

Accuracy in predicting other agents' actions on the held-out test set is used to measure imitation learning performance. *Figure 4.8* shows accuracy over the course of training. At each phase change marked in the figure, the ego-agent is given a new task, to test how the representation learning benefits from diverse ego-experience. We see that although BC obtains accurate train performance, it generalises poorly to the test set, reaching little over 80% accuracy. Without RL, ΨΦ-learning achieves similar performance. However, when using RL to improve imitation, ΨΦ-learning performs well on both the train and test set, achieving markedly higher accuracy ($\approx 95\%$) in predicting other agents' behaviour. This suggests that when ΨΦ-learning uses RL and interaction with the world to improve the estimation of the shared cumulants Φ, this in turn improves its ability to model the Q function of other agents and predict their behaviour. Further, ΨΦ-learning adapts well when the agent's goal changes, since it uses SFs to disentangle the representation of an agent's goal from environment dynamics. Taken together, these results demonstrate that ΨΦ-learning also works as a competitive imitation learning method (**H5**).

4.3.4 Transfer and few-shot generalisation

Since SFs have been shown to improve generalization and transfer in RL, here we test hypothesis **H6**: ΨΦ-learning will be able to generalize effectively to new tasks in a few-shot transfer setting. Using the `CoinGrid` environment, it is possible to precisely test whether the ego-agent can generalise to a task it has never experienced during training. Specifically, we would like to determine whether: (i) the ego-agent can generalise to tasks it was never rewarded for during training (but which it may have seen other agents demonstrate), and (ii) the ego-agent can generalise to tasks not experienced by *any* agents during training.

Table 4.2 shows the results of transfer experiments in which agents are given 0, 1, or 100 additional training episodes to adapt to a new task. Unlike SQIL, ΨΦ-learning is able to adapt 0-shot to obtain some reward on the new tasks, and fully adapt after a single episode to achieve the maximum reward on all transfer tasks. This is because ΨΦ-learning uses SFs to disentangle preferences (goals) in the task representation, and learn about the space of possible preferences from observing other agents. To adapt to a new task, it need only infer the correct preference vector. Task inference is trivially implemented as a least squares

Table 4.2: We evaluate how well ΨΦ-learning (ΨΦL) is able to transfer to new tasks in a few-shot fashion. We construct a multi-task variant of the `CoinGrid` environment: The ego-agent is provided demonstrations for either capturing only red coins `R` or only green coins `G`. Then it is evaluated on 4 different tasks: collecting (i) both red and green coins `R+G`, (ii) collecting red and avoiding green coins `R-G`, (iii) avoiding red and collecting green coins `-R+G` and (iv) avoiding both red and green coins `-R-G`. A "$\diamondsuit$" indicates methods that use a single model for all tasks, while "$\clubsuit$" indicates methods that require one model per task, i.e., they comprise of 4 models. Because it disentangles preferences from task representation, ΨΦ-learning is able to adapt to reach optimal performance on the new tasks after a single episode or improve intra-episode from the first episode after experiencing the first rewards. In contrast, SQIL (Reddy et al., 2019) takes 100 episodes to adapt.

Methods	`R+G`	`R-G`	`-R+G`	`-R-G`
0-shot				
SQILv2$^\clubsuit$ (Reddy et al., 2019)	1.0±0.0	0.0±0.0	0.0±0.0	−1.0±0.0
ΨΦ-learning$^\diamondsuit$ (ours, §4.2.2)	1.0±0.0	**0.2**±0.1	**0.2**±0.1	−**0.4**±0.2
1-shot				
SQILv2$^\clubsuit$ (Reddy et al., 2019)	**1.0**±0.0	0.0±0.0	0.0±0.0	−1.0±0.0
ΨΦ-learning$^\diamondsuit$ (ours, §4.2.2)	**1.0**±0.0	**1.0**±0.0	**1.0**±0.0	**1.0**±0.0
100-shot				
SQILv2$^\clubsuit$ (Reddy et al., 2019)	**1.0**±0.0	**1.0**±0.0	**1.0**±0.0	**1.0**±0.0
ΨΦ-learning$^\diamondsuit$ (ours, §4.2.2)	**1.0**±0.0	**1.0**±0.0	**1.0**±0.0	**1.0**±0.0

regression problem, see *Eqn.* (4.21a): Having experienced $\mathcal{B}^{\text{new}}$ in the new task, the ΨΦ-learner identifies the preference vector for the new task by solving $\min_{\mathbf{w}} \sum_{\mathcal{B}^{\text{new}}} \mathcal{L}_{\text{r}}(\theta_{\Phi}, \mathbf{w})$. In contrast, SQIL requires 100 episodes to reach the same performance (**H6**).

4.4 Related work

The literature review for imitation learning (IL) from expert demonstrations and the reinforcement learning (RL) problem setting and the Q-learning algorithm can be found in §2. In this section, we review methods for learning from sub-optimal demonstrations, using SFs and social learning, to better contexualise the contributions of this chapter.

Learning from sub-optimal demonstrations. Coates et al. (2008); Grollman and Billard (2011); Zheng et al. (2014); Choi et al. (2019); Shiarlis et al. (2016); Brown et al. (2019) have studied methods that enable, under certain assumptions, imitation learners to surpass their demonstrators' performance. In contrast, our method integrates demonstrations into an online RL pipeline and can use the demonstrations to improve learning on a new task. Our inverse temporal difference (ITD) learning offline multi-task inverse RL algorithm is similar to the Cascaded Supervised IRL (CSI) approach (Klein et al., 2013). However, CSI assumes a single-task, deterministic expert while our ITD learning does not.

Reinforcement learning (RL) with demonstrations. Demonstration trajectories have been used to accelerate the learning of RL agents (Taylor et al., 2011; Vecerik et al., 2017; Rajeswaran et al., 2017; Hester et al., 2018; Gao et al., 2018; Nair et al., 2018; Paine et al., 2018, 2019), as well as demonstrations where actions and/or rewards are unknown (Borsa et al., 2017; Torabi et al., 2018; Sermanet et al., 2018; Liu et al., 2018; Aytar et al., 2018; Brown et al., 2019). In contrast to the standard IL setup, these methods allow improving over the expert performance as the policy can be further fine-tuned via RL. Offline RL with online fine-tuning (Kalashnikov et al., 2018; Levine et al., 2020) can be framed under this settings too. Our method builds on the same principles, however, unlike these works, we do not assume that the demonstration data either come with reward annotations, or that they relate to the same task the RL agent is learning.

Successor features. Successor features (SFs) are a generalisation of the successor representation (Dayan, 1993) for continuous state and action spaces (Barreto et al., 2017). Prior work has used SFs for (i) zero-shot transfer (Barreto et al., 2017; Borsa et al., 2018; Barreto et al., 2020); (ii) exploration (Janz et al., 2019; Machado et al., 2020); (iii) skills discovery (Machado et al., 2017; Hansen et al., 2019); (iv) hierarchical RL (Barreto et al., 2019) and theory of mind (Rabinowitz et al., 2018). Nonetheless, in all the aforementioned settings, direct access to the rewards or cumulants was provided. Our method, instead, uses demonstrations without reward labels for inferring the cumulants and learning the corresponding SFs. More closely to this work, Lee et al. (2019b) propose learning cumulants and successor features for a *single-task* IRL setting. Their approach differs from ours in two key respects: first, they use a learned dynamics model to learn the cumulants. Second, the learned SFs are used for representing the action-value function, not to inform the behaviour policy with GPI.

Model of others in multi-agent learning. Our method draws inspiration and builds on the multi-agent learning setting, where multiple agents participate in the same environment and the states, actions of others are observed (Davidson, 1999; Lockett et al., 2007; He et al., 2016a; Jaques et al., 2019). However, we do not explore *strategic* settings, where recursive reasoning (Stahl, 1993; Yoshida et al., 2008) is necessary for optimal behaviour.

4.5 Conclusion & discussion

Summary of contributions. In this chapter, we studied reinforcement learning (RL) with sub-optimal (no-reward) demonstrations. Firstly, we introduced a novel and flexible offline multi-task inverse reinforcement learning (IRL) algorithm, called inverse temporal difference (ITD) learning, which discovers salient task-agnostic environment features in the form of cumulants, as well as learning successor features and preference vectors for each agent which provides demonstrations. Secondly, we presented ΨΦ-learning (ΨΦL), which combines ITD learning with ignorance-aware RL from online experience.

Insights & lessons learned. In our experiments §4.3, we showed that ΨΦ-learning is robust to the quality and relevance of the demonstrations. Moreover, we concluded that the role of ignorance quantification and ignorance-averse knowledge equivalents (KEs) is essential for planning from static data of questionable relevance and quality. Lastly, the decomposition of rewards and value functions into cumulants and SFs respectively, enable RL agents that learn online to exhibit fast adapt and modelling of other agents.

Limitations & next steps. ΨΦ-learning is developed with sequential decision-making in a multi-agent *strategic* environment in mind but it is not fully compatible with it. It relies on the assumption that other agents' goals are static and that their policies are stationary, both of which are not in general true (Foerster, 2018). Moreover, while ΨΦ-learning can learn from sub-optimal demonstrations and accelerate RL without requiring access to high quality expert demonstrations, it still requires access to: (i) the demonstrators actions; and (ii) identical action and state space between the learner and the demonstrators. Relaxing these two assumptions will enable learning agents to expand the scope of data sources they can learn from and simplify the data gathering and cleanup phases.

Outlook. One of the fundamental strengths of RL agents is that the do not only learn from their successes but they also learn from their mistakes—capable of bootstrapping from those and self-improving (Sutton and Barto, 2018). But we can do better than solitary trial-and-error, at least in many interesting settings. Most of the real-world environments are multi-agent, and full of other *learners*. There are not only opportunities to compete and collaborate in multi-agent environments but also to *learn*; from others' successes, failures and curricula (i.e., learning trajectories and experiences). As more and more learners (i.e., 'ego-agents' in the terminology of this chapter) are getting deployed, e.g., in the internet or the roads, we expect the to exhibit rapidly adaptive behaviour to the surrounding agents (i.e., 'other-agents), who will adapt to changes in the environment, their beliefs and interact accordingly with the ego-agents. We believe that, solitary RL or sanitised imitation learning (IL) will not be enough to reach this expectations.

"Consistency is the last refuge of the unimaginative."
— Oscar Wilde (1854–1900)

5
Plan with model-value inconsistency

Contents

Using a (learned) model of the environment and a value function, a reinforcement learning (RL) agent can construct many estimates of a state's value, by unrolling the model for different lengths and bootstrapping with its value function, as shown in *Figure 5.1* by varying $k \in \mathbb{N}$. The model-based policy evaluations methods (§2.3.3; Sutton, 1995; Feinberg et al., 2018; Hafner et al., 2019b; Byravan et al., 2020), exploit this observation for constructing value target estimators with reduced variance, e.g., $\text{TD}(\lambda)$ on model-generated rollouts.

Our key insight is that one can treat this set of value estimates as a *kind* of ensemble (see §2.2.3), which we refer to as *implicit value ensemble* (IVE) and hence use it in the same ways that deep ensembles (§2.2.3; Lakshminarayanan et al., 2017) are used in RL. In the context of ignorance-aware RL (see §2.3.5) and sequential-decision making more broadly, which are the focus of this chapter and thesis, we could use the "disagreement" between the IVE estimates as a *proxy* for the agent's ignorance (i.e., epistemic uncertainty). We term this signal *model-value inconsistency* or *self-inconsistency* for short.

[*]This chapter is based on the (Filos et al., 2022) publication.

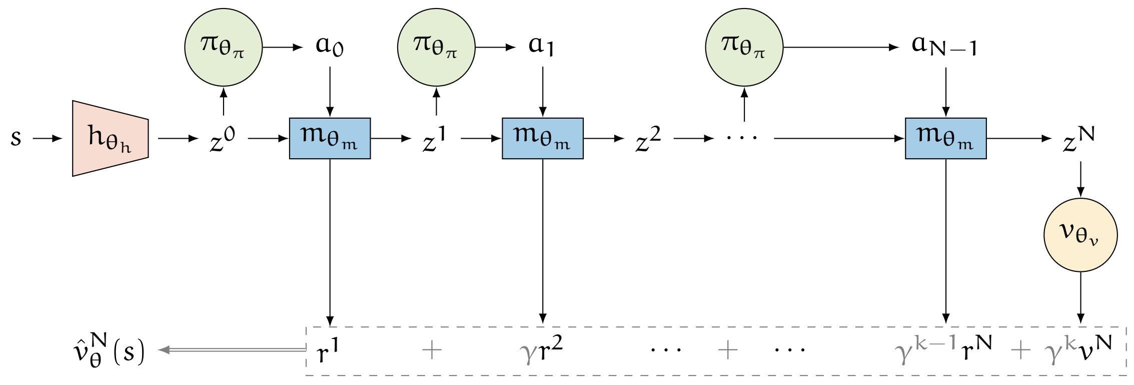


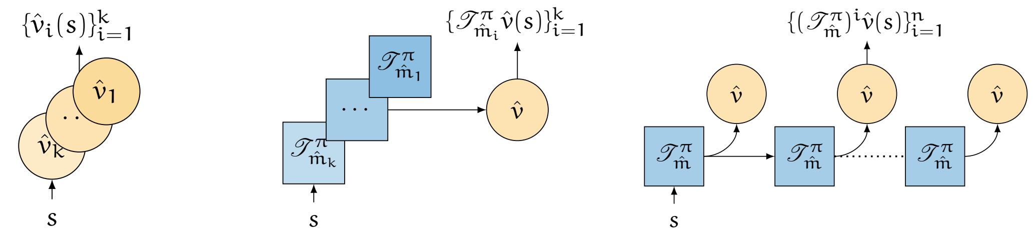
Figure 5.1: The computational graph of the one-sample Monte Carlo (MC) estimator of the k-step model-predicted value (k-MPV), i.e., $\hat{v}^k_\theta(s) = \sum_{i=0}^{k-1}(\gamma^i r^{i+1}) + \gamma^k v^k$.

Unlike prior work which quantifies ignorance by training a regular (i.e., *explicit*) ensemble of many models and/or value functions, IVE requires only a single model and value function, which are already being learned in most model-based RL algorithms. As a result, IVE is more computationally and memory efficient than explicit ensembles. Nonetheless, the sharing of parameters between the ensemble members for IVE inevitably introduces correlation and hence violates any independence assumption between them, which is essential for the theoretical interpretation of the ensemble members as samples from the Bayesian posterior distribution over model parameters (§2.2.3; Wilson and Izmailov, 2020).

Despite the lack of theoretical guarantees, in this chapter, we empirically study the effectiveness of self-inconsistency as a signal for ignorance. In particular, we carry out experiments in both tabular and function approximation settings from pixels (Cobbe et al., 2020; Tunyasuvunakool et al., 2020) and conclude from evidence that self-inconsistency is useful (i) as a signal for exploration, (ii) for acting safely under distribution shifts, and (iii) for robustifying value-based planning with a learned model. The key contributions are:

List of contributions *§5* (planning with model-value inconsistency)***:***

1. **Ignorance quantification for *any* model-base RL agent:** We present a novel signal for quantifying an agent's ignorance about it value function, computable by any model-based RL agent with a single (point) estimate of a world model and a value function. The learned model-induced Bellman operator is applied on the learned value function, generating multiple value estimates, which we term *implicit value ensemble* (IVE). The agent's ignorance is quantified by the disagreement of the IVE components and we call *self-inconsistency* for short.
2. **Match or outperform state-of-the-art ignorance-aware RL agents:** We provide empirical evidence that self-inconsistency provides a proxy of ignorance


$\{\hat{v}_i(s)\}_{i=1}^k$
$\hat{v}_1$
$\hat{v}_k$
s
$\{\mathscr{T}^{\pi}_{\hat{m}_i}\hat{v}(s)\}_{i=1}^k$
$\mathscr{T}^{\pi}_{\hat{m}_1}$
$\mathscr{T}^{\pi}_{\hat{m}_k}$
$\hat{v}$
s
$\{(\mathscr{T}^{\pi}_{\hat{m}})^i\hat{v}(s)\}_{i=1}^n$
$\hat{v}$
$\mathscr{T}^{\pi}_{\hat{m}}$
s

(a) EVE ***(b)*** EMVE ***(c)*** Implicit value ensemble (IVE)

Figure 5.2: Scalable ignorance quantification for reinforcement learning (RL) agents' value function. (a) Explicit ensemble of value functions (EVE, Osband et al., 2016) and (b) world models (EMVE, Chua et al., 2018), approximate samples from $p(v^{\pi}|\mathcal{B})$ and $p(m^*|\mathcal{B})$, respectively. The number of parameters grows linearly with the ensemble size. (c) Implicit value ensemble (IVE) make ensemble value predictions using a single learned value function and world model by exploiting the model-induced Bellman operator $\mathscr{T}^{\pi}_{\hat{m}}$.

(§5.3.1), and that this information can be used to guide exploration or act safely (§5.3.2), and to robustify value-based planning with a learned model (§5.3.3).

5.1 Background & problem setting

We model the agent's interaction with the environment as a Markov decision process (MDP, Puterman, 2014), i.e., $\mathcal{M} \triangleq \langle \mathbb{S}, \mathbb{A}, p, \rho_0, r\rangle$, as in §2.3.1 and *Eqn.* (2.27). We denote the policy parameters with θ_π. We use parametric function approximators, in particular deep neural networks (NNs), to approximate the model and value function: θ_m are the model and θ_v are the value function parameters, i.e., $m_{\theta_m}(\cdot, \cdot|s, a) \approx m^*(\cdot, \cdot|s, a)$ and $v_{\theta_v}(s) \approx v^{\pi_{\theta_\pi}}(s)$. The value function and world model are trained on rollouts $\mathcal{B}$ (see *Eqn.* (2.35)) using standard methods (see §2.3.2), e.g., temporal difference (TD) learning, as in *Eqn.* (2.43), and latent-state reconstruction world modelling, as in *Eqn.* (2.51).

Model-predicted value. The agent performs model-based policy evaluation using the (model-induced) Bellman operators (§2.3.3; Bellman, 1957b; Bertsekas, 2012; Puterman, 2014). We denote with $(\mathscr{T}^{\pi_{\theta_\pi}}_{m_{\theta_m}})^k$ the k-step $(m_{\theta_m}, \pi_{\theta_\pi})$-induced Bellman evaluation[†] operator, see *Eqns.* (2.63, 2.65), i.e., and its application to value estimator v_{θ_v} is given by:

$$v^k_{\theta}(s) \equiv (\mathscr{T}^{\pi_{\theta_\pi}}_{m_{\theta_m}})^k v_{\theta_v}(s) \triangleq \mathbb{E}_{\pi_{\theta_\pi}, m_{\theta_m}}[\sum_{i=0}^{k-1}(\gamma^i R_{i+1}) + \gamma^k v_{\theta_v}(S_k)|S_0 = s], \tag{5.1}$$

[†]In this section, we define everything in terms of the Bellman evaluation operator and an approximate on-policy value function. The Bellman *optimality* operator and an approximate optimal value function could be used instead. For completeness, see §C.3.

where $\mathbb{E}_{\pi_{\theta_\pi}, m_{\theta_m}}[\cdot|S_0 = s]$ denotes the expectation over the trajectories induced by unrolling policy π_{θ_π} in the world model m_{θ_m}, from state s, as defined in *Eqn.* (2.29d). For brevity, we write v_θ^k for the value estimator in *Eqn.* (5.1), where $\theta \triangleq (\theta_\pi, \theta_v, \theta_m)$, and we refer to this quantity as the k-step *model-predicted value* (MPV), or just k-MPV.[†]

In practice, sample-based methods are used to approximate the expectation in *Eqn.* (2.65), e.g., the one-sample Monte Carlo (MC) k-MPV is illustrated in *Figure 5.1* and given by:

$$\hat{v}_\theta^k(s) = \sum_{i=0}^{k-1} (\gamma^i r^{i+1}) + \gamma^k v_{\theta_v}(s^k), \tag{5.2}$$

where $s^0 = s$, $(s^{i+1}, r^{i+1}) \sim m_{\theta_m}(\cdot, \cdot|s^i, a^i)$ and $a^i \sim \pi_{\theta_\pi}(\cdot|s^i)$, as in *Eqn.* (2.66).

> ***Remark 5.1*** (interpolating between model-free and model-based value estimators)***:*** The k-MPV is a value estimator that interpolates between (i) a model-free value estimator, i.e., $k = 0$ and (ii) a purely model-based value estimator, i.e., $k \to \infty$.

Explicit ensemble RL methods. In this chapter, we refer to deep ensembles (§2.2.3; Lakshminarayanan et al., 2017) as *explicit* ensembles to distinguish them from the proposed *implicit* ensemble in §5.2. In particular, the explicit value ensemble (EVE, Lowrey et al., 2018; Osband et al., 2016; Anschel et al., 2017), depicted in *Figure 5.2a*, comprises of N value functions with parameters $\{\theta_{v,n}\}_{n=1}^N$ and its ensemble members are given by:

$$v_{\text{EVE}}^{1:N}(s) \triangleq \{v_{\theta_{v,n}}(s)\}_{n=1}^N. \tag{5.3}$$

The explicit model value ensemble (EMVE, Chua et al., 2018), illustrated in *Figure 5.2b*, comprises of N world models with parameters $\{\theta_{m,n}\}_{n=1}^N$ and a single learned value function with parameters θ_v. Then, one forms ensemble value predictions given by:

$$v_{\text{EMVE}}^{1:N}(s) \triangleq \{v_{\theta_n}^1(s)\}_{n=1}^N, \tag{5.4}$$

where $\hat{v}_{\theta_n}^1(s)$ is the 1-step MPV (see *Eqns.* (5.1, 5.2)) for parameters $\theta_n \triangleq (\theta_{m,n}, \theta_v)$. Both the EVE and the EMVE can be regularly trained, e.g., by following *Remark* 2.7.

5.2 Method

RL agents that quantify their ignorance can (i) explore more effectively; (ii) improve their robustness to distribution shifts; and (iii) stabilise their learning and planning (see §2.3.5). Explicit ensemble methods in §5.1 make use of generic ignorance quantification techniques.

[†]Similar quantities have been used in prior work, e.g., k-preturn (Silver et al., 2017) and MVE (Feinberg et al., 2018). We discuss them and their differences in more detail in §5.4.

5.2.1 Implicit value ensemble (IVE)

In this section, we present a proxy ignorance quantification mechanism for model-based RL agents, which we call *implicit value ensemble* (IVE). IVE exploits the structure of the sequential-decision making problem, as defined in §5.1 and §2.3.1, and, in particular, the Bellman consistency equations (§2.3.3; Bellman, 1957b) to quantify ignorance from point estimates of a world model and a value function, offering computational gains and simpler design than its explicit ensemble counterparts, e.g., EVE and EMVE from §5.1.

Formalisation. We define the IVE as the set of n-MPV estimators (see *Eqn.* (5.1)) for different values of n, i.e., the number of application of the model-induced Bellman operator (see §5.1) on a learned value function. For brevity, we denote the IVE with:

$$v_{\text{IVE}}^{0:N} \triangleq \{v_{\theta}^{n}(s)\}_{n=0}^{N}. \tag{5.5}$$

When the exact computation of the expectation in *Eqn.* (2.58) is viable, e.g., in small and finite MDPs, we can carry out the computation of the IVE members iteratively: First, we compute the 1-MPV using the one-step model-induced Bellman evaluation operator applied on v_{θ_v}, then we compute the 2-MPV by applying the one-step model-induced Bellman evaluation operator on the output of the previous step, etc., until we obtain N-MPV.

Intuition. The model-induced Bellman operators (see *Eqn.* (5.1)) offer us a way to use two seemingly different kinds of learned components—a world model and an approximate value function—to make predictions about the same quantity. In particular, the world model's outputs are distribution's over states and rewards and the output of the approximate value function is an estimate of (state-)value, i.e., the models operate on very different output spaces. Nonetheless, thanks to the Bellman operators (§2.3.3; Bellman, 1957b), all these predictions can be combined to form an ensemble on the (state-)value space. As a result, we say that the members of the IVE form a *heterogeneous* ensemble (Wichard et al., 2003) since they differ in (i) functional form, and (ii) learning algorithm.

> ***Remark 5.2*** (your model-based agent is secretely an ensemble of value functions and you should treat it like one)***:*** Any agent with a model and value function is, in effect, also equipped with an ensemble of value functions.

Practical implementation. In practice, as discussed in §5.1 and §2.3.3, the computation of the k-MPV is intractable and, as a result, we use sample-based approximations to obtain the IVE members. Specifically, we use the one-sample MC estimator, i.e., the n-th member of the IVE is given by *Eqn.* (5.2): We unroll once the policy π_{θ_π} in the world model m_{θ_m} for n steps, starting from state s and bootstrapping with value v_{θ_v}. To compute all the IVE members, we can use the *one-rollout* MC estimator: We unroll the policy π_{θ_π} in the world model m_{θ_m} for N steps, producing the model-generated rollout $(s^n, a^n, r^n)_{n=0}^{N}$.

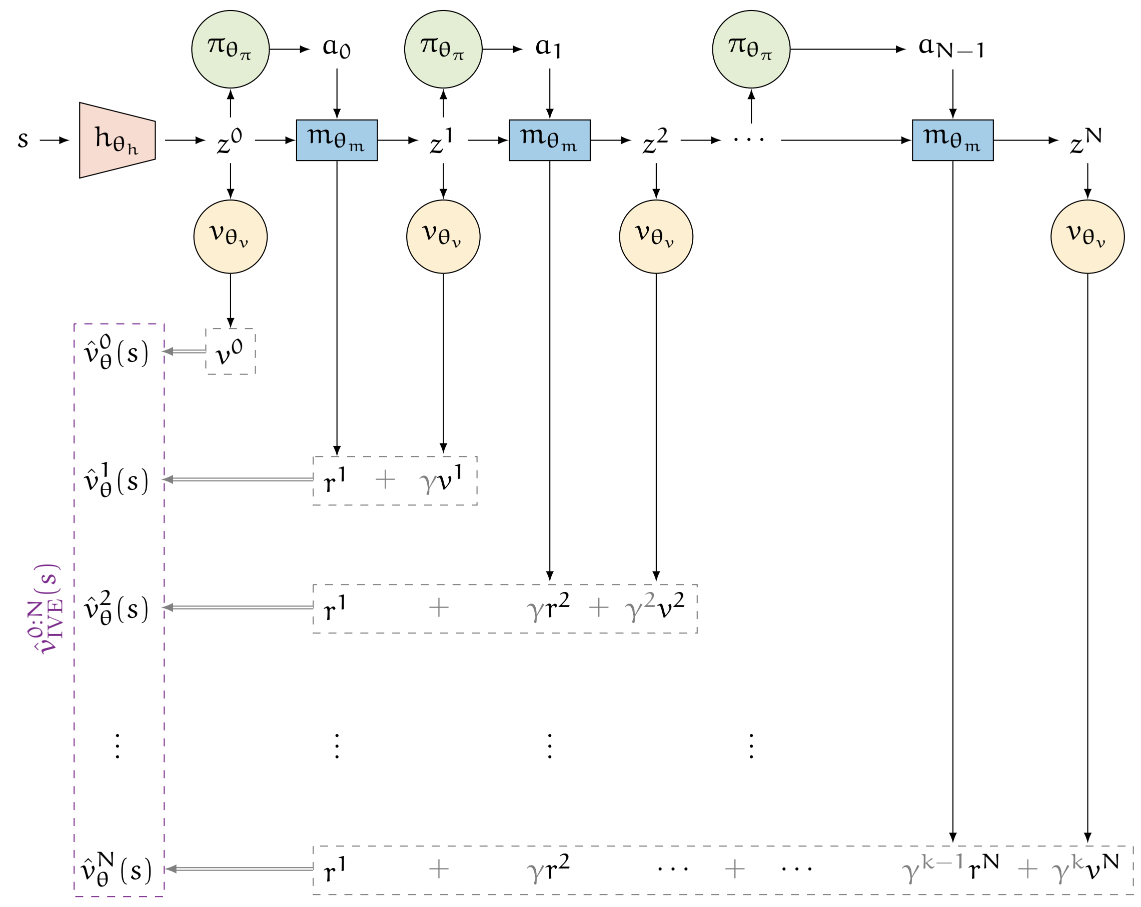


Figure 5.3: The computational graph of the one-rollout Monte Carlo (MC) estimator of the implicit value ensemble (IVE), using point estimates of a world model and value.

Then, the n-th IVE member, i.e., $\hat{v}^n_\theta(s)$, is calculated using *Eqn.* (5.2), i.e., we can get a MC estimate of all the IVE members from a single rollout, as shown in *Figure 5.3*, i.e.,

$$\hat{v}^{0:N}_{\text{IVE}} \triangleq \{\hat{v}^n_\theta(s)\}^N_{n=0}. \tag{5.6}$$

In theory, the IVE members should be highly correlated since: (i) they share the same NN parameters; and (ii) the one-rollout MC estimator further correlates the ensemble members. Although the former source of correlation is inherent to the IVE formulation, the latter could be addressed, e.g., by independently[†] sampling N rollouts from state s: An 1-step long, a 2-step long, ..., a N-step long for estimating $\hat{v}^1_\theta, \hat{v}^2_\theta, \ldots, \hat{v}^M_\theta$, respectively. In §5.3 and §C.4.1 we ablate this decision, arriving to the conclusion that the one-rollout MC estimate's computational efficiency outweighs the small losses due to this correlation.

[†]Use a different pseudo-random number generator seed for each rollout.

5.2.2 Model-value inconsistency

In this section, we present a signal for quantifying ignorance from the IVE. In particular, we term the "disagreement" of the IVE members as *model-value inconsistency* or just *self-inconsistency*, for short, since it quantifies the Bellman-inconsistency (Puterman, 2014; Farquhar et al., 2021) of the learned model and approximate value function. Then, we derive self-inconsistency- (ignorance-) (i) seeking; (ii) averse; and (iii) neutral agents by using appropriate knowledge equivalents (KEs, see §2.1.2).

Formalisation. Various statistics/aggregators can be used to quantify the "disagreement" between the IVE components. Since the n-MPVs are scalars, we can use any measure of disagreement of its components, e.g., the empirical standard deviation across the IVE members (similar to *Eqn.* (3.11a)), denoted by σ-IVE[N] for N members and given by:

$$\sigma\text{-IVE}[N](s) \triangleq \sqrt{\mathrm{Var}[v_{\mathrm{IVE}}^{0:N}(s)]} \overset{(5.5)}{=} \sqrt{\mathrm{Var}[\{v_\theta^n(s)\}_{n=0}^N]}. \tag{5.7}$$

Moreover, drawing inspiration from the ignorance-sensitive KEs from §2.1.2 and §3.2.1, we define μ-IVE(N) as the value prediction, given by the ensemble mean, and $\mu + \beta \cdot \sigma$-IVE(N) as the weighted sum of the IVE mean and standard deviation, where $\beta \in \mathbb{R}$. Therefore, we can use KE (see 2.1.2) and induce a self-inconsistency- (i) seeking; (ii) averse or (iii) neutral agent. When $\beta > 0$, $\beta < 0$ and $\beta = 0$, respectively, i.e.,

$$\mu\text{-IVE}[N](s) \triangleq \frac{1}{N+1} \sum_{n=0}^{N} v_\theta^n(s)\}_{n=0}^N \tag{5.8a}$$

$$(\mu + \beta \cdot \sigma)\text{-IVE}[N](s) \triangleq \mu\text{-IVE}[N](s) + \beta\sigma\text{-IVE}[N](s). \tag{5.8b}$$

Example 5.1 (didactic example for implicit value ensemble (IVE) for a Markov reward process (MRP))***:*** We focus on the prediction problem (Sutton and Barto, 2018), modelled as a MRP with an one-dimensional state space, i.e., $s \in \mathcal{S} = [-3, +3]$ and a discount factor $\gamma = 0.9$. We are provided with state-value target pairs, i.e., $\{(s_i, \bar{v}_i)\}_{i=1}^N$ with $N = 10$ and learn (i) a representation function $h(s; \theta_h)$, (ii) a value function $v(z; \theta_v)$ and (iii) a model $m(\cdot, \cdot | z; \theta_m) \triangleq (r(z; \theta_m), p(z; \theta_m))$, represented as neural networks with parameters, θ_h, θ_v and θ_m, respectively, similar to *Figure 5.1*. In particular:

$$h_{\theta_h}(s) = h(s; \theta_h) = \mathtt{tanh}(\mathrm{MLP}_{\theta_h}(s)) \triangleq z \in [-1, +1]^{32} \tag{5.9a}$$

$$v_{\theta_v}(z) = v(z; \theta_v) = \mathrm{MLP}_{\theta_v}(z) \triangleq v \in \mathbb{R} \tag{5.9b}$$

$$p_{\theta_m}(z) = p(z; \theta_m) = \mathrm{LSTM}_{\theta_m}(z, \mathbf{0}) \triangleq z^1 \in [-1, +1]^{32} \tag{5.9c}$$

$$r_{\theta_m}(z) = r(z; \theta_m) = \mathrm{MLP}_{\theta_m}(z) \triangleq r^1 \in \mathbb{R}, \tag{5.9d}$$

where all the multi-layer percepetrons (MLPs) have one hidden layer of 32 units with an ELU (Clevert et al., 2015) non-linearity and z^k is the (latent) state after taking k steps with the model m, starting from state $z^0 \triangleq z$ (Silver et al., 2017).

We make value prediction by repeatedly applying the m model-induced Bellman operator $\mathscr{T}_m$ on the value function v_{θ_v}, i.e., constructing different k-step model predicted values (k-MPVs, *Eqn.* (5.1)). In particular, the predictions are given by:

$$\hat{v}^0_\theta(s) = (\mathscr{T}_{m_{\theta_m}})^0 v_{\theta_v}(h_{\theta_h}(s)) = v_{\theta_v}(h_{\theta_h}(s)) = v_{\theta_v} \circ h_{\theta_h}(s) \tag{5.10a}$$

$$\hat{v}^1_\theta(s) = (\mathscr{T}_{m_{\theta_m}})^1 v_{\theta_v}(h_{\theta_h}(s)) = (r_{\theta_m} + \gamma v_{\theta_v}) \circ p_{\theta_m} \circ h_{\theta_h}(s) \tag{5.10b}$$

$$\begin{aligned}\hat{v}^2_\theta(s) = (\mathscr{T}_{m_{\theta_m}})^2 v_{\theta_v}(h_{\theta_h}(s)) = \\ (r_{\theta_m} + \gamma(r_{\theta_m} + \gamma v_{\theta_v}) \circ p_{\theta_m}) \circ p_{\theta_m} \circ h_{\theta_h}(s)\end{aligned} \tag{5.10c}$$

$$\vdots$$

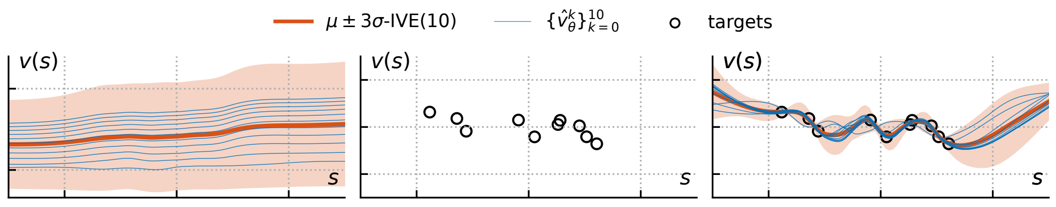
$$\begin{aligned}\hat{v}^k_\theta(s) &= (\mathscr{T}_{m_{\theta_m}})^k v_{\theta_v}(h_{\theta_h}(s)) \\ &= \left(\sum_{j=0}^{k-1} (\gamma^j r_{\theta_m} \circ \underbrace{(p_{\theta_m} \circ \cdots \circ p_{\theta_m})}_{(j+1)\text{-times}}) + \right. \\ &\qquad \left. \gamma^k v_{\theta_v} \circ \underbrace{(p_{\theta_m} \circ \cdots \circ p_{\theta_m})}_{k\text{-times}} \right) \circ h_{\theta_h}(s)\end{aligned} \tag{5.10d}$$

where $\circ$ denotes function composition. Obviously, the k-MPVs with different k have different functional forms, as the predictions at initialisation suggest at *Figures 5.4a & 5.4a*, too. Note that there is no Monte Carlo (MC) sampling—the learned model is deterministic and the policy is implicit.

We learn the neural network parameters $\theta \triangleq (\theta_h, \theta_v, \theta_m)$ using the ADAM (Kingma and Ba, 2014) optimiser with decoupled weight decay (Loshchilov and Hutter, 2017) to minimise the empirical squared value prediction error for all $k \in \{0, \dots, 10\}$, i.e.,

$$\min_\theta \sum_{i=1}^{N} \sum_{k=0}^{K} \|\hat{v}^k_\theta(s_i) - v_i\|_2^2. \tag{5.11}$$

The only source of variability between the k-MPVs (implicit value ensemble (IVE) members) is their functional form, induced by different compositions of the learned parametric networks v_{θ_v}, p_{θ_m} and r_{θ_m} as *Eqn.* (5.10) shows.



(a) At initialisation ***(b)*** Value targets ***(c)*** After training

Figure 5.4: A value prediction problem of an implicit policy, modelled as a Markov reward process (MRP, Sutton and Barto, 2018) with an one-dimensional state space, i.e., $s \in \mathbb{S} = [-3, 3]$. We learn a model $\hat{m}$ and a value function $\hat{v}$ and construct a k-step model predicted value (k-MPV, *Eqn.* (5.1)) by applying the model induced Bellman operator $\mathscr{T}_{\hat{m}}$ repeatedly k times on the learned value function v_{θ_v}, i.e., $\hat{v}_\theta^k(s) \triangleq (\mathscr{T}_{\hat{m}})^k v_{\theta_v}(s)$. We visualise the k-MPVs, a.k.a components of the *implicit value ensemble* (IVE, *Eqn.* (5.5)) for $k \in \{0, \dots, 10\}$ (in blue) along with the ensemble mean and standard deviation (in orange), constructed from a single (point) estimate of the value function and model. (a) The predictions at initialisation, i.e., before training. (b) The data, i.e., state and value target pairs. (c) The predictions after training every IVE member towards the value targets in (b), i.e., $\min_{m,v} \sum_i \sum_k \|\hat{v}_\theta^k(s_i) - v_i\|_2^2$. We observe in (c) that the ensemble components fit the value targets and their standard deviation is zero at and around the observed (in-distribution) data but it is non-zero otherwise (out-of-distribution points). Therefore the IVE members' disagreement can be used as a signal for epistemic uncertainty. In this example, the variability between the IVE members' predictions is only due to their different functional forms.

Intuition. As our learned model and value function better approximate their "true" counterparts, the self-inconsistency reduces since the "true" model and value function are Bellman consistent. Therefore, we could treat self-inconsistency as a proxy signal for modelling error but with a caveat. When the learned model and approximate value function are self-consistent, they are not necessarily correct, too, since trivial approximations, such as the zero-everywhere value and reward functions, are always self-consistent. Nonetheless, a self-inconsistent pair is by definition inaccurate. Therefore we arrive to the following rule of thumb for interpreting self-inconsistency.

Remark 5.3 (self-inconsistency on in- and out-of-training distribution (OOD) states)***:*** In regions of state space where the learned model and value function are accurate, they are also self-consistent. With high self-inconsistency the learned model or/and value should be inaccurate.

5.3 Experiments

We conduct a series of tabular and deep RL experiments[†] to determine how effective model-value inconsistency is as a signal for ignorance. Our goal is *not* to show that the IVE is better than explicit ensembles. Instead, since IVE is present in any model-based RL agent, we want to empirically study its properties and validate its usefulness.

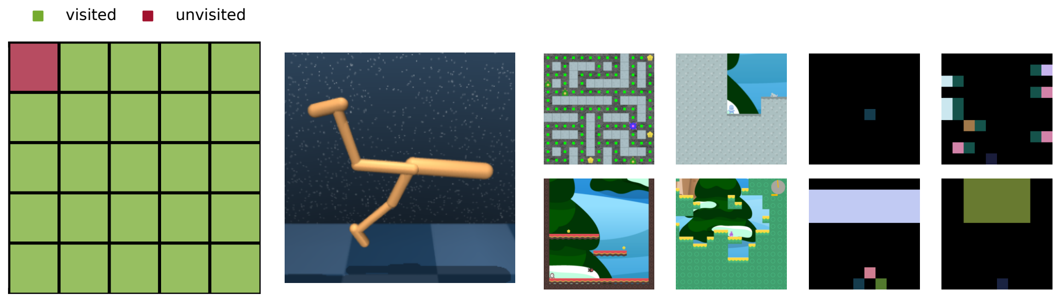
Baselines. In the tabular experiments, we learn value functions with expected SARSA (Van Seijen et al., 2009) and use maximum likelihood estimation (MLE) for model learning (see §2.3.2). The explicit ensemble components are trained according to *Remark* 2.7.

In the deep RL experiments, we built on the following model-based agents: (i) **Muesli** (Hessel et al., 2021) is a policy optimisation method with a learned multi-step expectation model (see *Eqn.* (2.54)). Muesli also learns a state-value function, using Retrace (Munos et al., 2016) to correct for the off-policiness of the replayed experience. The learned model is used for representation learning and for constructing action-value estimates, by one-step model unroll, used for policy improvement. The model parameters are trained to predict reward and value k-step into the future (corresponding to the individual terms in the k-MPV); (ii) **Dreamer** (Hafner et al., 2019a) is a policy optimisation method with an MLE (i.e., reconstruction-based) world model (see *Eqn.* (2.51)). The model is an action-conditioned hidden Markov model, trained to maximise (a lower bound on) the likelihood of the reward and observation sequences. Dreamer learns a value function using *only* rollouts from the learned model and its parameters are learned such that the learned value function becomes (self-)consistent with the model; (iii) **VPN** (Oh et al., 2017) is a value-based planning method with a multi-step expectation model. The action-value function and model are trained simultaneously with n-step Q-learning (Watkins and Dayan, 1992). In this case, the k-MPV is the value estimate after applying k times the model-induced Bellman *optimality* operator on the learned value function (see §C.3 for a formal exposition).

Environments. In the tabular experiments, we use an empty 5×5 `GridWorld`, and collect data by rolling out a uniformly random policy, initialised at the bottom right cell. We exclude from the dataset any transitions to the top left cell, as illustrated in *Figure 5.5a*, controlling for visited (in-distribution) and unvisited (out-of-training distribution) states.

In the deep RL experiments, we use a selection of 5 tasks from the `Procgen` suite (*Figure 5.5c*; Cobbe et al., 2020) to (i) control the number of distinct levels used for training the agent (i.e., `#levels`) and (ii) hold out a set of test levels that are not seen during training. We also use a modification of the `walker` walk task from the DeepMind Control suite (*Figure 5.5b*; Tunyasuvunakool et al., 2020). The original `walker` task has a per-step reward r_t bounded in $[0, 1]$ which is computed based on the agent's torso height and

[†]Further experiments, details on the experimental protocol and implementations can be found in §C.4, §C.1 and §C.2, respectively.



(a) `GridWorld` *(b)* `walker` *(c)* `Procgen` *(d)* `MinAtar`

Figure 5.5: Environments studied in this chapter. ***(a)*** Small and finite MDP for tabular experiments. ***(b)*** Continuous control from pixels. ***(c)*** Procedurally generated suite of pixel-based tasks. ***(d)*** Small-scale ALE-inspired (Bellemare et al., 2013) tasks.

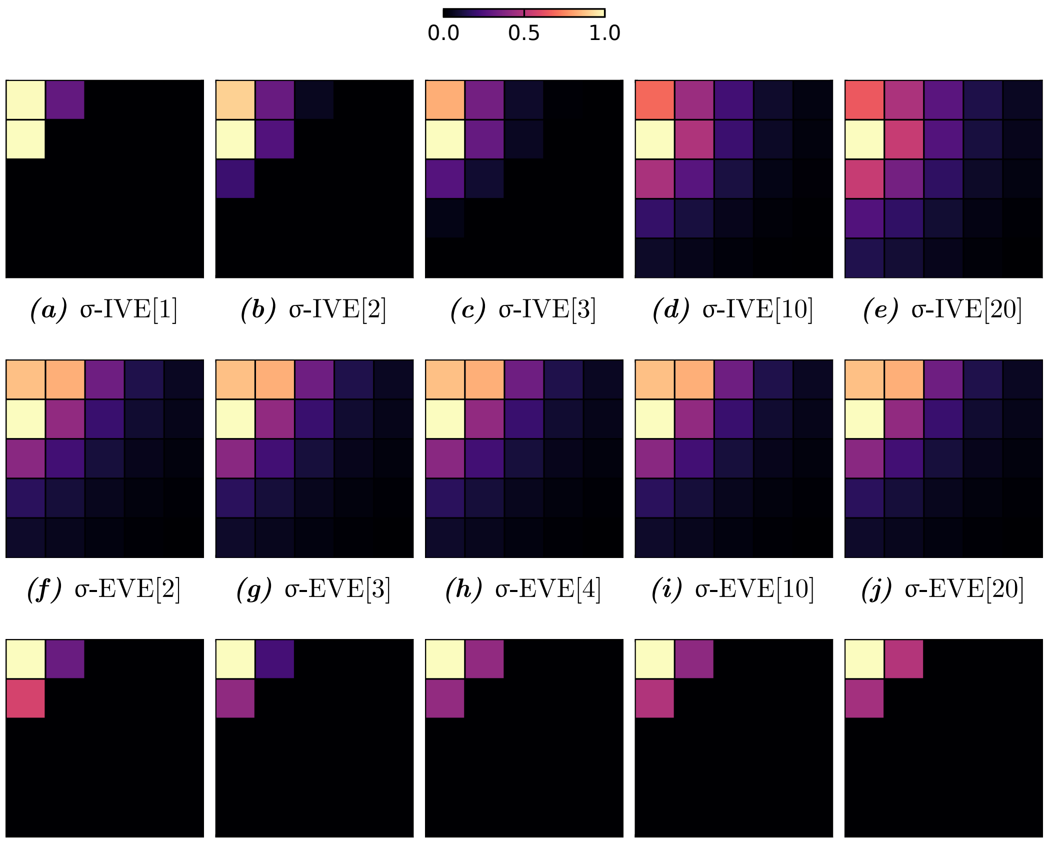
forward velocity. To parameterise exploration difficulty, we modify the reward function to set any reward less than η to zero: $\tilde{r}_t = \mathcal{H}(r_t - \eta) r_t$, where $\mathcal{H}$ is the Heaviside step function. For large $\eta \in \mathbb{R}^+$, agents that rely on naive exploration methods will struggle to find rewards and solve the task. Lastly, we use the original `MinAtar` (*Figure 5.5d*; Young and Tian, 2019) suite for fast experimentation with value-based agents (Mnih et al., 2013).

5.3.1 Detecting out-of-distribution regimes with self-inconsistency

Based on the proposed role of self-inconsistency as a signal for ignorance, and how ignorance changes between in- and out-of-training distribution (OOD) regimes, we expect the following hypotheses to hold. **H1**: Self-inconsistency is low in in-distribution regions of the state-action space. **H2**: Self-inconsistency is high in OOD regions. **H3**: Self-inconsistency in an OOD test distribution is reduced by bringing the training distribution closer to it.

Tabular. *Figures 5.6a-5.6e* show the self-inconsistency, measured as σ-IVE[N] for different values of N, in the tabular `GridWorld`. As N grows from 1 to 20, the standard deviation across the IVE is qualitatively similar to the EVE's in *Figure 5.6j*. We observe that the self-inconsistency is lower for visited states (**H1**) than unvisited (OOD) ones (**H2**).

Deep RL. *Figure 5.7a* shows the Muesli agent's performance and *Figure 5.7b* its self-inconsistency—calculated after training as the σ-IVE[5]—for the different `Procgen` tasks and for varying training `#levels`, after 100M environment steps. The self-inconsistency for the training (in-distribution) levels is always low, regardless of the `#levels` used for training the agent (**H1**). We also observe that the self-inconsistency in the test (OOD) levels is higher than the train ones (**H2**). Importantly, as the number of training levels increases the self-inconsistency on the test levels decreases, which confirms **H3**. Also as expected, this reduced self-inconsistency correlates with improved test performance.



(a) σ-IVE[1] *(b)* σ-IVE[2] *(c)* σ-IVE[3] *(d)* σ-IVE[10] *(e)* σ-IVE[20]

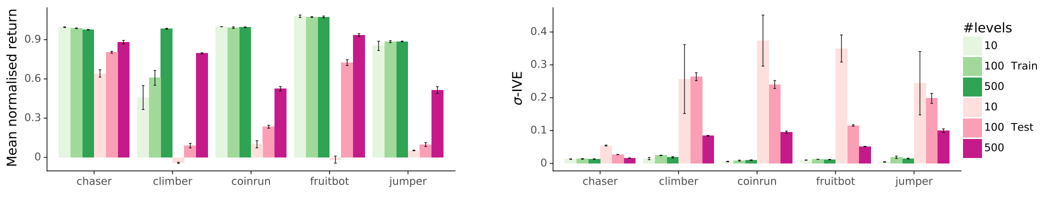
(f) σ-EVE[2] *(g)* σ-EVE[3] *(h)* σ-EVE[4] *(i)* σ-EVE[10] *(j)* σ-EVE[20]

(k) σ-EMVE[2] *(l)* σ-EMVE[3] *(m)* σ-EMVE[4] *(n)* σ-EMVE[10] *(o)* σ-EMVE[20]

Figure 5.6: Standard deviation across value ensembles for different numbers of ensemble components N, trained on offline data from `GridWorld` environment in *Figure 5.5a*, excluding the top left state from the training data. ***(a-e)*** Implicit value ensemble (IVE) (ours, see *Figure 5.2c*), where we vary the length of the rollout used to estimate the members, as defined in *Eqn.* (5.6). ***(f-j)*** Explicit value ensemble (EVE) (ours, see *Figure 5.2a*), where we vary the number of value networks. ***(k-o)*** Explicit model value ensemble (EMVE) (ours, see *Figure 5.2b*), where we vary the number of world model networks. The standard deviation σ is normalised in range $[0, 1]$ per figure.

5.3.2 Optimism and pessimism in the face of self-inconsistency

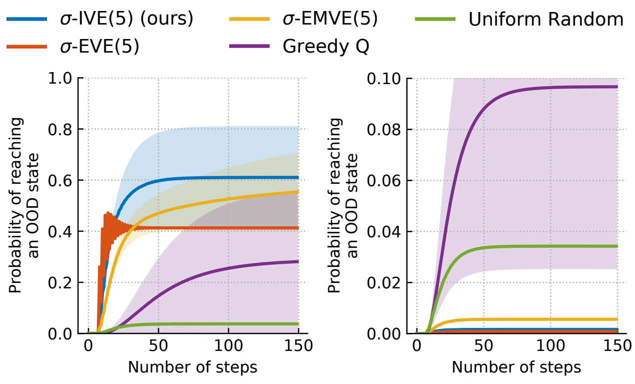
In §2.1.2 and §2.3.5, we showed that decision-makers under uncertainty: (i) seek ignorance to drive exploration (Sekar et al., 2020) and (ii) avoid it for acting safely (§3; Filos et al., 2020). This section addresses two hypotheses. **H4**: Self-inconsistency is an effective signal for exploration. **H5**: Avoiding self-inconsistency leads to robustness to distribution shifts.



(a) Episodic returns *(b)* Self-inconsistency

Figure 5.7: Self-inconsistency as a signal for distribution shifts. (a) Normalised training and test performance for a Muesli agent evaluated on both training and unseen test levels of 5 `Procgen` games after 100M environment frames, for different numbers of unique levels seen during training. Values are normalised by the min and max scores for each game. (b) σ-IVE(5) computed using the model of the Muesli agent while evaluating on both training and unseen test levels, for different numbers of unique levels seen during training. Bars, error-bars show mean and standard error across 3 seeds, respectively.

Tabular. *Figure 5.8a* shows the probability of reaching the novel state in `GridWorld` when a self-inconsistency-seeking policy is followed (+σ-IVE). Seeking self-inconsistency improves upon a uniformly random or greedy policy and is on par with an explicit ensemble of values (EVE) method (**H4**). For the experiment in *Figure 5.8b*, a distribution shift is performed by raising the environment stochasticity from $\delta = 0.1$ to $\delta = 0.5$, and the probability of a self-inconsistency-avoiding policy (-σ-IVE) is illustrated. We observe that the self-inconsistency-avoiding policy is robust to the drift of the environment dynamics (**H5**).



(a) Optimism *(b)* Pessimism

Figure 5.8: Probability of reaching the out-of-distribution state in a tabular `GridWorld`, starting from the bottom right cell (Figure 5.5a) by (a) seeking or (b) avoiding self-inconsistency (σ-IVE, see §5.2.2) or explicit value or model ensemble (EVE, EMVE) standard deviation. Error bars show standard error over 100 seeds.

Deep RL. *Table 5.1* gives the performance of the Dreamer agent and variants that use the model for online planning (Ma et al., 2020) as we increase reward sparsity for the `walker` task, e.g., $\eta = 0$ is the original task and $\eta = 0.5$ sets rewards below 0.5 to zero. We used the mean of IVE components in place of the learned policy for acting (μ-IVE[5]), and combined the mean with the self-inconsistency signal for acting optimistically in the face of uncertainty (μ + σ-IVE[5]). The self-inconsistency-seeking Dreamer-variant,

Table 5.1: Pixel-based continuous control experiments. Results for the Dreamer (Hafner et al., 2019a) agent and IVE variants on a modified version of the Walker Walk task with varying degrees of reward sparsity controlled by η, where higher η corresponds to harder exploration. A "$\diamondsuit$" indicates methods that use online-planning for acting. We report mean and standard error of episodic returns (rounded to the nearest tenth) over 3 runs after 1M steps. Higher-is-better and the performance is upper bounded by 1000. The **best performing** method, per-task, is in bold.

Methods	$\eta = 0.0$	$\eta = 0.2$	$\eta = 0.3$	$\eta = 0.5$
Dreamer	**1000**±00	720±10	570±60	80±50
μ-IVE[5]$^{\diamondsuit}$	**1000**±00	860±40	690±70	210±60
$\mu + \sigma$-EVE[5]$^{\diamondsuit}$	**1000**±00	**1000**±00	**980**±10	**280**±50
$\mu + \sigma$-EMVE[5]$^{\diamondsuit}$	**1000**±00	910±20	730±40	210±60
$\mu + \sigma$-IVE[5]$^{\diamondsuit}$	**1000**±00	**1000**±00	**1000**±00	**330**±70

i.e., $\mu + \sigma$-IVE[5], is performing well for $\eta = 0.3$ and $\eta = 0.5$ while the base agent fails, corroborating **H4**. Similar to the tabular experiment results, the IVE is on par with the the explicit value ensemble (EVE, *Figure 5.2a*) and outperforms the explicit model value ensemble (EMVE, *Figure 5.2b*).

5.3.3 Planning with averaged model-predicted values

Bayesian model averaging (BMA), i.e., integrating over ignorance for making predictions—linear knowledge equivalent (KE), has been used to boost performance (Wilson and Izmailov, 2020). The interpretation of the IVE as an ensemble allows to justify prior methods in the literature that have argued for averaging MPVs (Oh et al., 2017; Byravan et al., 2020) in order to robustify value-based planning, casting them as approximate BMA methods. This section addresses one hypothesis: **H6**: Ensemble averaging of the IVE members is *in general* more robust for value prediction than any component individually.

Deep RL. *Table 5.2* shows the final performance of a VPN(5) agent that uses μ-IVE[5] value targets and its $\hat{v}^1_\theta$ and $\hat{v}^5_\theta$ variants' on the `MinAtar` tasks. The ensembled μ-IVE[5] value predictor is consistently better than the single value predictors, supporting **H6**.

5.4 Related work

The literature review for model-free and model-based reinforcement learning (RL), knowledge equivalents (KEs), ignorance quantification in deep learning (DL) and deep en-

Table 5.2: Value-based planning experiments on `MinAtar` tasks, testing the impact of planning with the IVE ensembled mean. The original VPN(5) (Oh et al., 2017) is the same with our μ-IVE(5). Non-ensembled value targets ($\hat{v}_\theta^1$, $\hat{v}_\theta^5$) lead to significant deterioration in final performance. We report mean and standard error of episodic returns over 3 runs after 2M steps, higher-is-better. The **best performing** method, per-task, is in bold.

Methods	`Asterix`	`Breakout`	`Freeway`	`Seaquest`	`Space Invaders`
DQN	14.7±0.4	12.1±1.2	**49.6**±0.3	2.3±0.6	47.2±1.3
VPN+$\hat{v}_\theta^1$	15.1±0.6	13.8±0.8	**49.1**±0.7	4.7±0.9	53.9±1.8
VPN+$\hat{v}_\theta^5$	7.1±2.3	4.2±2.3	24.3±4.2	1.2±1.4	28.6±8.3
μ-IVE(5)	**18.3**±0.2	**22.0**±0.7	**49.4**±0.5	**8.6**±0.3	**97.3**±9.6

sembles can be found in §2. In the section, we review the use of model-predicted value (MPV) estimators, self-consistency signals and implicit ensembles in the context of deep learning (DL), to better contextualise the contribution of this chapter.

Model-predicted value. The single-sample Monte Carlo (MC) estimator of k-MPV in *Eqn.* (5.2) and *Figure 5.1*, a.k.a. model value expansion (Feinberg et al., 2018) has been used for constructing value targets (see §2.3.2). In particular, Feinberg et al. (2018); Buckman et al. (2018); Byravan et al. (2020) follow a two-step process: (i) they learn a (latent-state) world model by reconstruction (see *Eqn.* (2.51)) and then (ii) learn the value function by regressing it to MPV predictions/targets. Oh et al. (2017); Silver et al. (2017); Farquhar et al. (2017); Gregor et al. (2019); Schrittwieser et al. (2020); Nikishin et al. (2021) train the model and value function jointly, with a direct regression loss on the MPV. Both the IVE and self-inconsistency signal are compatible with these learning approaches.

Adapting k. With varying k, MPV interpolates between the learned model and value function. In particular, for (i) $k = 0$ the value predictions are based only on the learned value function and for (ii) $k \to \infty$ only the learned model contributes to the value predictions. The λ-predictron (Silver et al., 2017) uses a learned and adaptive mechanism for mixing the predictions for different ks. STEVE (Buckman et al., 2018) is an ignorance-informed mechanism for weighting the different MPVs. It learns an explicit ensemble of models and value functions and weights the MPV using an inverse variance weighting of the means, calculated across the *explicit* ensemble. This should not be confused with our σ-IVE(n) signal, which is the variance across the MPVs and cannot be used for selecting the "best" k-th element but quantifies the model-value disagreement.

Novelty signals. Non-explicit ensemble methods have been proposed for estimating the model prediction error and use this as a proxy signal for novelty. Most of these methods make novelty predictions for a state s_t, *after* observing a transition $s_t \xrightarrow{a_t} s_{t+1}$ (Stadie et al., 2015; Pathak et al., 2017; Raileanu and Rocktäschel, 2020) and therefore are termed *retrospective novelty predictors* in the literature (Sekar et al., 2020). Lopes et al. (2012) assume that the agent's learning progress is a predictable process and fit a model to it. While (s_t, a_t, s_{t+1}) triplets are necessary for training the novelty predictor, after training, the signal can be calculated *before* observing s_{t+1} and hence can be used for planning purposes, which we term a *plannable novelty predictor*. The σ-IVE signal can be interpreted as a prediction error estimate that quantifies how the learned value function and model disagree in their predictions and hence we can use it as a plannable novelty signal.

Self-consistency regularisation. Silver et al. (2017) and Farquhar et al. (2021) regularised their learned value and model pairs to be self-consistent for prediction and control tasks, respectively. Self-consistency regularisation has been used for learned world models by matching the predictions of a forward dynamics model with a backward dynamics model (Yu et al., 2021). Similar regularisation ideas have been used in other areas of machine learning, including offline multi-task inverse RL (§4; Filos et al., 2021), natural language processing (Bojar and Tamchyna, 2011; Edunov et al., 2018) and generative modelling (Zhu et al., 2017). All prior work directly "forces" (i.e., trains for) self-consistency on modelled quantities as a form of regularisation, e.g., applied on imagined data (Farquhar et al., 2021). Instead, we treat self-inconsistency as a proxy for ignorance (i.e., epistemic uncertainty) and, e.g., indirectly promote self-consistency by actively guiding data collection/exploration with a self-inconsistency-seeking policy (see §5.3.2). Consequently, this avoids degenerate but self-consistent solutions since the learned model and value functions are trained on real data (i.e., external consistency).

Implicit ensembles. Ensembles from a single neural network (NN) have been proposed and successfully used in supervised learning but they require modifications to the learning algorithm (Huang et al., 2017; Maddox et al., 2019; Antorán et al., 2020) or architecture (Huang et al., 2016; Dusenberry et al., 2020). In contrast, IVE relies on the structure of the RL problem and leverages the Bellman consistency (Puterman, 2014) that the "true" model and value function satisfy and hence their learned counterparts should also do.

5.5 Conclusion & discussion

Summary of contributions. In this chapter, we studied ignorance quantification in reinforcement learning (RL). We introduced a proxy signal for capturing ignorance, called model-value inconsistency or just self-inconsistency, which we found emperically useful for: (i) guiding exploration; (ii) increasing an agent's ability to handle distribution shifts; and (iii) robustify value-based planning methods. Our key insight is that a *single* (point) estimate of a world model and value function can be used to generate multiple estimates of the state value, termed model-predicted values (MPVs), which can be combined to form an implicit value ensemble (IVE). The "disagreement" amongst the IVE members, e.g., quantified by the ensemble standard deviation, is the self-inconsistency signal.

Insights & lessons learned. In our experiments §5.3, we showed that self-inconsistency is low in in-distribution and high in out-of-training distribution (OOD) regions of the state-action space, making it a valid proxy signal for ignorance. Moreover, we can steer the agent's behaviour by optimising for ignorance-sensitive, i.e., self-inconsistency-seeking or -averse, knowledge equivalents (KEs), i.e., making it exploratory or conservative to novelty. Also, in agreement with prior work (Anschel et al., 2017), treating the IVE as an ensemble and planning with the ensemble mean leads to more stable and performant RL agents.

Limitations & next steps. The computationally efficient one-rollout Monte Carlo (MC) estimator for the IVE members is susceptible to risk (e.g., aleatoric uncertainty). Therefore, for highly stochastic environments, it is not possible to disentangle risk from ignorance from the estimated self-inconsistency signal. Hence alternative approximations need to be developed, preserving the computational benefits of IVE over explicit ensemble methods but without compromising performance. Moreover, new self-inconsistency-guided planning algorithms can be developed. For example, Monte Carlo tree search (MCTS, Coulom, 2006; Silver et al., 2016) at its "expansion" step it calculates a MPV and at its "backpropagation" step it aggregates the expanded values to the parent leaves of the tree. The latter step is an incremental way of computing the IVE ensemble mean. Similarly, an incremental calculating of the IVE ensemble standard deviation could be possible, giving rise to an ignorance-aware MCTS variant, efficiently implemented with self-inconsistency. Lastly, generalising self-inconsistency to non-RL settings could be promising.

Outlook. In §4 and §4.5, we highlighted the importance of learning from unstructured data sources, leveraging as much knowledge as possible from the environment and the agents in it. We can think of it as a form of *external* consistency, i.e., how the learning agents beliefs change to match the experiences and observations from the external world/environment, which (possibly) includes the other agents too. Although this is powerful and provides grounding to external sources of information, we expect that generally capable agents should be *internally* consistent, too. In this chapter, we demonstrated that internal/self-consistency can be connected to an agent's ignorance and leveraged accordingly, leading to empirically more performant agents. We believe that internal consistency can be a guiding mechanism for self-improvement, which learning agents can leverage to learn without any external percepts. We will not go as far as Plato (Scott, 2006) and state that *"all learning is recollection"* but we are comfortable to claim that *some* learning is recollection. Also, the AI research community starts to explore the deeper connections between (self-)consistency and rationality that philosophers have studied (Schick, 1963).

"We can only see a short distance ahead, but we can see plenty there that needs to be done."

— Alan Turing (1912–1954)

6 Afterword

In this thesis, we motivated, proposed and investigated a number of ignorance-aware planning agents that focus on different aspects of sequential decision-making and learning. These include: (i) learning from expert demonstrations and acting robustly in novel out-of-training distribution situations (*imitation learning*, §3); (ii) learning from sub-optimal demonstrations to accelerate solitary trial-and-error reinforcement learning (*social learning*, §4); and (iii) learning efficiently and steadily from trial-and-error (§5). We verified our intuitions in small scale settings, e.g., tabular environments, and showed that our methods scale gracefully to larger scale settings, e.g., simulated urban autonomous driving from high-dimensional LIDAR observations and continuous control from pixels.

In particular, in §3, we demonstrate ignorance-unaware imitation learning methods' brittleness in out-of-training distribution autonomous driving settings and provide the first instance of a robust ignorance-aware imitation learner. In §4, we introduce the first offline inverse reinforcement learning-based social learner, which seamlessly integrates trial-and-error learning and sub-optimal demonstrations (without reward information) to rapidly adapt to challenging autonomous driving and pixel-based video games. In §5, we presented the first performant (explicit) ensemble-free ignorance-aware reinforcement learner, capable of solving challenging sparse-reward continuous control tasks from pixels.

While we provide supporting evidence for the importance of ignorance-awareness in planning agents and some instances of such agents that either challenge or outperform the state-of-the-art, some open challenges remain, and we outline and discuss them next.

Ignorance as a first-class citizen. We have adopted a mechanistic view of ignorance in this thesis. We associate it with the posterior distribution over models upon observing some data. While this is an actionable view, which allow us to make progress, it is not fully integrated in the problem definition of the sequential-decision making problem. The learning agent is not trained with the notion of being able to take an action "I do not know", instead there is a post-hoc/meta-mechanism that operates at the posterior distribution and makes such decisions, often based on heuristics. We argue that designing agents (and environments) in which ignorance is treated as a "first-class citizen" can have profound impact on the way we think about and deal with unknowns (Li et al., 2008).

Temporally abstract planning and reasoning about ignorance. As we have seen repeatedly in this thesis, planning with a learned model offers us the opportunity to lookahead for making a decision now, informed by potential realisations of future events. Nonetheless, both the methods we presented here and the state-of-the-art planning agents (Schrittwieser et al., 2020; Hafner et al., 2020) plan at the "atomic" (i.e., low-level) action space (e.g., 'turn left', 'turn right', 'lift this joint by 1cm'), limiting the scope of possibilities. By extensions, these agents can neither quantify nor make use of longer-term ignorance. We believe that generally capable agents should be able to reason about their ignorance in different time scales and incorporate these into their decision-making.

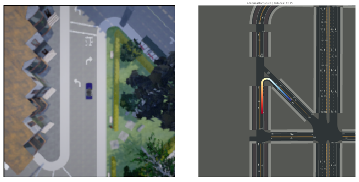
Continual learning & target shifts. Generally capable agents should be able to be robust and adaptable to novel settings. We distinguish novelty with respect to (i) "input" variables, e.g., states and plans, and (ii) "output" variables, e.g., evaluations (Lipton et al., 2018). In this thesis, we have focused on how planning agents can combat the former, i.e., *distribution shifts.* However, the latter, i.e., *target shifts*, are very challenging and especially relevant for planning agents, as well. For example, the arguably simplest planning method, *value iteration* (Bellman, 1957b), even under no distribution shifts, undergoes target shifts since the value estimates (targets) for the *same* states and action pairs change throughout learning, i.e., they are non-stationary (Ring et al., 1994). The problem is exacerbated when neural network function approximators are used due to their susceptibility to continually changing (non-stationary) targets (McCloskey and Cohen, 1989; Sahoo et al., 2017). We believe that one of the great challenges for building the next generation of agents would be to quantify and incorporate ignorance into planning under *target* shifts. Although some of the insights and methods proposed in this thesis may be useful to address target shifts, further innovations in deep learning methods and reinforcement learning algorithms may be necessary to advance this research direction.

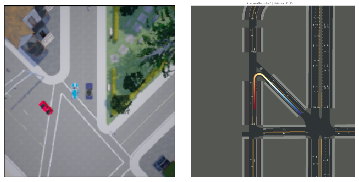
We believe that making progress on the above challenges, as well as the ones mentioned in the discussion sections of §3.5, 4.5 and 5.5, has the potential to transform the next generation of artificial agents and contribute to further our pursuit for full autonomy.

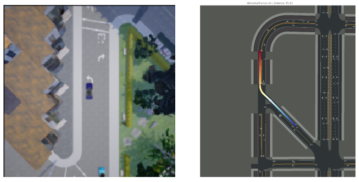
A

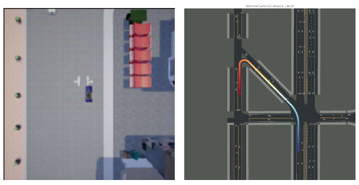
Plan and adapt from expert demonstrations

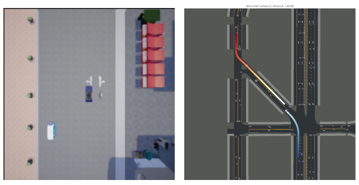
A.1 `CARNOVEL`: Suite of tasks under distribution shift

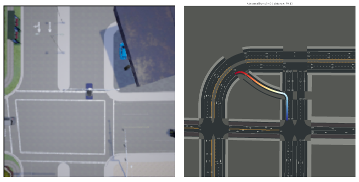

(a) `AbnormalTurns0-v0`

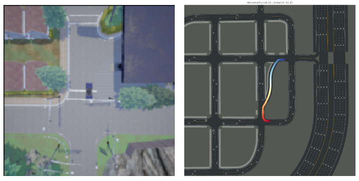

(b) `AbnormalTurns1-v0`

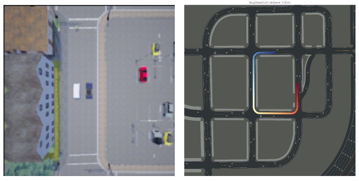

(c) `AbnormalTurns2-v0`

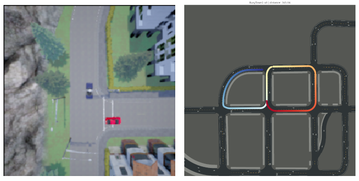

(d) `AbnormalTurns3-v0`

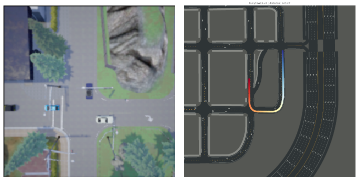

(e) `AbnormalTurns4-v0`

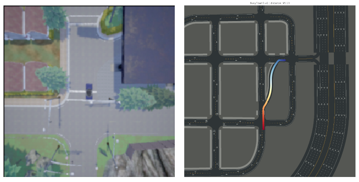

(f) `AbnormalTurns5-v0`

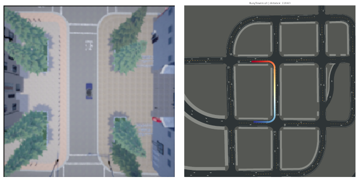

(g) `AbnormalTurns6-v0`

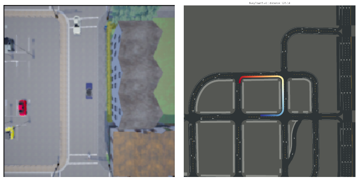

(h) `BusyTown0-v0`

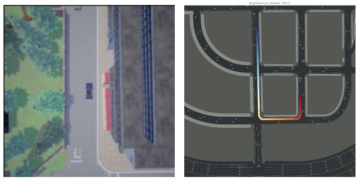

(i) `BusyTown1-v0`

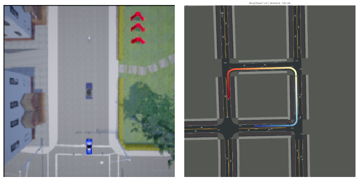

(j) BusyTown2-v0

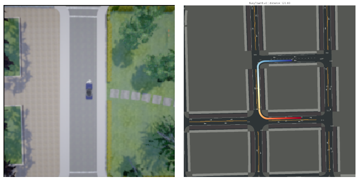

(k) BusyTown3-v0

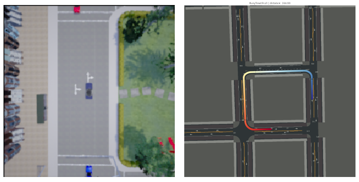

(l) BusyTown4-v0

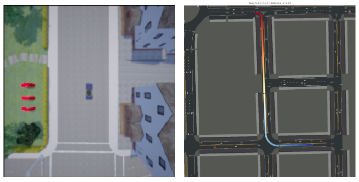

(m) BusyTown5-v0

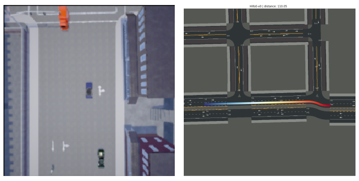

(n) BusyTown6-v0

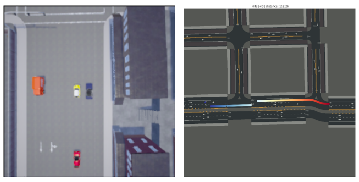

(o) BusyTown7-v0

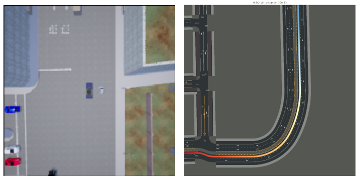

(p) BusyTown8-v0

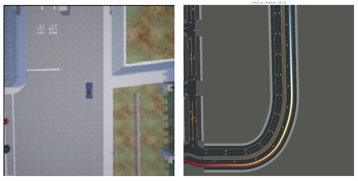

(q) BusyTown9-v0

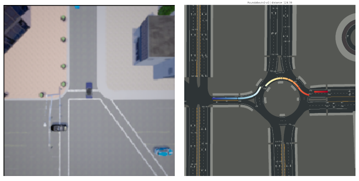

(r) BusyTown10-v0

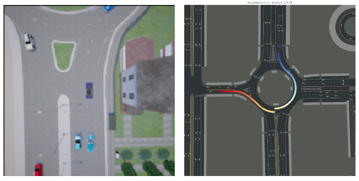

(s) Hills0-v0

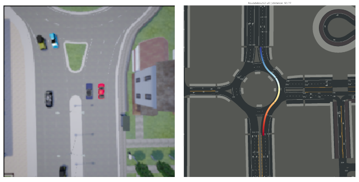

(t) Hills1-v0

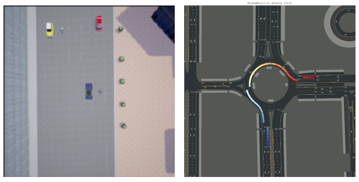

(u) Hills2-v0


(v) Hills3-v0


(w) Roundabouts0-v0


(x) Roundabouts1-v0


(y) Roundabouts2-v0


(z) Roundabouts3-v0

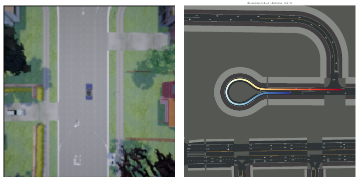

(aa) Roundabouts4-v0

Figure A.1: The spawn location and the route for completing each CARNOVEL task.

A.2 AdaRIP examples

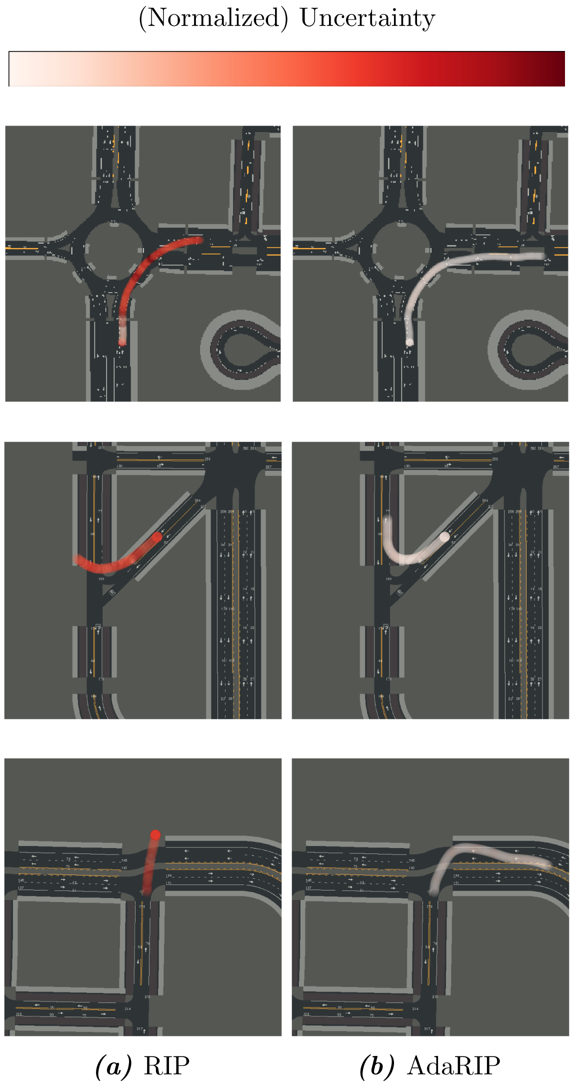


Figure A.2: Examples where the non-adaptive method (a) fails to recover from a distribution shift, despite it being able to detect it. The adaptive method (b) queries the human driver when uncertain (dark red), then uses the online demonstrations for updating its model, resulting into confident (light red, white) and safe trajectories.

B

Plan and adapt from sub-optimal demonstrations

B.1 Experimental details

In this section we describe the environments used in our experiments (see §4.3) and the experiment design.

B.1.1 `Highway`

We build on the `highway-v0` task from the `highway-env` traffic simulator (Leurent, 2018). The task is specified by:

1. **State space, $\mathbb{S}$:** The kinematic information of the ego vehicle and the five closest vehicles (ordered from closest to the furthest) is used as the Markov state, i.e., $s_t = \{[x_t, y_t, \dot{x}_t, \dot{y}_t]\}_{\text{ego, other}_1, \text{ ..., other}_5} \in \mathbb{R}^{6\times4}$. The ego-car is illustrated in green and the other cars in blue.
2. **Action space, $\mathbb{A}$:** We use a discrete action space, constructed by K-means clustering of the continuous actions of the intelligent driving model (Kesting et al., 2010). We found out that keeping 9 actions was sufficient, i.e., $a_t \in \{0, \dots, 8\}$.
3. **Demonstrations, $\mathcal{D}$:** At each time-step, the ego-car observes online the state-action pairs for the 5 closest cars.

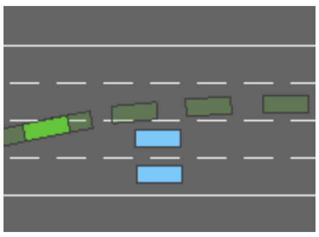


Figure B.1: `Highway`

B.1.2 `Roundabout`

We build on the `roundabot-v0` task from the `highway-env` traffic simulator (Leurent, 2018). The task is specified by:

1. **State space, $\mathbb{S}$:** The kinematic information of the ego vehicle and the five closest vehicles (ordered from closest to the furthest) is used as the Markov state, i.e., $s_t = \{[x_t, y_t, \dot{x}_t, \dot{y}_t]\}_{\text{ego, other}_1, \ldots, \text{other}_3} \in \mathbb{R}^{4\times4}$. The ego-car is illustrated in green and the other cars in blue.
2. **Action space, $\mathbb{A}$:** We use a discrete action space, constructed by K-means clustering of the continuous actions of the intelligent driving model (Kesting et al., 2010). We found out that keeping 6 actions was sufficient, i.e., $a_t \in \{0, \ldots, 5\}$.
3. **Demonstrations, $\mathcal{D}$:** At each time-step, the ego-car observes online the state-action pairs for the 3 closest cars.

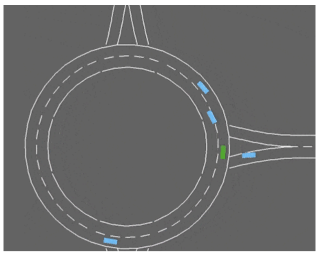


Figure B.2: `Roundabout`

B.1.3 `CoinGrid`

We build a simple multi-task gridworld. The task is specified by:

1. **State space, $\mathbb{S}$:** We use a symbolic, multi-channel representation of the 7×7 gridworld (Chevalier-Boisvert et al., 2019): the first three channels specify the presence or absence of the three different coloured boxes, the forth channel was the walls mask and the fifth and last channel was the position and orientation of the agent. We represent the orientation of the agent by 'painting' the cell in front of the agent. Therefore $s_t \in \{0, 1\}^{7\times7\times5}$.
2. **Action space, $\mathbb{A}$:** We use the $\{$`LEFT`, `RIGHT`, `FORWARD`$\}$ actions from Minigrid (Chevalier-Boisvert et al., 2018) to navigate the maze, i.e., $a_t \in \{0, 1, 2\}$.
3. **Demonstrations, $\mathcal{D}$:** At the beginning of training, the agent is given state-action pairs of other agents collecting either red or green coins.

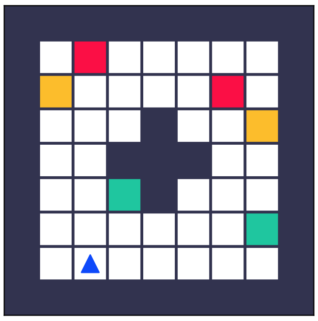


Figure B.3: `CoinGrid`

B.1.4 Fruitbot

We build on the `Fruitbot` environment from OpenAI's ProcGen benchmark (Cobbe et al., 2020). The task is specified by:

1. **State space, $\mathbb{S}$:** We use the original high-dimensional 64×64 RGB observations, i.e., $s_t \in [0, 1]^{64\times64\times3}$.
2. **Action space, $\mathbb{A}$:** We use the original 15 discrete actions, i.e., $a_t \in \{0, \dots, 14\}$.
3. **Demonstrations, $\mathcal{D}$:** At each time-step, the agent observes online the states and actions of 3 trained agents playing the game in parallel: One agent collects both fruits and other objects, one collects other objects and avoids fruits and the last one randomly selects actions.

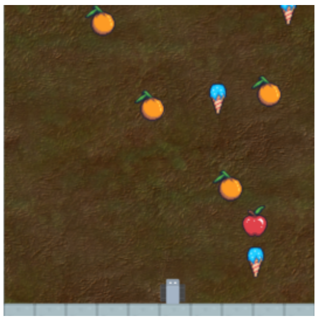


Figure B.4: `FruitBot`

B.2 Implementation details

For our experiments we used Python (Van Rossum and Drake Jr, 1995). We used JAX (Bradbury et al., 2018; Babuschkin et al., 2020) as the core computational library, Haiku (Babuschkin et al., 2020) and Acme (Hoffman et al., 2020) for implementing $\Psi\Phi$-learning ($\Psi\Phi$L) and the baselines, see §4.3. We also used Matplotlib (Hunter, 2007) for the visualisations and Weights & Biases (Biewald, 2020) for managing the experiments.

Compute resources. All the experiments were run on Microsoft Azure `Standard_NC6s_v3` machines, i.e., with a 6-core vCPU, 112GB RAM and a single NVIDIA Tesla V100 GPU. The iteration cycle for (i) `Highway` experiments was 3 hours; (ii) `CoinGrid` experiments was 5.5 hours and (iii) `FruitBot` experiments was 19 hours.

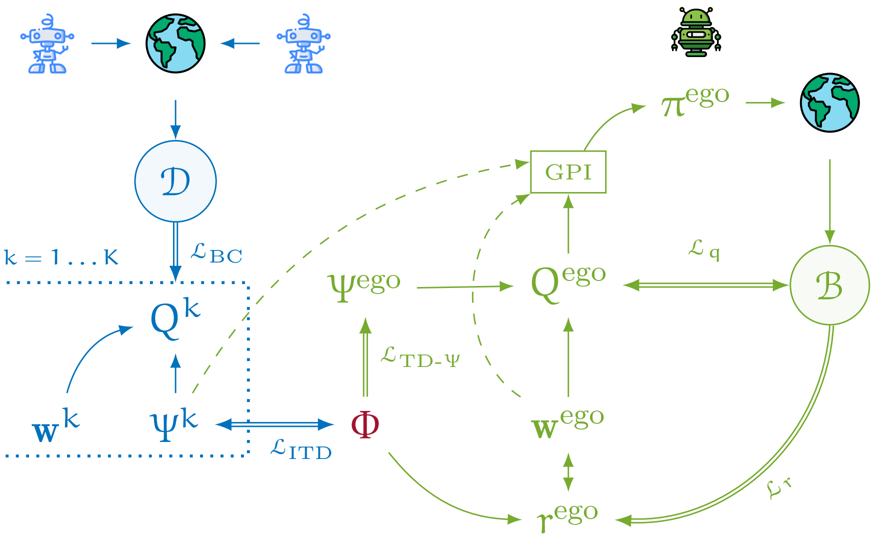


Figure B.5: Computational graph of the ΨΦ-learning algorithm. Demonstrations $\mathcal{D}$ contain data from other agents for *unknown* tasks. We employ *inverse temporal difference learning* (ITD, see §4.2.1) to recover other agents' successor features (SFs) and preferences. The ego-agent combines the estimated SFs of others along with its own preferences and successor features with generalised policy improvement (GPI, see §4.1), generating experience. Both the demonstrations and the ego-experience are used to learn the shared cumulants. Losses $\mathcal{L}_*$ are represented with double arrows and gradients flow according to the pointed direction(s).

Table B.1: ΨΦ-learner's hyperparameters per environment. The tuning was performed on a DQN (Mnih et al., 2013) baseline with population based training (Jaderberg et al., 2017) using Weights & Biases (Biewald, 2020) integration with Ray Tune (Liaw et al., 2018). We selected the best hyperparameters configuration out of 32 trials per environment and used this for our ΨΦ-learner.

	`Highway`	`CoinGrid`	`FruitBot`
Torso network, $\mathcal{E}$	MLP([512, 256])	IMPALA, shallow (no LSTM)	IMPALA, deep (no LSTM)
Cumulants approximator, Φ	MLP([128, 128])	MLP([256, 128])	MLP([256, 128])
Successor features approximator, Ψ	MLP([256, 128])	MLP([512, 256])	MLP([512, 256])
Ensemble size, Ψ	2	2	2
$\mathcal{L}_1$ coefficient	0.05	0.05	0.05
Number of dimensions in Φ	8	4	64
Minibatch size	512	64	32
n-step	4	8	128
Discount factor, γ	1.0	0.9	0.999
Target network update period	100	1000	2500
Optimiser	ADAM, `lr=1e-3`	ADAM, `lr=1e-4`	ADAM, `lr=5e-5`

B.3 Proofs

First, we formalise the statement of *Theorem 4.2*.

> ***Theorem 2.1*** (Validity of the ITD learning minimiser)***:*** The minimisers of $\mathcal{L}_{\text{BC}}$ and $\mathcal{L}_{\text{ITD}}$ are potentially-shaped cumulants that explain the observed reward-free demonstrations.

Proof. First, we prove the validity of the minimiser of the inverse temporal difference learning for the single-task setting. Next, we show that the result holds true in the vector (i.e., cumulants) case.

Single task. We assume that our demonstrations are generated by an expert, who samples actions from a Boltzmann policy, according to the optimal (for its task) action-value function $q^{\pi_{\text{expert}}}$ and temperature $\nu > 0$, i.e., $\mathcal{D} = \{(s, a)\} \sim \pi_{\text{expert}}$ s.t.

$$\pi_{\text{expert}}(a|s) \triangleq p(A = a|s) = \frac{\exp(\frac{1}{\nu} q^{\pi_{\text{expert}}}(s, a))}{\sum_a \exp(\frac{1}{\nu} q^{\pi_{\text{expert}}}(s, a))}, \ \forall s \in \mathbb{S}, a \in \mathbb{A}\,. \tag{B.1}$$

The minimiser of the behavioural cloning loss $\mathcal{L}_{\text{BC}}(\theta_q)$, i.e., *Eqn.* (4.17), for a single expert is such that

$$\theta_q^* \in \arg\min_{\theta_q} \mathbb{E}_{(s,a)\sim\mathcal{D}} \log \frac{\exp(q(s, a; \theta_q))}{\sum_a \exp(q(s, a; \theta_q))} \tag{B.2a}$$

$$\Rightarrow q(s, a; \theta_q^*) = \frac{1}{\nu} q^{\pi_{\text{expert}}}(s, a) + F(s), \ \forall s \in \mathbb{S}, a \in \mathbb{A}\,, \tag{B.2b}$$

where $F : \mathbb{S} \to \mathbb{R}$ is a state-dependent (bounded potential) function. We arrive at *Eqn.* (B.2b) by (i) testing $\frac{1}{\nu} q^{\pi_{\text{expert}}}(s, a)$ as a solution and noting that the "softmax" function is convex in the exponent (Boyd et al., 2004) and (ii) using the translation invariance property of the assumed Boltzmann policy parametrisation, i.e., for any $f : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ and $g : \mathbb{S} \to \mathbb{R}$

$$\frac{\exp(f(s, a) + g(s))}{\sum_a \exp(f(s, a) + g(s))} = \frac{\exp(g(s)) \exp(f(s, a))}{\exp(g(s)) \sum_a \exp(f(s, a))} = \frac{\exp(f(s, a))}{\sum_a \exp(f(s, a))}\,. \tag{B.3}$$

The minimiser of the inverse temporal difference learning loss $\mathcal{L}_{\text{ITD}}(\theta_q, \theta_r)$, *Eqn.* (4.18), for a single expert is such that

$$\theta_q^*, \theta_r^* \in \arg\min_{\theta_q, \theta_r} \mathbb{E}_{(s,a,s',a')\sim\mathcal{D}} \| q(s, a; \theta_q) - r(s, a; \theta_r) - \gamma q(s', a'; \theta_q) \| \tag{B.4}$$

where θ_q^* is minimising $\mathcal{L}_{\text{BC}}(\theta_q)$ simultaneously, as in *Eqn.* (B.2b). Therefore, it holds that $\mathcal{L}_{\text{ITD}}(\theta_q^*, \theta_r^*) = 0$

$$\begin{aligned}
r(s,a;\theta_r^*) &= q(s,a;\theta_q^*) - \gamma q(s',a';\theta_q^*) && \text{(B.5a)}\\
&\overset{(B.2b)}{=} \frac{1}{\nu} q^{\pi_{\text{expert}}}(s,a) + F(s) - \gamma \frac{1}{\nu} q^{\pi_{\text{expert}}}(s',a') - \gamma F(s') && \text{(B.5b)}\\
&= \frac{1}{\nu}\Bigg[\underbrace{q^{\pi_{\text{expert}}}(s,a) - \gamma q^{\pi_{\text{expert}}}(s',a')}_{r^{\text{expert}}(s,a)}\Bigg] + F(s) - \gamma F(s') && \text{(B.5c)}\\
&= \frac{1}{\nu} r^{\text{expert}}(s,a) + \underbrace{F(s) - \gamma F(s')}_{\substack{\text{potential-based reward}\\ \text{shaping function}}}, && \text{(B.5d)}
\end{aligned}$$

where r^{expert} is the (unobserved) expert's reward function. We have shown that the minimiser of $\mathcal{L}_{\text{BC}}$ and $\mathcal{L}_{\text{ITD}}$ leads to a reward function $r(s,a;\theta_r^*)$ which is a potential-based shaped and scaled reward function of the expert reward function and hence the optimal policy for $r(s,a;\theta_r^*)$ is also optimal for $r^{\text{expert}}(s,a)$ for all s, a (Ng et al., 1999).

Multiple tasks. The minimiser of the behavioural cloning loss $\mathcal{L}_{\text{BC}}(\theta_{\psi^k}, \mathbf{w}^k)$, i.e., *Eqn.* (4.17), for the k-th expert is s.t.

$$\begin{aligned}
\theta_{\psi^k}^*, \mathbf{w}^{k*} &\in \underset{\theta_{\psi^k}, \mathbf{w}^k}{\arg\min}\ \mathbb{E}_{(s,a)\sim\mathcal{D}^k} \log \frac{\exp(\Psi(s,a;\theta_{\psi^k})^\top \mathbf{w}^k)}{\sum_a \exp(\Psi(s,a;\theta_{\psi^k})^\top \mathbf{w}^k)} && \text{(B.6a)}\\
&\Rightarrow \Psi(s,a;\theta_{\psi^k})^\top \mathbf{w}^k = \frac{1}{\nu} q^{\pi_{\text{k-expert}}}(s,a) + F^k(s),\ \forall s \in \mathbb{S}, a \in \mathbb{A}, && \text{(B.6b)}
\end{aligned}$$

where $q^{\pi_{\text{k-expert}}}$ is the k-agent's action-value function and $H^k : \mathbb{S} \to \mathbb{R}$ a state-dependent (bounded potential) function. Next, the minimiser of the inverse temporal difference learning loss $\mathcal{L}_{\text{ITD}}(\theta_{\psi^k}, \theta_\Phi)$, *Eqn.* (4.18), for the k-th expert is s.t.

$$\theta_{\psi^k}^*, \theta_\Phi^* \in \underset{\theta_{\psi^k}, \theta_\Phi}{\arg\min}\ \mathbb{E}_{(s,a,s',a')\sim\mathcal{D}^k} \|\Psi(s,a;\theta_{\psi^k}) - \Phi(s,a;\theta_\Phi) - \gamma\Psi(s',a';\theta_{\psi^k})\| \quad \text{(B.7)}$$

where $\theta_{\psi^k}^*$ is minimising $\mathcal{L}_{\text{BC}}(\theta_{\psi^k}, \mathbf{w}^k)$ simultaneously, as in *Eqn.* (B.6b). Therefore, it holds that for $\mathcal{L}_{\text{ITD}}(\theta_{\psi^k}^*, \theta_\Phi^*) = 0$

$$\begin{aligned}
\Phi(s,a;\theta_\Phi^*) &= \Psi(s,a;\theta_{\psi^k}^*) - \gamma\Psi(s',a';\theta_{\psi^k}^*) && \text{(B.8a)}\\
\Phi(s,a;\theta_\Phi^*)^\top \mathbf{w}^{k*} &= \Psi(s,a;\theta_{\psi^k}^*)^\top \mathbf{w}^{k*} - \gamma\Psi(s',a';\theta_{\psi^k}^*)^\top \mathbf{w}^{k*} && \text{(B.8b)}\\
&\overset{(B.6b)}{=} \frac{1}{\nu} q^{\pi_{\text{k-expert}}}(s,a) + F^k(s) - \gamma\frac{1}{\nu} q^{\pi_{\text{k-expert}}}(s',a') - \gamma F^k(s') && \text{(B.8c)}\\
&= \frac{1}{\nu}\Bigg[\underbrace{q^{\pi_{\text{k-expert}}}(s,a) - \gamma q^{\pi_{\text{k-expert}}}(s',a')}_{r^{\text{k-expert}}(s,a)}\Bigg] + F^k(s) - \gamma F^k(s') && \text{(B.8d)}
\end{aligned}$$

$$= \frac{1}{\nu} r^{\text{k-expert}}(s, a) + \underbrace{F^k(s) - \gamma F^k(s')}_{\substack{\text{potential-based reward}\\ \text{shaping function}}} , \tag{B.8e}$$

We have shown that the minimiser of $\mathcal{L}_{\text{BC}}$ and $\mathcal{L}_{\text{ITD}}$ leads to agent-agnostic cumulants $\Phi(s, a; \theta^*_\Phi)$ and agent-specific preference vector $\mathbf{w}^{k*}$, which when dot-producted, form a potential-based shaped and scaled reward function of the k-th expert reward function and hence the optimal policy for $\Phi(s, a; \theta^*_\Phi)^\top \mathbf{w}^{k*}$ is also optimal for $r^{\text{k-expert}}(s, a)$ for all s, a (Ng et al., 1999). The result holds for all $k \in \{1 \dots K\}$ since no assumptions were made for the proof about k. □

Next, we prove the generalisation bound of the ΨΦ-learning (ΨΦL). When not specified the norm $\|\cdot\|$ refers to the 2-norm. Given a function $F : \mathcal{X} \to \mathbb{R}^d$ for some finite set $\mathcal{X}$, we will write $F(x)$ to denote the value of the function on input x and F to denote the matrix representation of this function in $\mathbb{R}^{|\mathcal{X}| \times d}$.

Theorem 2.2 (Generalisation bound of ΨΦ-learning (ΨΦL))***:*** Let $\mathcal{C} = (\mathbb{S}, \mathbb{A}, P, \gamma)$ be a CMP with a finite state space. Let $\Phi : \mathbb{S} \to \mathbb{R}^d$, and let $\Phi = \Phi(\mathbb{S}) \in \mathbb{R}^{|\mathbb{S}| \times d}$. Let $(r_i)_{i=1}^k$ denote a set of reward functions on $\mathcal{C}$, $\tilde{\Psi}^i$ be a collection of successor features approximations for policies $(\pi^i)_{i=1}^k$ (π_i optimal for r_i) with true successor feature values Ψ^i, and w_i the best least-squares linear approximator of r_i given Φ, with errors

$$\|\Phi w_i - r_i\|_\infty < \delta_r \text{ and } \|\tilde{\Psi}^i - \Psi^i\| < \delta_\Psi \quad \forall i.$$

Let w' be a new preference vector for a reward function r', with maximal error δ_r as well. Let $\tilde{q}^i = \tilde{\Psi}^i w'$. Let π^* be the optimal policy for the ego task w' and let π be the GPI policy obtained from $\{\tilde{q}^{\pi_i}\}$, with δ_r, δ_Ψ the reward and successor feature approximation errors. Then for all s, a

$$q^*(s, a) - q^\pi(s, a) \leqslant \frac{2}{1-\gamma} \left[(\Phi_{\max} \|w_j - w'\| + 2\delta_r) + \|w'\| \delta_\Psi + \frac{1}{(1-\gamma)} \delta_r \right] \tag{B.9}$$

Barreto et al. (2017) construct their bound on the sub-optimality of the GPI policy as a function of the error of the value approximations $\tilde{q}^i$. Because we bound the reward approximation error, rather than the value approximation error, we require an additional step to obtain a bound on the errors of the value funciton approximations. To prove Theorem 1, we must therefore first use the following lemma to bound the effect of the *reward approximation error* on the value approximation error. While this result is straightforward, we include a short proof for completeness.

Lemma 2.1: Fix some policy π. Let r be reward vector and let w be the least-squares solution to $\min \|\Phi w - r\|$. Let Ψ^π be the true successor features for Φ under policy π, and let q^π be the value. Let $\delta_r = r(\mathbb{S}) - \Phi w$, $\delta_{max} = \|\delta_r\|_\infty$. Then letting $\tilde{q} = \Psi w$, we have

$$\|q^\pi - \tilde{q}\|_\infty \leqslant \frac{1}{1-\gamma}\delta_r \tag{B.10}$$

Proof.

$$\|q^\pi - \tilde{q}\|_\infty \leqslant \sum \gamma^t \|P^{\pi t}(\Phi w - r)\|_\infty \tag{B.11a}$$

$$\leqslant \sum \gamma^t \|P^{\pi t}\delta_r\|_\infty = \sum_t \gamma^t \max_{s'} |\sum_{s \in \mathbb{S}} (P^\pi)^t(s', s)\delta_r(s)| \tag{B.11b}$$

Since P^π is a stochastic matrix, so are all of its powers, and so the rows of $(P^\pi)^t$ sum to 1.

$$\leqslant \sum_t \gamma^t \max_{s'} |\sum P^{\pi t}(s, s')\delta_{\max}| = \sum_t \gamma^t \delta_{\max} = \frac{1}{1-\gamma}\delta_{\max} \tag{B.11c}$$

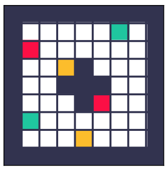
□

Proof. We follow the proof of Barreto et al. (2017, Theorem 2), with additional error terms to account for the reward and successor feature approximation errors.

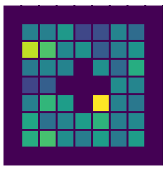
$$\begin{aligned}
&q^*(s, a) - q^\pi(s, a) \\
&\leqslant q^*(s, a) - q^{\pi_j}(s, a) + \frac{2}{1-\gamma}\epsilon && \text{(Barreto et al., 2017, Theorem 1)} \\
&\leqslant \frac{2}{1-\gamma}\|r_j - r'\|_\infty + \frac{2}{1-\gamma}\epsilon && \text{(Barreto et al., 2017, Lemma 1)} \\
&\leqslant \frac{2}{1-\gamma}\|\Phi w_j + \delta_j - \Phi w' - \delta'\|_\infty + \frac{2}{1-\gamma}\epsilon \\
&\leqslant \frac{2}{1-\gamma}(\Phi_{\max}\|w_j - w'\| + \delta_r + \delta_r) + \frac{2}{1-\gamma}\epsilon \\
&\leqslant \frac{2}{1-\gamma}(\Phi_{\max}\|w_j - w'\| + 2\delta_r) + \frac{2}{1-\gamma}\|\tilde{\Psi}^j w' - \Psi_j w' + \Psi_j w' - q_j\| \\
&\leqslant \frac{2}{1-\gamma}(\Phi_{\max}\|w_j - w'\| + 2\delta_r) + \frac{2}{1-\gamma}\|\tilde{\Psi}^j w' - \Psi_j w'\| + \|\Psi_j w' - q_j\| \\
&\leqslant \frac{2}{1-\gamma}(\Phi_{\max}\|w_j - w'\| + 2\delta_r) + \frac{2}{1-\gamma}\|w'\|\delta_\Psi + \frac{2}{1-\gamma}\|\Psi_j w' - q_j\| \\
&\leqslant \frac{2}{1-\gamma}(\Phi_{\max}\|w_j - w'\| + 2\delta_r) + \frac{2}{1-\gamma}\|w'\|\delta_\Psi + \frac{2}{1-\gamma}(\frac{1}{1-\gamma}\delta_r) && \text{(\textit{Lemma 2.1})} \\
&= \frac{2}{1-\gamma}\left[(\Phi_{\max}\|w_j - w'\| + 2\delta_r) + \|w'\|\delta_\Psi + \frac{1}{(1-\gamma)}\delta_r\right]
\end{aligned}$$

□

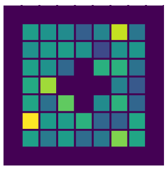
B.4 Visualisations



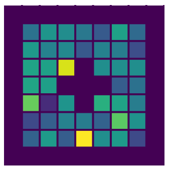
(a) CoinGrid



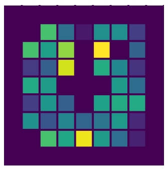
(b) $\hat{\Phi}_1$



(c) $\hat{\Phi}_2$

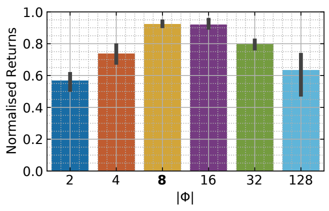


(d) $\hat{\Phi}_3$

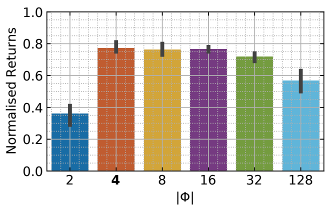


(e) $\hat{\Phi}_4$

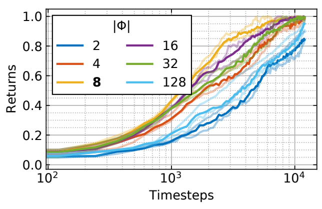
Figure B.6: Qualitative evaluation of the learned cumulants in the CoinGrid task. Cumulants $\hat{\Phi}_1$, $\hat{\Phi}_2$, and $\hat{\Phi}_3$ seem to capture the red, green, and yellow blocks, respectively. The yellow blocks are captured by both and $\hat{\Phi}_4$. Therefore, linear combinations of the learned cumulants can represent arbitrary rewards in the environment, which involve stepping on the coloured blocks.



(a) ITD for `Roundabout`



(b) ITD for `CoinGrid`



(c) ΨΦL for `MultiHighway`

Figure B.7: Sensitivity of our ITD (see §4.2.1) and ΨΦ-learning (see §4.2.2) algorithms to the dimensionality of the learned cumulants. We consistently observe across all three experiments (a)-(c) that for a small number of Φ dimensions the cumulants are not expressive enough to capture the axis of variation of the different agents' reward functions (including the ego-agent in (c)). We also note that the performance of both ITD and ΨΦ-learning is relative robust for a medium and large number of Φ dimensions. We attribute this to the used sparsity prior, i.e., $\mathcal{L}_1$ loss, to the preferences $\mathbf{w}$. In our experiments we selected the smallest number of Φ dimensions that demonstrated good performance to keep the number of model parameters as small as possible (in bold in the figures and reported in *Table B.1*).

C

Plan with model-value inconsistency

C.1 Experimental details

In this section, we describe the environments used in our experiments (see §5.3) and the experiment design.

C.1.1 Environments

In this section, we provide details on the specification of each task used in our experiments.

Tabular environment

We use an empty 5×5 gridworld (`GridWorld`) environment for our tabular experiments. The task is specified by:

1. **State space, $\mathbb{S}$:** A finite discrete state space, i.e., $s \in \{0, 1, \dots, 24\}$.
2. **Action space, $\mathbb{A}$:** A finite discrete action space for moving the agent in the four cardinal directions (`N`, `W`, `S`, `E`), i.e., $s \in \{0, 1, 2, 3\}$.
3. **Reward function, $r(s, a)$:** The zero function, i.e., $r(s, a) = 0, \forall (s, a) \in \mathbb{S} \times \mathbb{A}$.
4. **Transition dynamics, $p(s'|s, a)$:** We consider the

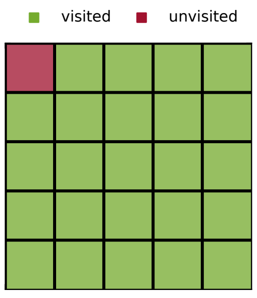


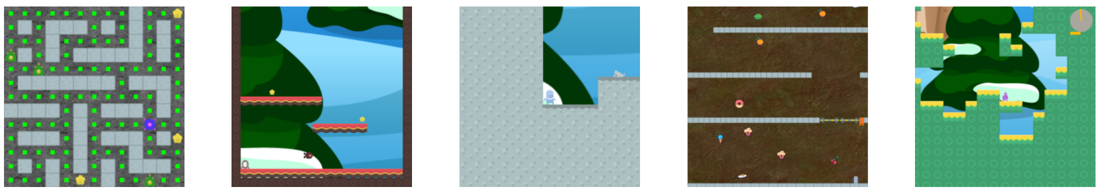
Figure C.1: `GridWorld`

episodic setting, i.e., `episode_length = 20`, and the dynamics are (optionally) stochastic. In particular, we use a single parameter that controls the stochasticity, called `wind_prob` $\in [0, 1]$ and implement stochastic dynamics as actuator noise, i.e., there is a `wind_prob` probability that the agent action is ignores and an other action is applied to the environment by sampling randomly from the action space.

Procgen (Cobbe et al., 2020)

We used 5 tasks from the Procgen (`Procgen`, Cobbe et al., 2020) suite, shown at *Figure C.2*. We used the default settings for the environments and we only varied the number of training levels used for learning, which we term `#levels`. The tasks are generally partially observed Markov decision processes (POMDPs) specified by:

1. **Observation space, $\mathcal{O}$:** The original 64×64 RGB pixel-observations, i.e., $o_t \in [0, 1]^{64\times64\times3}$.
2. **Action space, $\mathbb{A}$:** The original 15 discrete actions, i.e., $a_t \in \{0, \dots, 14\}$.



(a) `chaser` *(b)* `climber` *(c)* `coinrun` *(d)* `fruitbot` *(e)* `jumper`

Figure C.2: `Procgen` tasks.

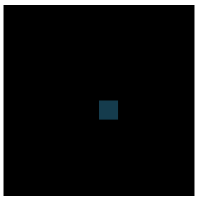
MinAtar (Young and Tian, 2019)

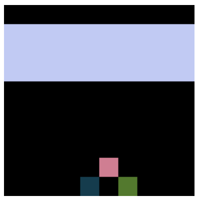
We used all 5 tasks from the MinAtar (`MinAtar`, Young and Tian, 2019) suite, shown in *Figure C.3*, with the default settings. The tasks are fully-observed and specified by:

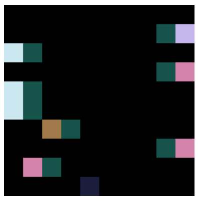
1. **State space, $\mathbb{S}$:** The original $10 \times 10 \times \texttt{n\_channels}$ symbolic observations, i.e., $s_t \in [0, 1]^{10\times10\times\texttt{n\_channels}}$, where `n_channels` varies between tasks, from 4 to 10.
2. **Action space, $\mathbb{A}$:** The original 6 discrete (non-minimal) actions, i.e., $a_t \in \{0, \dots, 5\}$.
3. **Transition dynamics, $p(s'|s, a)$:** The default 0.1 probability for sticky actions is used.

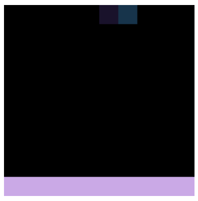
DeepMind continuous control (Tunyasuvunakool et al., 2020)

We use the `walker` walk task from the DeepMind Continuous Control (Tunyasuvunakool et al., 2020) suite and modified its reward function. Pixel-observations are


(a) asterix


(b) breakout


(c) freeway


(d) seaquest

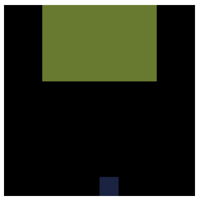

(e) space_invaders

Figure C.3: `MinAtar` tasks.

used, and the problem is generally partially-observed. The task is specified by:

1. **Observation space,** $\mathcal{O}$**:** A 64×64 RGB pixel-observation, where the robot body is in the centre of the frame, i.e., $o_t \in [0,1]^{64\times64\times3}$.
2. **Action space,** $\mathbb{A}$**:** A six-dimensional continuous action, i.e., $a_t \in [-1,+1]^6$.
3. **Reward function,** $r(s,a)$**:** Originally, the reward is bounded in $[0,1]$, i.e., $r_t \in [0,1]$, which is computed based on the robot's torso height and forward velocity. We modify the original per-step reward, by setting to zero any reward below a parameter η, i.e., $\tilde{r}_t = \mathcal{H}(r_t - \eta)r_t$, where $\mathcal{H}$ is the Heaviside step function. For $\eta = 0$, we recover the original reward, and for $\eta > 0$ we obtain an increasingly more difficult, in terms of exploration, `walker` task.

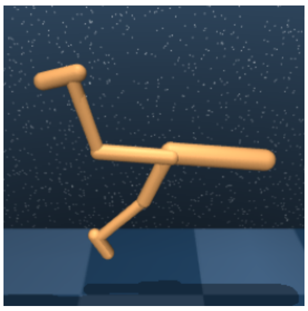


Figure C.4: `walker`

C.1.2 Experiments

In this section, we provide details on the experimental protocol of each experiment.

Figure 5.6

Data. We collect experience/data $\mathcal{B}$ by running a uniformly random policy π_{uniform} for 500 steps (i.e., 25 episodes). We exclude transitions from and to the top left cell, which we call the *out-of-training distribution* (OOD) or unvisited state.

Value learning. We learn a tabular action-value function $\hat{q} \approx q^{\pi_{\text{uniform}}}$ using expected SARSA (Van Seijen et al., 2009) and then we induce a (state-)value function, i.e.,

$$\hat{v}(s) \triangleq \mathbb{E}_{a \sim \pi_{\text{uniform}}}[\hat{q}(s, a)], \forall (s, a) \in \mathbb{S} \times \mathbb{A}. \tag{C.1}$$

Model learning. Reconstruction-based model learning (see §2.3.2), with data $\mathcal{B}$ is used for learning the tabular model of the environment, such that $\hat{m} \approx m^*$.

Visualisations. The mean and standard deviations are normalised in $[0, 1]$, i.e., for given quantity x_s for state s and $x_{\min}$ ($x_{\max}$ the minimum (maximum) quantity across all states, we plot $\bar{x}_s = (x_s - x_{\min})/(x_{\max} - x_{\min})$. We report the results for a single repetition of the experiment since it is a qualitative observation.

IVE[N]. We calculate the MPVs exactly, according to *Eqn.* (5.1). We vary the parameter N, i.e., maximum number of applications of the model-induced Bellman operator $\mathscr{T}_m$ on the learned value function $\hat{v}$.

EVE[N] and EMVE[N]. The explicit value ensemble (EVE, *Figure 5.2a*) and explicit model value ensemble (EMVE, *Figure 5.2b*) are also trained on the same data using the same value and model learning algorithms, according to the *Remark* 2.7 strategy.

Table 5.1

We use the `walker` task and train Dreamer (Hafner et al., 2019a) for 1M steps. An action repeat of 2 is used thus 0.5M agent-environment interaction steps are made per run. We repeat each experiment 3 times, varying the random seed in each one. We report the episodic returns (rounded to the nearest tenth) at the end of training by setting the agents in "evaluation" mode and average their performance across 10 episodes.

Figure 5.7

We train Muesli (without any modification to its acting strategy or learning algorithm) for 100M environment frames. *Figure 5.7a* reports the final performance of the agent evaluated on an additional 10M frames on the train and test levels. Mean episode returns are normalised as: $\tilde{R} = (R - R_{\min})/(R_{\max} - R_{\min})$, using min and max scores for each game (Cobbe et al., 2020).

The model-value self-inconsistency, reported in *Figure 5.7b*, is computed by unrolling the model for 5 steps using actions sampled from the policy and taking the standard deviation over the IVE:

$$\text{k-MVP}(s) = \hat{v}^k_\theta(s) \overset{(5.2)}{=} \sum_{i=1}^{k-1} \gamma^{i-1} r^{i+1} + \gamma^k \hat{v}(s^k) \tag{C.2a}$$

$$\sigma\text{-IVE}[5](s) = \sqrt{\text{Var}_k[\{\text{k-MVP}(s)\}_{k=1}^5]}, \tag{C.2b}$$

using the notation from §5.1.

Figure 5.8

For training the values and model and calculating IVE and EVE, we follow the same protocol as in *Figure 5.6*. In this experiment, we use the learned *action*-value functions instead of the state-values, see Section C.3.2 for a formal discussion. We denote with σ-IVE[5] and σ-EVE[5] the standard deviation across the 5 ensemble members of the implicit and explicit ensembles of the action-values, respectively. Also, $\sigma\text{-IVE}[5] \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ and $\sigma\text{-IVE}[5](s, a)$ is the standard deviation of the implicit value ensemble at the state s for action a. We use the standard deviation across the ensemble of action-values for inducing policies that are novelty- seeking or avoiding:

- In *Figure 5.8a*, the action that maximises the standard deviation across the value ensemble is selected, per-state, i.e., $\pi_{\text{seeking}}(s) = \arg\max_{a\in\mathcal{A}} \sigma\text{-XVE}[5](s, a)$, where XVE ∈ {IVE, EVE}. These are the novelty-seeking policies that their probability of reaching the novel state is higher than a uniformly random policy.
- In *Figure 5.8b*, the action that minimises the standard deviation across the value ensemble is selected, per-state, i.e., $\pi_{\text{avoiding}}(s) = \arg\min_{a\in\mathcal{A}} \sigma\text{-XVE}[5](s, a)$, where XVE ∈ {IVE, EVE}. These are the novelty-avoiding policies that their probability of reaching the novel state is lower than a uniformly random policy.

We calculate the probabilities by constructing a Markov chain, induced by the coupling of the policy under consideration π and the "true" environment model, m^*. The Markov chain's transition kernel is given by $p^{\pi}_{m^*}(s'|s) \triangleq \sum_{a\in\mathcal{A}} p_{m^*}(s'|s,a)\pi(a|s)$. We can write the transition kernel as a matrix $P^{\pi}_{m^*} \in \mathbb{R}^{\mathcal{S}\times\mathcal{S}}$, such that $P^{\pi}_{m^*}[i,j] = p^{\pi}_{m^*}(j|i)$. The (i,j) entry of the transition matrix, i.e., $P^{\pi}_{m^*}[i,j]$ is the probability of reaching the state j after one-step when starting from state i and following policy π in the environment with model m^*. The (i,j) entry of the l-th power of the transition matrix, i.e., $(P^{\pi}_{m^*})^{l}[i,j]$ is the probability of reaching the state j after l-step when starting from state i and following policy π in the environment with model m^*.

In *Figure 5.8*, we start from the bottom right cell, i.e., $i =$ `bottom right` and plot the probability of reaching the top left cell, i.e., $j =$ `top right` after l-step, and we vary l from 1 to 150. We repeat each experiment 100 times, varying the random seed in each one.

Table 5.2

We use the `MinAtar` tasks and train VPN (Oh et al., 2017) and some variants of it for 2M steps. The only modification to the original VPN(5) is the way value estimates are constructed:

- $\hat{v}^1_\theta$ is VPN variant that uses the 1-MPV for value estimation.
- $\hat{v}^5_\theta$ is VPN variant that uses the 5-MPV for value estimation.
- μ-IVE[5] is the original VPN(5) agent that uses the mean over the implicit value ensemble with $N = 5$ for value estimation.

The estimated values are used for value-based planning, as discussed in (§C.1, Oh et al., 2017).

C.2 Implementation details

For our experiments we used Python (Van Rossum and Drake Jr, 1995). We used JAX (Bradbury et al., 2018; Babuschkin et al., 2020) as the core computational library for implementing Muesli (Hessel et al., 2021) and VPN (Oh et al., 2017). We used the official TensorFlow (Abadi et al., 2015) implementation of Dreamer (Hafner et al., 2019a). We also used Matplotlib (Hunter, 2007) for the visualisations.

C.2.1 Tabular methods

We initialise the rewards, transition logits and action-values by sampling from a normal distribution with mean 0 and standard deviation 1. The ADAM (Kingma and Ba, 2014) optimiser with learning rate $5e\text{-}5$ is used, and all losses converge after $10,000$ epochs of stochastic gradient descent with batch size 128.

C.2.2 Dreamer (Hafner et al., 2019a)

We use the Dreamer agent's default hyperparameters, as introduced by (Hafner et al., 2019a). For the self-inconsistency-seeking variant, i.e., $\mu + \sigma\text{-IVE}[5]$, we used a scalar weighting factor $\beta^*_{\text{IVE}} = 0.1$ to balance the mean and standard deviation across the ensemble members, tuned with grid search in $\{0.05, 0.1, 0.2, 1.0, 10.0\}$. The same tuning procedure is used for the baselines. The reported scores are for $\beta^*_{\text{EVE}} = 0.2$ and $\beta^*_{\text{EMVE}} = 0.1$.

C.2.3 Muesli (Hessel et al., 2021)

We use the Muesli agent's hyperparameters. In particular we use the ones from the large-scale Atari experiments by Hessel et al. (2021). Nonetheless, we set the fraction of replay data in each batch to 0.8 (instead of the original 0.95) to shorten training time. To encourage diversity in value and reward predictions for unvisited states we have augmented the value and reward prediction heads of the model with untrainable *randomized prior networks* (Osband et al., 2018), using a prior scale of 5.0. Note that unlike in Osband et al. (2018), we did not introduce additional heads per prediction or modify the training procedure.

C.2.4 VPN (Oh et al., 2017)

We use the MinAtar DQN-torso (Young and Tian, 2019) and an LSTM (Hochreiter and Schmidhuber, 1997) with 128 hidden units and otherwise follow the original VPN(5) hyperparameters, as introduced by Oh et al. (2017).

C.3 Extensions

C.3.1 IVE with the Bellman optimality operator

In §5.2, we defined the k-step model-predicted value (k-MPV) in terms of the model-induced Bellman evaluation operator and an approximate value function $v_{\theta_v} \approx v^*$, to construct the implicit value ensemble (IVE) accordingly. Similarly, we can define the k-MPV and hence IVE in terms of the model-induced Bellman optimality operator, as defined in *Eqn.* (2.64), and an approximation to the optimal value function $v_{\theta_v} \approx v^*$, i.e.,

$$v_\theta^k(s) \equiv (\mathscr{T}^*_{m_{\theta_m}})^k v_{\theta_v}(s) \triangleq \operatorname*{arg\,max}_{a^{0:k-1}} \mathbb{E}_{m_{\theta_m}}[\sum_{i=0}^{k-1}(\gamma^i R_{i+1}) + \gamma^k v_{\theta_v}(S_k)|S_0 = s]. \tag{C.3}$$

The IVE with the Bellman optimality operator can be used for values learned with, e.g., Q-learning (§2.3.2; Watkins and Dayan, 1992), or with other value-based agents, such as the VPN (Oh et al., 2017). We use this formulation in §C.4.

C.3.2 MPV with action-value functions

In order to be able to modulate action selection using the self-inconsistency signal, we have computed the k-MPV conditioned on both the starting state and action, i.e.,

$$q_\theta^k(s,a) \triangleq \mathbb{E}_{\pi_{\theta_\pi}, m_{\theta_m}}[\sum_{i=0}^{k-1}(\gamma^i R_{i+1}) + \gamma^k v_{\theta_v}(S_k)|S_0 = s, A_0 = a] \tag{C.4a}$$

$$\hat{q}_\theta^k(s,a) \triangleq \sum_{i=0}^{k-1}(\gamma^i r^{i+1}) + \gamma^k v_{\theta_v}(s^k) \tag{C.4b}$$

where now reward and value predictions are computed after unrolling the model using action a for one step, and actions sampled from the policy for the remaining $k-1$ steps.

C.4 Additional experiments

C.4.1 Measuring self-inconsistency in OOD states

To complement our results in *Figure 5.7*, we have also evaluated self-inconsistency by computing the IVE as an average over 100 action sequences sampled from the policy, see *Figure C.5*. We observed only minor quantitative differences compared to the results presented in *Figure 5.7* (where we were using a single action sequence to estimate the IVE).

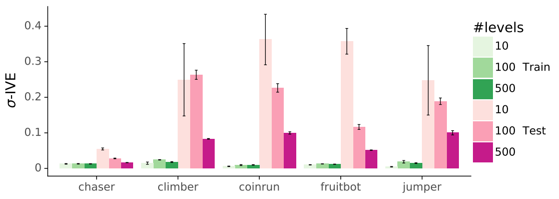
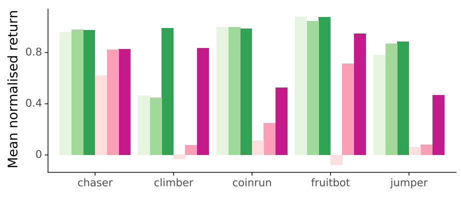


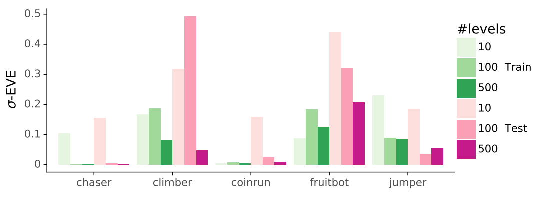
Figure C.5: σ-IVE[5] computed using the model of the Muesli agent while evaluating on both training and unseen test levels, for different numbers of unique levels seen during training. To estimate the IVE, we used 100 action sequences from the policy. Bars, error-bars show mean and standard error across 3 seeds, respectively.

C.4.2 Measuring explicit value ensemble variance in OOD states

To complement our results in *Figure 5.7* for IVE, we provide the EVE results in *Figure C.6*. We observe that EVE behaves similar to IVE in terms of ensemble variance as a function of `#levels` for both training and testing levels.



(a) Episodic returns



(b) Self-inconsistency

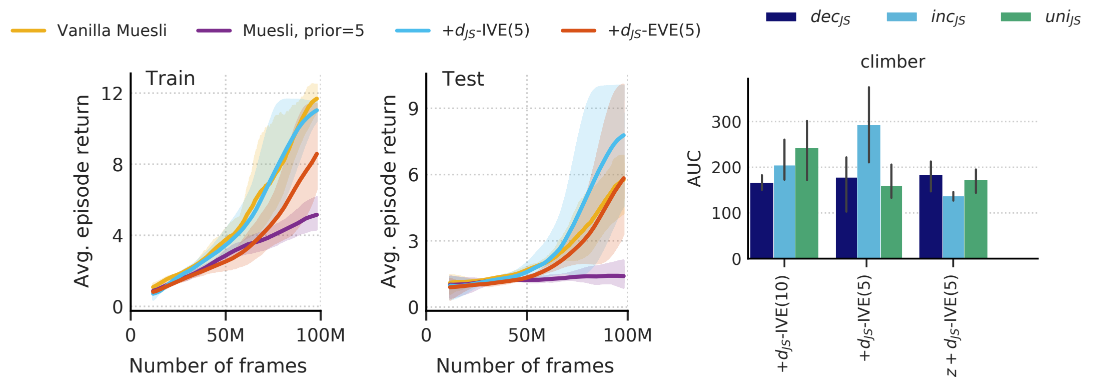
Figure C.6: σ-EVE[5] computed using the Muesli agent augmented with an ensemble of 5 value heads (different random initialisation) while evaluating on both training and unseen test levels, for different numbers of unique levels seen during training. (a) Performance for training (green) and test (pink) for varying number of levels. (b) Explicit value ensemble inconsistency measured by standard deviation of the 5 different heads.

C.4.3 How to use the IVE[5] signal?

In the following experiments we consider the self-inconsistency signal as an optimistic bonus to encourage better exploration during training, hence generalising better during evaluation. We test variants of the $+\sigma$-IVE[5] signal by mixing the policy with the self-inconsistency in probability space $+\sigma\text{-IVE}[5] \triangleq (1-\beta)\pi + \beta \cdot \sigma\text{-IVE}[5]$, and by mixing the signal with the policy logits: $z + \sigma\text{-IVE}[5] \triangleq \text{softmax}(z_\pi + \beta \cdot \sigma\text{-IVE}[5])$. We vary the number of MPV in the ensemble for $n = 5, 10$. Use further test using a different metric for measuring the disagreement across the nMPVs that considers different weighting averages over k:

$$d_{JS} = \text{JSD}_w(\text{IVE}(n)) = H\left(\sum_k w_k \hat{v}^k_{\hat{m}}\right) - \sum_k w_k H\left(\hat{v}^k_\theta\right) \tag{C.5}$$

with three weighting schemes: a decreasing weight $\text{dec}_{JS} : w_k = r^k / (\sum_j r^j)$ such that the weight decreases to $1/3$ over n, an increasing weight inc_{JS} with the inverse trend, and a uniform weight uni_{JS} that corresponds to the uniform mixing over n $w_k = 1/n$.



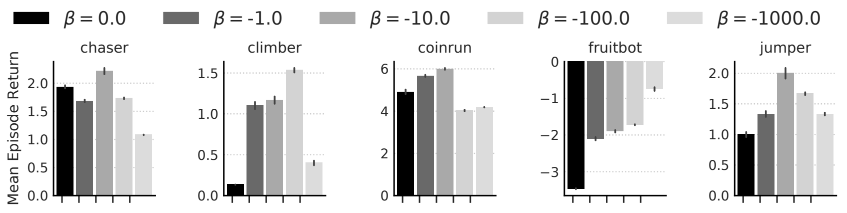
(a) Train *(b)* Test *(c)* Variations

Figure C.7: Model-value inconsistency (see §5.2.2) as the Jensen-Shannon divergence of the implicit value ensemble (see §5.2.1) for different numbers of ensemble components n, trained across 100 `Procgen` levels error bars show SE over 3 seeds. (a) Mean episode return during training with 100 `Procgen` levels, for Muesli baselines and for an agent trained with optimistic divergence over an explicit ensemble d_{JS}-EVE[5] and over IVE[5], both with an increasing Jensen-Shannon disagreement. (b) Mean episode return for evaluation without the optimistic disagreement for the same methods. (c) Ablation study over d_{JS}-IVE of varying length $n = 5, 10$ and by mixing in logit space $z + d$-IVE vs. mixing in probability space $+d$-IVE.

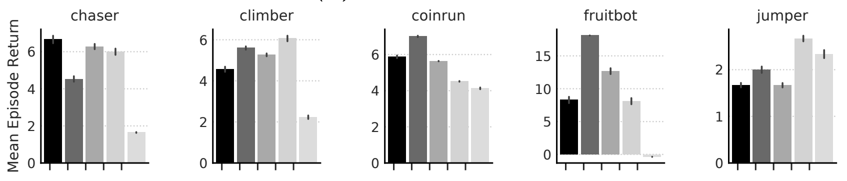
In *Figure C.7b* we observe that learning with an optimistic bonus helps with generalisation at evaluation time. *Figure C.7c* we observe that mixing over probability space is less sensitive to re-scaling β, but yields higher variance. We notice a trade-off between the weighting scheme used vs. the size of the IVE, for higher ns the best performing metric has less weight on the larger k-MPVs. For the decreasing metric the results remain more robust, suggesting that the inconsistencies are higher for larger ks. We used $\beta = 0.1$ for mixing in probability and $\beta = 1$ for the logit case.

C.4.4 Ablation on pessimism for evaluation

We evaluate in *Figure C.8* how sensitive the self-inconsistency signal is to different rescaling parameters β when acting pessimistically at test time $z - \beta$ d-σ-IVE[5] with an increasing weight. We trained a vanilla Muesli agent using 10/100 levels over 150M frames and evaluated with a pessimistic bonus for the consecutive 20M frames over 3 seeds



(a) `#levels` $= 10$



(b) `#levels` $= 100$

Figure C.8: Mean episode return evaluated with pessimism bonus $-d_{JS}$-IVE with increasing weights for each procgen environment on a trained vanilla Muesli using (a) 10 levels and (b) 100 levels. Error bars show 95% CI.

C.4.5 Dreamer variants

In §5.3.2, we modified the Dreamer (Hafner et al., 2019a) agent to improve its exploration without having to learn an explicit ensemble of value functions. We modify the behavioural policy used for collecting data, using the mean and standard deviation of the implicit value ensemble, i.e., μ-IVE[5] and σ-IVE[5], respectively. We use the original Dreamer setup otherwise.

In particular, for each time-step t, we sample action a_π^t from the learned policy π and then calculate the IVE(5), similar to *Eqn.* (5.2). Then, we can form the mean-variance knowledge equivalent (KE, Markowitz, 1952), given by:

$$\mathcal{U}(s) = \mu\text{-IVE[5]}(s) + \beta \cdot \sigma\text{-IVE[5]}(s). \tag{C.6}$$

We use online gradient-based or sample-based planning, a.k.a. model predictive control (MPC, Garcia et al., 1989) for selecting an action (see §2.3.3).

We used $\beta = 0.1$, 10 gradient steps or 10 samples from the learned policy for guiding the search in all of our experiments, shown in *Table C.1*.

Table C.1: Results for the Dreamer (Hafner et al., 2019a) agent and IVE variants on a modified version of the `walker` task with varying degrees of reward sparsity controlled by η, where higher η corresponds to harder exploration. A "$\diamondsuit$" indicates methods that use gradient-based trajectory optimisation, while "$\clubsuit$" indicates methods that use sample-based trajectory optimisation. We report mean and standard error of episodic returns (rounded to the nearest tenth) over 3 runs after 1M steps. Higher-is-better and the performance is upper bounded by 1000. The **best performing** method, per-task, is in bold.

Methods	$\eta = 0.0$	$\eta = 0.2$	$\eta = 0.3$	$\eta = 0.5$
Dreamer	**1000**±00	720±10	570±60	80±50
Dreamer$^\diamondsuit$	**1000**±00	540±30	240±50	40±30
μ-IVE[5]$^\diamondsuit$	**1000**±00	860±40	690±70	210±60
$\mu+\sigma$-EVE[5]$^\diamondsuit$	**1000**±00	**1000**±00	**980**±10	**280**±50
$\mu+\sigma$-EMVE[5]$^\diamondsuit$	**1000**±00	910±20	730±40	210±60
$\mu+\sigma$-IVE[5]$^\diamondsuit$	**1000**±00	**1000**±00	**1000**±00	**330**±70
$\mu+\sigma$-IVE[5]$^\clubsuit$	**1000**±00	**1000**±00	**1000**±00	**280**±40

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015.

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018.

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

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

Brandon Amos, Laurent Dinh, Serkan Cabi, Thomas Rothörl, Sergio Gómez Colmenarejo, Alistair Muldal, Tom Erez, Yuval Tassa, Nando de Freitas, and Misha Denil. Learning awareness models. *arXiv preprint arXiv:1804.06318*, 2018.

Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *International conference on machine learning*, pages 176–185. PMLR, 2017.

Javier Antorán, James Urquhart Allingham, and José Miguel Hernández-Lobato. Depth uncertainty in neural networks. *arXiv preprint arXiv:2006.08437*, 2020.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

Kenneth J. Arrow. The Theory of Risk-Bearing: Small and Great Risks. *Journal of Risk and Uncertainty*, 1965.

Kavosh Asadi, Dipendra Misra, Seungchan Kim, and Michel L Littman. Combating the compounding-error problem with a multi-step model. *arXiv preprint arXiv:1905.13320*, 2019.

Arsenii Ashukha, Alexander Lyzhov, Dmitry Molchanov, and Dmitry Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. *arXiv preprint arXiv:2002.06470*, 2020.

Christopher G Atkeson and Stefan Schaal. Robot learning from demonstration. In *ICML*, volume 97, pages 12–20. Citeseer, 1997.

Yusuf Aytar, Tobias Pfaff, David Budden, Tom Le Paine, Ziyu Wang, and Nando de Freitas. Playing hard exploration games by watching youtube. *arXiv preprint arXiv:1805.11592*, 2018.

Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Claudio Fantacci, Jonathan Godwin, Chris Jones, Tom Hennigan, Matteo Hessel, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Lena Martens, Vladimir Mikulik, Tamara Norman, John Quan, George Papamakarios, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Wojciech Stokowiec, and Fabio Viola. The DeepMind JAX Ecosystem, 2020.

Philip Ball, Jack Parker-Holder, Aldo Pacchiano, Krzysztof Choromanski, and Stephen Roberts. Ready policy one: World building through active learning. In *International Conference on Machine Learning*, pages 591–601. PMLR, 2020.

David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, pages 4055–4065, 2017.

André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Shibl Mourad, David Silver, Doina Precup, et al. The option keyboard: Combining skills in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 13052–13062, 2019.

André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117(48):30079–30087, 2020.

Eric Baum and David Haussler. What size net gives valid generalization? *Advances in neural information processing systems*, 1, 1988.

Feryal Behbahani, Kyriacos Shiarlis, Xi Chen, Vitaly Kurin, Sudhanshu Kasewa, Ciprian Stirbu, Joao Gomes, Supratik Paul, Frans A Oliehoek, Joao Messias, et al. Learning from demonstration in the wild. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 775–781. IEEE, 2019.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29:1471–1479, 2016.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Marc G Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, Sameera S Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020.

Richard Bellman. *Dynamic programming*. Princeton University Press, 1957a.

Richard Bellman. A Markovian decision process. *Journal of mathematics and mechanics*, 6(5):679–684, 1957b.

Daniel Bernoulli. Exposition of a new theory on the measurement of risk. 1738.

Dimitri Bertsekas. *Dynamic programming and optimal control*, volume 4. Athena scientific, 2012.

Lukas Biewald. Experiment Tracking with Weights and Biases, 2020. Software available from wandb.com.

Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Survey: Robot programming by demonstration. *Handbook of robotics*, 59(BOOK_CHAP), 2008.

Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015.

Ondřej Bojar and Aleš Tamchyna. Improving translation model by monolingual data. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 330–336, 2011.

Diana Borsa, Bilal Piot, Rémi Munos, and Olivier Pietquin. Observational learning by reinforcement learning. *arXiv preprint arXiv:1706.06617*, 2017.

Diana Borsa, André Barreto, John Quan, Daniel Mankowitz, Rémi Munos, Hado van Hasselt, David Silver, and Tom Schaul. Universal successor features approximators. *arXiv preprint arXiv:1812.07626*, 2018.

Olivier Bousquet and André Elisseeff. Stability and generalization. *The Journal of Machine Learning Research*, 2:499–526, 2002.

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

Daniel S Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. *arXiv preprint arXiv:1904.06387*, 2019.

Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, 365(6456):885–890, 2019.

Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *arXiv preprint arXiv:1807.01675*, 2018.

Lars Buesing, Theophane Weber, Sébastien Racaniere, SM Eslami, Danilo Rezende, David P Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, et al. Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*, 2018.

Arunkumar Byravan, Jost Tobias Springenberg, Abbas Abdolmaleki, Roland Hafner, Michael Neunert, Thomas Lampe, Noah Siegel, Nicolas Heess, and Martin Riedmiller. Imagined value gradients: Model-based policy optimization with tranferable latent dynamics models. In *Conference on Robot Learning*, pages 566–589. PMLR, 2020.

Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. *arXiv preprint arXiv:1910.05449*, 2019.

Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018.

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. *arXiv preprint arXiv:1912.12294*, 2019.

Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691. PMLR, 2014.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic Gridworld Environment for OpenAI Gym, 2018.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First Steps Towards Grounded Language Learning With a Human In the Loop. In *International Conference on Learning Representations*, 2019.

Sungjoon Choi, Kyungjae Lee, and Songhwai Oh. Robust learning from demonstrations with mixed qualities using leveraged gaussian processes. *IEEE Transactions on Robotics*, 35(3):564–576, 2019.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204. PMLR, 2015.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Adam Coates, Pieter Abbeel, and Andrew Y Ng. Learning for control from multiple demonstrations. In *Proceedings of the 25th international conference on Machine learning*, pages 144–151, 2008.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *International Conference on Computer Vision (ICCV)*, pages 9329–9338, 2019.

G Coley, A Wesley, N Reed, and I Parry. Driver reaction times to familiar, but unexpected events. *TRL Published Project Report*, 2009.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

Constantin Cronrath, Emilio Jorge, John Moberg, Mats Jirstrand, and Bengt Lennartson. BAgger: A Bayesian Algorithm for Safe and Query-efficient Imitation Learning, 2018.

Henggang Cui, Vladan Radosavljevic, Fang-Chieh Chou, Tsung-Han Lin, Thi Nguyen, Tzu-Kuo Huang, Jeff Schneider, and Nemanja Djuric. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2090–2096. IEEE, 2019.

Aaron Davidson. Using artificial neural networks to model opponents in Texas hold'em. *Unpublished manuscript*, 1999.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

Pim de Haan, Dinesh Jayaraman, and Sergey Levine. Causal confusion in imitation learning. In *Neural Information Processing Systems (NeurIPS)*, pages 11693–11704, 2019.

Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian Q-learning. In *Aaai/iaai*, pages 761–768, 1998.

Richard Dearden, Nir Friedman, and David Andre. Model-based Bayesian exploration. *arXiv preprint arXiv:1301.6690*, 1999.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

Marc Peter Deisenroth, Peter Englert, Jan Peters, and Dieter Fox. Multi-task policy search for robotics. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3876–3881. IEEE, 2014.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *Advances in neural information processing systems*, 30, 2017.

Michael Dusenberry, Ghassen Jerfel, Yeming Wen, Yian Ma, Jasper Snoek, Katherine Heller, Balaji Lakshminarayanan, and Dustin Tran. Efficient and scalable bayesian neural nets with rank-1 factors. In *International conference on machine learning*, pages 2782–2792. PMLR, 2020.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. *arXiv preprint arXiv:1808.09381*, 2018.

Daniel Ellsberg. Risk, ambiguity, and the Savage axioms. *The quarterly journal of economics*, 75(4):643–669, 1961.

Paul Embrechts, Claudia Klüppelberg, and Thomas Mikosch. *Modelling extremal events: for insurance and finance*, volume 33. Springer Science & Business Media, 2013.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

Amir-massoud Farahmand, Andre Barreto, and Daniel Nikovski. Value-aware loss function for model-based reinforcement learning. In *Artificial Intelligence and Statistics*, pages 1486–1494. PMLR, 2017.

Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning. *arXiv preprint arXiv:1710.11417*, 2017.

Gregory Farquhar, Kate Baumli, Zita Marinho, Angelos Filos, Matteo Hessel, Hado P van Hasselt, and David Silver. Self-Consistent Models and Values. In *Advances in Neural Information Processing Systems*, volume 34, pages 1111–1125, 2021.

Sebastian Farquhar, Michael A Osborne, and Yarin Gal. Radial bayesian neural networks: Beyond discrete support in large-scale bayesian deep learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1352–1362. PMLR, 2020.

Stefan Faußer and Friedhelm Schwenker. Neural network ensembles in reinforcement learning. *Neural Processing Letters*, 41(1):55–69, 2015.

Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value expansion for efficient model-free reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

Angelos Filos, Panagiotis Tigkas, Rowan McAllister, Nicholas Rhinehart, Sergey Levine, and Yarin Gal. Can Autonomous Vehicles Identify, Recover From, and Adapt to Distribution Shifts? In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3145–3153. PMLR, 2020.

Angelos Filos, Clare Lyle, Yarin Gal, Sergey Levine, Natasha Jaques, and Gregory Farquhar. PsiPhi-Learning: Reinforcement Learning with Demonstrations using Successor Features and Inverse Temporal Difference Learning. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3305–3317. PMLR, 2021.

Angelos Filos, Eszter Vértes, Zita Marinho, Gregory Farquhar, Diana Borsa, Abram Friesen, Feryal Behbahani, Tom Schaul, Andre Barreto, and Simon Osindero. Model-Value Inconsistency as a Signal for Epistemic Uncertainty. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 6474–6498. PMLR, 2022.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58, 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, pages 1126–1135, 2017.

Sebastian Flennerhag, Jane X Wang, Pablo Sprechmann, Francesco Visin, Alexandre Galashov, Steven Kapturowski, Diana L Borsa, Nicolas Heess, Andre Barreto, and Razvan Pascanu. Temporal Difference Uncertainties as a Signal for Exploration. *arXiv preprint arXiv:2010.02255*, 2020.

J Foerster. *Deep multi-agent reinforcement learning*. PhD thesis, University of Oxford, 2018.

Jakob Foerster, Gregory Farquhar, Maruan Al-Shedivat, Tim Rocktäschel, Eric Xing, and Shimon Whiteson. DiCE: The infinitely differentiable monte carlo estimator. In *International Conference on Machine Learning*, pages 1529–1538. PMLR, 2018.

Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2019.

Stan Franklin and Art Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 21–35. Springer, 1996.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

Yarin Gal. *Uncertainty in deep learning*. PhD thesis, University of Cambridge, 2016.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning (ICML)*, pages 1050–1059, 2016.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pages 10835–10866. PMLR, 2023.

Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018.

Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: Theory and practice—A survey. *Automatica*, 25(3):335–348, 1989.

Dibya Ghosh, Marlos C Machado, and Nicolas Le Roux. An operator view of policy gradient methods. *Advances in Neural Information Processing Systems*, 33:3397–3406, 2020.

Itzhak Gilboa, Andrew Postlewaite, and David Schmeidler. Is it always rational to satisfy Savage's axioms? *Economics & Philosophy*, 25(3):285–296, 2009.

Adam Gleave and Geoffrey Irving. Uncertainty estimation for language reward models. *arXiv preprint arXiv:2203.07472*, 2022.

Jordi Grau-Moya, Grégoire Delétang, Markus Kunesch, Tim Genewein, Elliot Catt, Kevin Li, Anian Ruoss, Chris Cundy, Joel Veness, Jane Wang, et al. Beyond Bayes-optimality: meta-learning what you know you don't know. *arXiv preprint arXiv:2209.15618*, 2022.

Alex Graves. Practical variational inference for neural networks. In *Neural Information Processing Systems (NeurIPS)*, pages 2348–2356, 2011.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. Deep autoregressive networks. In *International Conference on Machine Learning*, pages 1242–1250. PMLR, 2014.

Karol Gregor, Danilo Jimenez Rezende, Frederic Besse, Yan Wu, Hamza Merzic, and Aaron van den Oord. Shaping belief states with generative environment models for rl. *arXiv preprint arXiv:1906.09237*, 2019.

Christopher Grimm, André Barreto, Satinder Singh, and David Silver. The value equivalence principle for model-based reinforcement learning. *arXiv preprint arXiv:2011.03506*, 2020.

Daniel H Grollman and Aude Billard. Donut as I do: Learning from failed demonstrations. In *2011 IEEE International Conference on Robotics and Automation*, pages 3804–3809. IEEE, 2011.

Arthur Guez, Fabio Viola, Théophane Weber, Lars Buesing, Steven Kapturowski, Doina Precup, David Silver, and Nicolas Heess. Value-driven hindsight modelling. *arXiv preprint arXiv:2002.08329*, 2020.

Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Bernardo A Pires, and Rémi Munos. Neural predictive belief representations. *arXiv preprint arXiv:1811.06407*, 2018.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019b.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

Jessica B Hamrick, Abram L Friesen, Feryal Behbahani, Arthur Guez, Fabio Viola, Sims Witherspoon, Thomas Anthony, Lars Buesing, Petar Veličković, and Théophane Weber. On the role of planning in model-based deep reinforcement learning. *arXiv preprint arXiv:2011.04021*, 2020.

Steven Hansen, Will Dabney, Andre Barreto, Tom Van de Wiele, David Warde-Farley, and Volodymyr Mnih. Fast task inference with variational intrinsic successor features. *arXiv preprint arXiv:1906.05030*, 2019.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016b.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28, 2015.

Joseph Henrich. *The secret of our success: How culture is driving human evolution, domesticating our species, and making us smarter*. Princeton University Press, 2017.

Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. Agent modeling as auxiliary task for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, pages 31–37, 2019.

José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *International Conference on Machine Learning (ICML)*, pages 1861–1869, 2015.

TM Heskes. Solving a huge number of similar tasks: a combination of multi-task learning and a hierarchical bayesian approach, 1998.

Matteo Hessel, Ivo Danihelka, Fabio Viola, Arthur Guez, Simon Schmitt, Laurent Sifre, Theophane Weber, David Silver, and Hado van Hasselt. Muesli: Combining improvements in policy optimization. *arXiv preprint arXiv:2104.06159*, 2021.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13, 1993.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *arXiv preprint arXiv:1606.03476*, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, et al. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020.

Keith Hoskin. The 'awful idea of accountability': inscribing people into the measurement of objects. *Accountability: Power, ethos and the technologies of managing*, 265, 1996.

John Houston, Guido Zuidhof, Luca Bergamini, Yawei Ye, Long Chen, Ashesh Jain, Sammy Omari, Vladimir Iglovikov, and Peter Ondruska. One thousand and one hours: Self-driving motion prediction dataset. *arXiv preprint arXiv:2006.14480*, 2020.

National Highway Traffic Safety Administration. Pre-crash scenario typology for crash avoidance research, 2007.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

John D Hunter. Matplotlib: A 2D graphics environment. *IEEE Annals of the History of Computing*, 9(03):90–95, 2007.

Alekseĭ Grigor'evich Ivakhnenko and Valentin Grigorevich Lapa. *Cybernetic predicting devices*. Joint Publications Research Service, 1966.

Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Gordon Wilson. What are Bayesian neural network posteriors really like? In *International conference on machine learning*, pages 4629–4640. PMLR, 2021.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

William James. *The principles of psychology*, volume 1. Macmillan London, 1890.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

David Janz, Jiri Hron, Przemysław Mazur, Katja Hofmann, José Miguel Hernández-Lobato, and Sebastian Tschiatschek. Successor uncertainties: exploration and uncertainty in temporal difference learning. In *Advances in Neural Information Processing Systems*, pages 4507–4516, 2019.

Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International Conference on Machine Learning*, pages 3040–3049. PMLR, 2019.

Philipp Jund, Chris Sweeney, Nichola Abdo, Zhifeng Chen, and Jonathon Shlens. Scalable scene flow from point clouds in the real world. *IEEE Robotics and Automation Letters*, 7(2):1589–1596, 2021.

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

Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.

Daniel Kahneman. *Thinking, fast and slow*. Macmillan, 2011.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR, 2018.

Rudolf Emil Kalman. When is a linear control system optimal? *Journal of Basic Engineering*, 1964.

Gabriel Kalweit and Joschka Boedecker. Uncertainty-driven imagination for continuous deep reinforcement learning. In *Conference on Robot Learning*, pages 195–206. PMLR, 2017.

Hilbert J Kappen, Vicenç Gómez, and Manfred Opper. Optimal control as a graphical model inference problem. *Machine learning*, 87:159–182, 2012.

Zachary Kenton, Angelos Filos, Owain Evans, and Yarin Gal. Generalizing from a few environments in safety-critical reinforcement learning. *arXiv preprint arXiv:1907.01475*, 2019. Presented at the SafeML ICLR 2019 Workshop.

R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. Lyft Level 5 AV Dataset 2019, 2019.

Arne Kesting, Martin Treiber, and Dirk Helbing. Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1928): 4585–4605, 2010.

Hajime Kimura, Shigenobu Kobayashi, et al. An Analysis of Actor/Critic Algorithms Using Eligibility Traces: Reinforcement Learning with Imperfect Value Function. In *ICML*, volume 98, 1998.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015.

Edouard Klein, Bilal Piot, Matthieu Geist, and Olivier Pietquin. A cascaded supervised learning approach to inverse reinforcement learning. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 1–16. Springer, 2013.

Frank Hyneman Knight. *Risk, uncertainty and profit*, volume 31. Houghton Mifflin, 1921.

Mykel J Kochenderfer, Tim A Wheeler, and Kyle H Wray. *Algorithms for decision making*. MIT press, 2022.

Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zac Kenton, Jan Leike, and Shane Legg. Specification gaming: the flip side of AI ingenuity. *DeepMind Blog*, 3, 2020.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Panqanamala Ramana Kumar and Pravin Varaiya. *Stochastic systems: Estimation, identification, and adaptive control*. SIAM, 2015.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Neural Information Processing Systems (NeurIPS)*, pages 6402–6413, 2017.

Kevin N Laland. *Darwin's unfinished symphony: How culture made the human mind*. Princeton University Press, 2018.

Nathan Lambert, Brandon Amos, Omry Yadan, and Roberto Calandra. Objective mismatch in model-based reinforcement learning. *arXiv preprint arXiv:2002.04523*, 2020.

Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 29–37. JMLR Workshop and Conference Proceedings, 2011.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Alex X Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*, 2019a.

Donghun Lee, Srivatsan Srinivasan, and Finale Doshi-Velez. Truly batch apprenticeship learning with deep successor features. *arXiv preprint arXiv:1903.10077*, 2019b.

Adam Lerer, Hengyuan Hu, Jakob Foerster, and Noam Brown. Improving policies via search in cooperative partially observable games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7187–7194, 2020.

Edouard Leurent. An Environment for Autonomous Driving Decision-Making, 2018.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Lihong Li, Michael L Littman, and Thomas J Walsh. Knows what it knows: a framework for self-aware learning. In *Proceedings of the 25th international conference on Machine learning*, pages 568–575, 2008.

Zhihao Li, Toshiyuki Motoyoshi, Kazuma Sasaki, Tetsuya Ogata, and Shigeki Sugano. Rethinking self-driving: Multi-task knowledge for better generalization and accident explanation ability. *arXiv preprint arXiv:1809.11100*, 2018.

Litian Liang, Yaosheng Xu, Stephen McAleer, Dailin Hu, Alexander Ihler, Pieter Abbeel, and Roy Fox. Reducing variance in temporal-difference value estimation via ensemble of deep networks. In *International Conference on Machine Learning*, pages 13285–13301. PMLR, 2022.

Xiaodan Liang, Tairui Wang, Luona Yang, and Eric Xing. Cirl: Controllable imitative reinforcement learning for vision-based self-driving. In *European Conference on Computer Vision (ECCV)*, pages 584–599, 2018.

Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv preprint arXiv:1807.05118*, 2018.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

Robert K Lindsay, Bruce G Buchanan, Edward A Feigenbaum, and Joshua Lederberg. DENDRAL: a case study of the first expert system for scientific hypothesis formation. *Artificial intelligence*, 61(2):209–261, 1993.

Zachary Lipton, Yu-Xiang Wang, and Alexander Smola. Detecting and correcting for label shift with black box predictors. In *International conference on machine learning*, pages 3122–3130. PMLR, 2018.

Chenggang Liu and Christopher G Atkeson. Standing balance control using a trajectory library. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3031–3036. IEEE, 2009.

YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1118–1125. IEEE, 2018.

Alan J Lockett, Charles L Chen, and Risto Miikkulainen. Evolving explicit opponent models in game playing. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 2106–2113, 2007.

Manuel Lopes, Tobias Lang, Marc Toussaint, and Pierre-Yves Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Neural Information Processing Systems (NIPS)*, 2012.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. In *International Conference on Machine Learning*, pages 2218–2227. PMLR, 2017.

Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. *arXiv preprint arXiv:1811.01848*, 2018.

Yuping Luo, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. *arXiv preprint arXiv:1807.03858*, 2018.

Xiao Ma, Siwei Chen, David Hsu, and Wee Sun Lee. Contrastive variational model-based reinforcement learning for complex observations. *arXiv e-prints*, pages arXiv–2008, 2020.

Marlos C Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. Eigenoption discovery through the deep successor representation. *arXiv preprint arXiv:1710.11089*, 2017.

Marlos C Machado, Marc G Bellemare, and Michael Bowling. Count-based exploration with the successor representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5125–5133, 2020.

David JC MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Harry Markowitz. Portfolio Selection. *The Journal of Finance*, 7(1):77–91, 1952.

David A McAllester. Some pac-bayesian theorems. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 230–234, 1998.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

Kunal Menda, Katherine Driggs-Campbell, and Mykel J Kochenderfer. EnsembleDagger: A bayesian approach to safe imitation learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5041–5048. IEEE, 2019.

Rhiannon Michelmore, Marta Kwiatkowska, and Yarin Gal. Evaluating uncertainty quantification in end-to-end autonomous driving control. *arXiv preprint arXiv:1811.06817*, 2018.

Thomas P Minka. Expectation propagation for approximate Bayesian inference. *arXiv preprint arXiv:1301.2294*, 2001.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G Bellemare. Safe and efficient off-policy reinforcement learning. *arXiv preprint arXiv:1606.02647*, 2016.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

Kamal Ndousse, Douglas Eck, Sergey Levine, and Natasha Jaques. Multi-agent Social Reinforcement Learning Improves Generalization. *arXiv preprint arXiv:2010.00581*, 2020.

Radford M Neal. *Bayesian learning for neural networks*. Springer Science & Business Media, 1995.

Gergely Neu and Csaba Szepesvári. Training parsers by inverse reinforcement learning. *Machine learning*, 77:303–337, 2009.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pages 663–670, 2000.

Tianwei Ni, Harshit Sikchi, Yufei Wang, Tejus Gupta, Lisa Lee, and Ben Eysenbach. f-IRL: Inverse reinforcement learning via state marginal matching. In *Conference on Robot Learning*, pages 529–551. PMLR, 2021.

Evgenii Nikishin, Romina Abachi, Rishabh Agarwal, and Pierre-Luc Bacon. Control-Oriented Model-Based Reinforcement Learning with Implicit Differentiation. *arXiv preprint arXiv:2106.03273*, 2021.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. *arXiv preprint arXiv:1707.03497*, 2017.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. *Advances in neural information processing systems*, 29: 4026–4034, 2016.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *arXiv preprint arXiv:1806.03335*, 2018.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in neural information processing systems*, 32, 2019.

Tom Le Paine, Sergio Gómez Colmenarejo, Ziyu Wang, Scott Reed, Yusuf Aytar, Tobias Pfaff, Matt W Hoffman, Gabriel Barth-Maron, Serkan Cabi, David Budden, et al. One-shot high-fidelity imitation: Training large-scale deep nets with rl. *arXiv preprint arXiv:1810.05017*, 2018.

Tom Le Paine, Caglar Gulcehre, Bobak Shahriari, Misha Denil, Matt Hoffman, Hubert Soyer, Richard Tanburn, Steven Kapturowski, Neil Rabinowitz, Duncan Williams, et al. Making Efficient Use of Demonstrations to Solve Hard Exploration Problems. *arXiv preprint arXiv:1909.01387*, 2019.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pages 5062–5071. PMLR, 2019.

Tim Pearce, Felix Leibfried, and Alexandra Brintrup. Uncertainty in neural networks: Approximately bayesian ensembling. In *International conference on artificial intelligence and statistics*, pages 234–244. PMLR, 2020.

Tung Phan-Minh, Elena Corina Grigore, Freddy A Boulton, Oscar Beijbom, and Eric M Wolff. CoverNet: Multimodal Behavior Prediction using Trajectory Sets. *arXiv preprint arXiv:1911.10298*, 2019.

Alois Pichler and Ruben Schlotter. Entropy based risk measures. *European Journal of Operational Research*, 285(1):223–236, 2020.

Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Neural Information Processing Systems (NeurIPS)*, pages 305–313, 1989.

Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991.

John W Pratt. Risk aversion in the small and in the large. In *Uncertainty in economics*, pages 59–79. Elsevier, 1978.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Martin L Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24(11):1127–1137, 1978.

Neil C Rabinowitz, Frank Perbet, H Francis Song, Chiyuan Zhang, SM Eslami, and Matthew Botvinick. Machine theory of mind. *arXiv preprint arXiv:1802.07740*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Roberta Raileanu and Tim Rocktäschel. RIDE: Rewarding impact-driven exploration for procedurally-generated environments. *arXiv preprint arXiv:2002.12292*, 2020.

Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

Louis B Rall. *Automatic differentiation: Techniques and applications*. Springer, 1981.

Deepak Ramachandran and Eyal Amir. Bayesian Inverse Reinforcement Learning. In *IJCAI*, volume 7, pages 2586–2591, 2007.

Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736, 2006.

Siddharth Reddy, Anca D Dragan, and Sergey Levine. Sqil: Imitation learning via reinforcement learning with sparse rewards. *arXiv preprint arXiv:1905.11108*, 2019.

Danilo J Rezende, Ivo Danihelka, George Papamakarios, Nan Rosemary Ke, Ray Jiang, Theophane Weber, Karol Gregor, Hamza Merzic, Fabio Viola, Jane Wang, et al. Causally correct partial models for reinforcement learning. *arXiv preprint arXiv:2002.02836*, 2020.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

Nicholas Rhinehart, Rowan McAllister, Kris Kitani, and Sergey Levine. PRECOG: PREdiction Conditioned On Goals in Visual Multi-Agent Settings. *International Conference on Computer Vision*, 2019.

Nicholas Rhinehart, Rowan McAllister, and Sergey Levine. Deep Imitative Models for Flexible Inference, Planning, and Control. In *International Conference on Learning Representations (ICLR)*, April 2020.

Jacques Richalet, André Rault, JL Testud, and J Papon. Model predictive heuristic control. *Automatica (journal of IFAC)*, 14(5):413–428, 1978.

Mark Bishop Ring et al. Continual learning in reinforcement environments, 1994.

Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*, volume 6. International Conference on Representation Learning, 2018.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

German Ros, Vladlen Koltun, Felipe Codevilla, and M. Antonio Lopez. CARLA Challenge, 2019.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Artificial Intelligence and Statistics (AISTATS)*, pages 627–635, 2011.

Ariel Rubinstein. *Modeling bounded rationality*. MIT press, 1998.

Håvard Rue, Sara Martino, and Nicolas Chopin. Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 71(2):319–392, 2009.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Stuart Russell. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103, 1998.

Stuart Russell. *Human compatible: Artificial intelligence and the problem of control*. Penguin, 2019.

Doyen Sahoo, Quang Pham, Jing Lu, and Steven CH Hoi. Online deep learning: Learning deep neural networks on the fly. *arXiv preprint arXiv:1711.03705*, 2017.

A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959. doi: 10.1147/rd.33.0210.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

Axel Sauer, Nikolay Savinov, and Andreas Geiger. Conditional affordance learning for driving in urban environments. *arXiv preprint arXiv:1806.06498*, 2018.

Leonard J Savage. *The foundations of statistics*. Wiley Publications in Statistics, 1954.

Frederic Schick. Consistency and rationality. *The Journal of philosophy*, 60(1):5–19, 1963.

Jürgen Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *1990 IJCNN international joint conference on neural networks*, pages 253–258. IEEE, 1990.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Dominic Scott. *Plato's Meno*. Cambridge University Press, 2006.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR, 2020.

Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE, 2018.

Kyriacos Shiarlis, Joao Messias, and SA Whiteson. Inverse reinforcement learning from failure, 2016.

Edward H Shortliffe. Mycin: A knowledge-based computer program applied to infectious diseases. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, page 66. American Medical Informatics Association, 1977.

Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International conference on machine learning*, pages 5779–5788. PMLR, 2019.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David Silver, Hado Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, pages 3191–3199. PMLR, 2017.

Leo Smigel. Algorithmic Trading History: A Brief Summary, 2022.

Jasper Snoek, Yaniv Ovadia, Emily Fertig, Balaji Lakshminarayanan, Sebastian Nowozin, D Sculley, Joshua Dillon, Jie Ren, and Zachary Nado. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *Neural Information Processing Systems (NeurIPS)*, pages 13969–13980, 2019.

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

Dale O Stahl. Evolution of smart-n players. *Games and Economic Behavior*, 5(4):604–617, 1993.

Masashi Sugiyama and Motoaki Kawanabe. *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT Press, 2012.

Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in Perception for Autonomous Driving: An Open Dataset Benchmark. *arXiv preprint arXiv:1912.04838*, 2019.

Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020.

Wen Sun, Geoffrey J Gordon, Byron Boots, and J Bagnell. Dual policy iteration. *Advances in Neural Information Processing Systems*, 31, 2018.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

Richard S Sutton. TD models: Modeling the world at a mixture of time scales. In *Machine Learning Proceedings 1995*, pages 531–539. Elsevier, 1995.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

George T Taoka. Brake reaction times of unalerted drivers. *ITE journal*, 59(3):19–21, 1989.

Matthew E Taylor, Halit Bener Suay, and Sonia Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 617–624, 2011.

Robert Tibshirani. A comparison of some error estimates for neural network models. *Neural Computation*, 8(1):152–163, 1996.

Tishby, Levin, and Solla. Consistent inference of probabilities in layered networks: predictions and generalizations. In *International 1989 joint conference on neural networks*, pages 403–409. IEEE, 1989.

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

Harm Van Seijen, Hado Van Hasselt, Shimon Whiteson, and Marco Wiering. A theoretical and empirical analysis of Expected Sarsa. In *2009 ieee symposium on adaptive dynamic programming and reinforcement learning*, pages 177–184. IEEE, 2009.

Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 1971.

Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

Nino Vieillard, Tadashi Kozuno, Bruno Scherrer, Olivier Pietquin, Rémi Munos, and Matthieu Geist. Leverage the average: an analysis of kl regularization in reinforcement learning. *Advances in Neural Information Processing Systems*, 33:12163–12174, 2020.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton university press, 1944.

Abraham Wald. Contributions to the theory of statistical estimation and testing hypotheses. *The Annals of Mathematical Statistics*, 10(4):299–326, 1939.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. King's College, Cambridge United Kingdom, 1989.

Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. *arXiv preprint arXiv:1506.07365*, 2015.

Paweł Wawrzyński. Real-time reinforcement learning by sequential actor–critics and experience replay. *Neural networks*, 22(10):1484–1497, 2009.

Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. *arXiv preprint arXiv:1301.2315*, 2001.

Théophane Weber, Nicolas Heess, Lars Buesing, and David Silver. Credit assignment techniques in stochastic computation graphs. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2650–2660. PMLR, 2019.

Florian Wenzel, Kevin Roth, Bastiaan S Veeling, Jakub Swiatkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. How good is the bayes posterior in deep neural networks really? *arXiv preprint arXiv:2002.02405*, 2020.

Paul J Werbos. Learning how the world works: Specifications for predictive networks in robots and brains. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics, NY*, 1987.

Peter Whittle. Risk-sensitive linear/quadratic/Gaussian control. *Advances in Applied Probability*, 13(4):764–777, 1981.

Jörg Wichard, Christian Merkwirth, and Maciej Ogorzalek. Building ensembles with heterogeneous models, 2003.

Bernard Widrow. Pattern recognition and adaptive control. *IEEE Transactions on Applications and Industry*, 83(74):269–277, 1964.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *arXiv preprint arXiv:2002.08791*, 2020.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Wako Yoshida, Ray J Dolan, and Karl J Friston. Game theory of mind. *PLoS Comput Biol*, 4(12):e1000254, 2008.

Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

Tao Yu, Cuiling Lan, Wenjun Zeng, Mingxiao Feng, Zhizheng Zhang, and Zhibo Chen. Playvirtual: Augmenting cycle-consistent virtual trajectories for reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016.

Jiangchuan Zheng, Siyuan Liu, and Lionel M Ni. Robust bayesian inverse reinforcement learning with sparse behavior noise. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

Allan Zhou, Eric Jang, Daniel Kappler, Alex Herzog, Mohi Khansari, Paul Wohlhart, Yunfei Bai, Mrinal Kalakrishnan, Sergey Levine, and Chelsea Finn. Watch, try, learn: Meta-learning from demonstrations and reward. *arXiv preprint arXiv:1906.03352*, 2019.

Kemin Zhou, JC Doyle, and Keither Glover. Robust and optimal control. *Control Engineering Practice*, 4(8):1189–1190, 1996.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.