On Structural and Temporal Credit Assignment in Reinforcement Learning

by

Arash Tavakoli

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

Dyson School of Design Engineering

Imperial College London

London, UK

December 2021

Statement of Originality

I hereby declare that this thesis is my own work, except where specific reference is made to the work of others or where work done in collaboration with others is acknowledged.

Arash Tavakoli
December 2021

Copyright Declaration

Acknowledgements

This thesis would not have been possible without the support of my mentors, peers, friends, and family. First, I would like to express my sincere gratitude to my advisor Petar Kormushev for providing me with the opportunity to pursue this research and his guidance. I would also like to extend special thanks to my thesis committee members, Karl Tuyls and Sarah Filippi, and my candidacy and proposal evaluators, Marc Deisenroth and András György. I am deeply grateful to my collaborators Fabio Pardo, Mehdi Fatemi, Harm van Seijen, and Vitaly Levdik. I am extremely thankful for the support and friendship from my peers Nemanja Rakicevic and Sina Ghiassian. Additional thanks are due to Richard Sutton for hosting me during a research visit. My interactions with him and others at the RLAI lab are among the most memorable and rewarding experiences of my doctoral journey.

I am grateful to my friends Amirpasha, Meysam, Sam, and Erfan for their constant encouragement throughout the years. I am hugely indebted to my family members Roya, Ali, Shahin, Amin, and Roozbeh for their guidance, unparalleled support, immense enthusiasm in my every endeavour, and for being a great source of inspiration throughout my life. Last but not least, my deepest gratitude goes to my wife Natalie. She has been my comrade through each step of this doctoral journey, every idea, challenge, and success.

Abstract

Reinforcement learning, or learning how to map situations to actions that maximise a numerical reward signal, poses two fundamental interdependent problems: exploration and credit assignment. The *exploration* problem concerns an agent's ability to discover useful experiences. The *credit assignment* problem pertains to an agent's ability to incorporate the discovered experiences. The latter comprises two distinct subproblems itself: structural and temporal credit assignment. The *structural credit assignment* problem involves determining how to assign credit for the outcome of an action to the many component structures, or internal decisions, that could have been involved in producing that action. The *temporal credit assignment* problem has to do with determining how to assign credit for outcomes of a sequence of experiences to the actions that could have contributed to those outcomes. In this thesis, we broadly study the credit assignment problem in reinforcement learning, making contributions to each of its subproblems in isolation.

In the first part of this thesis we address the reinforcement learning problem in environments with multi-dimensional discrete action spaces, a problem setting that plagues structural credit assignment, or generalisation, due to the Bellman's *curse of dimensionality.* We argue that leveraging the combinatorial structure of such action spaces is crucial for achieving rapid generalisation from limited data. To this end, we introduce two approaches for estimating action values that feature a capacity for leveraging such structures, in each case empirically validating that significant performance improvements in sample complexity can be gained. Furthermore, we demonstrate that our approaches unleash significant benefits concerning space and time complexity, thus allowing them to successfully scale to high-dimensional discrete action spaces where the conventional approach becomes computationally intractable.

In the second part of this thesis we address the temporal credit assignment

problem. Specifically, we identify and analyse general training scenarios where appropriate temporal credit assignment is hindered by the mishandling of *time limits* or by the choice of *discount factor*. To address the first matter, we formalise the ways in which time limits may be interpreted in reinforcement learning and how they should be handled in each case accordingly. To address the second matter, we produce a possible explanation for why the performance of low discount factors tends to fall flat when used in conjunction with function approximation. In turn, this leads us to develop a method that enables a much larger range of discount factors by rectifying the hypothesised root cause.

List of Publications

The following is a list of the author's peer-reviewed research completed during the doctoral studies. The works that are not discussed in this thesis are indicated. The author's contributions are stated for the works on which the thesis is based and where the author is not the principal contributor.

1. **Tavakoli, A.**, Pardo, F., and Kormushev, P. (2018). Action branching architectures for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4131–4138.

2. Pardo, F., **Tavakoli, A.**, Levdik, V., and Kormushev, P. (2018). Time limits in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 4045–4054.

   $\rightarrow$ The author contributed to designing the experiments, discussing and analysing the results, and writing the manuscript.

3. **Tavakoli, A.**, Levdik, V., Islam, R., Smith, C. M., and Kormushev, P. (2019). Exploring restart distributions. In *Multidisciplinary Conference on Reinforcement Learning and Decision Making.*

   $\rightarrow$ This work is not discussed in the thesis.

4. van Seijen, H., Fatemi, M., and **Tavakoli, A.** (2019). Using a logarithmic mapping to enable lower discount factors in reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 14134–14144.

   $\rightarrow$ The author contributed to discussing and analysing the results, developing the code, running the experiments, and writing the manuscript.

5. Kriváchy, T., Cai, Y., Cavalcanti, D., **Tavakoli, A.**, Gisin, N., and Brunner, N. (2020). A neural network oracle for quantum nonlocality problems in networks. *npj Quantum Information* 6(1), pp. 1–7.

   $\rightarrow$ This work is not discussed in the thesis.

6. **Tavakoli, A.**, Fatemi, M., and Kormushev, P. (2021). Learning to represent action values as a hypergraph on the action vertices. In *Proceedings of the International Conference on Learning Representations*.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Intelligence

We refer to ourselves as *Homo sapiens*, which is Latin for “wise man,” signifying how important we think of our intelligence in making us special (Russell and Norvig, 2009). This then begs the curious question of what defines intelligence. In truth, there is not yet a clear consensus on a concrete definition of intelligence. Nevertheless, the definition given by McCarthy (2007, p. 2) is considered generally satisfactory: “Intelligence is the computational part of the ability to achieve goals in the world.” This concise definition of intelligence admits some of its widely accepted facets: intelligence is a matter of degree, concerns only computational competency, and manifests itself in outcomes rather than mechanisms (Sutton, 2020).

1.2 Artificial Intelligence

Understanding intelligence has long been a scholarly pursuit, studied in fields such as philosophy, psychology, and neuroscience. The related field of artificial intelligence (AI) differs from these fields in that it tries not only to understand but also to build intelligent entities (Russell and Norvig, 2009). Furthermore, AI is not restricted per se to mechanisms underlying intelligence in biological systems. This is in contrast to these other fields, at least in their traditional context, which study intelligence from the perspective of biological systems, and primarily humans. Nonetheless, building strong intelligent entities without drawing inspiration from biological systems is a daunting task, given the search space of possible solutions is enormous and very sparse (Hassabis et al., 2017). Hence, it is common in AI research to take inspiration and use findings in fields studying biological cognition. In fact, the origins of the AI methods on which this thesis is founded lie in such research: reinforcement learning, which was originally inspired by research into animal learning, and deep learning, which took root in ideas from neuroscience.

1.3 Learning from Interaction

McCarthy's definition of intelligence does not specify how "the ability to achieve goals" comes into existence. In biological systems, such ability is partly innate and partly acquired, with the ability to acquire being at least initially innate. The process of acquiring the ability to achieve goals is referred to as *learning*. The first idea to occur to us when we think about learning in the world, and one that underlies nearly all theories of learning and intelligence, is learning from *interaction* (Sutton and Barto, 2018). This is simply the idea of an entity, which can sense aspects of their environment and can influence it through their actions, learning to achieve goals by trial and error. A learning entity as such is often referred to as a learning *agent*[1] due to the fact that it can make decisions and enact them on the world, with "agent" literally meaning "the one who acts" (Harutyunyan, 2020).

1.4 Reinforcement Learning

The notion of "achieving goals" is at the heart of McCarthy's definition of intelligence. However, this definition abstracts away how a goal should be specified. Reinforcement learning provides a formalism for learning from interaction in which the agent's goal is specified as maximising a cumulative numerical signal, called the *reward*. The use of a reward signal to formalise the idea of a goal is one of the most distinctive features of reinforcement learning. The generality of formalising the idea of a goal in this way relies on the *reward hypothesis*:

> That all of what we mean by goals and purposes can be well thought of as the maximisation of the expected value of the cumulative sum of a received scalar signal (Sutton and Barto, 2018, p. 53).

In this thesis, we assume that the reward hypothesis holds and instead focus on building learning agents that become increasingly intelligent by learning

[1]In this thesis we frequently use "agent" as a convenient shorthand for a learning agent.

"the ability to achieve goals" through reinforcement. In other words, we aim to build goal-directed agents that learn to map perceived environment's states to *optimal* actions, ones that maximise the expected total reward (or some cumulative measure in terms of rewards) over the long run.

1.4.1 Policies, Value Functions, and Models

One can identify three main subelements of a reinforcement learning agent: a *policy*, a *value function*, and a *model* of the environment.

The behaviour of an agent in any state is defined by its *policy*, the inevitable subelement that governs the agent's side of the agent-environment interaction process. Sometimes the agent's policy is *directly* represented, where the other subelements could be used to facilitate learning this policy. Whereas other times the policy is inferred *indirectly* from the agent's other subelements.

A *value function* specifies the total amount of reward an agent can expect to accumulate over the future, starting from a state (in the case of a *state*-value function) or a state-action pair (in the case of an *action*-value function). By estimating the latter, the agent's best action in any given state can be simply found by maximising the predicted action values in that state. On the other hand, if only state values are estimated, then either a model of the environment or a direct policy is needed for action selection.

A *model* is something that mimics the behaviour of the environment. For example, a model might predict the expected next state and reward from a given state and action. A learning agent may utilise a model of the environment to facilitate learning a direct policy or a value function, or to decide on a course of action by deliberating about possible futures before they occur (also known as *planning*). Methods for solving reinforcement learning problems that use models are termed *model-based* methods, as opposed to simpler *model-free* methods which do not.

1.4.2 Lookup Tables and Function Approximation

The functions of a reinforcement learning agent may be represented using two fundamental means: a *lookup table* or a *function approximator*.

For example, a *lookup table* can be used to estimate a state-value function or an action-value function, respectively, in environments with finite state or state-action spaces. In this way the desired function can be represented *exactly*. However, in most realistic problems the state space, action space, or both are combinatorial and enormous. In such cases the memory required to store the desired function in a lookup table becomes intractable. An even more pressing issue with lookup tables is that of *generalisation*. For instance, a lookup table does not allow the agent to usefully generalise its experience with a limited subset of the state space to produce good predictions for a much larger subset.

To resolve the aforesaid issues, an *approximation* of a desired function may be constructed by learning an appropriate set of weights for the parameters of a *function approximator*. This also enables coping with continuous states and actions, extending the application of reinforcement learning beyond environments with finite state or state-action spaces. The topic of generalisation using function approximation has been extensively studied under the *supervised learning* paradigm in machine learning research. In theory, any of the methods studied in supervised learning can be used for function approximation within reinforcement learning. Of particular importance in the context of function approximation are neural network, or deep learning, methods (LeCun et al., 2015; Schmidhuber, 2015). The combination of reinforcement learning with deep learning, widely called deep reinforcement learning, enables building general-purpose agents which are able to generalise in enormous environments by learning about the underlying structures in samples. It is worth mentioning that neural networks are general nonlinear function approximators which subsume lookup tables and linear function approximation as special cases.

1.4.3 Exploration and Credit Assignment

It is often useful to decompose the reinforcement learning problem into two fundamental interdependent subproblems: *exploration* and *credit assignment*. The *exploration* problem refers to aspects of the reinforcement learning problem that have to do with determining how to discover useful experiences. The *credit assignment* problem concerns the aspects of the reinforcement learning problem that have to do with determining how to best incorporate the discovered experiences into the agent.

1.4.4 Structural and Temporal Credit Assignment

The problem of credit assignment itself consists of two distinct subproblems: *structural* and *temporal* credit assignment (Sutton, 1984; Silver, 2018).

The *structural credit assignment* problem refers to aspects of the credit assignment problem that concern determining how to distribute credit for success (or blame for failure) of an action among the many internal structures that could have been involved in producing it (Sutton and Barto, 2018). The forenamed subproblem is not unique to reinforcement learning. Rather it is intimately tied to the problem of *generalisation*, which is primarily studied under the supervised learning paradigm. This is evident in the multitude of supervised learning methods that are used in reinforcement learning for this purpose. For example, a fundamental solution method for this problem is *backpropagation* in neural networks (Goodfellow et al., 2016). Furthermore, combining backpropagation with *structured* neural networks such as recurrent, convolutional, and graph networks facilitates generalisation or structural credit assignment by forming useful internal structures.

The *temporal credit assignment* problem refers to aspects of the credit assignment problem that have to do with determining how to assign credit for outcomes of a sequence of experiences to actions that contributed to those outcomes. Unlike the structural credit assignment problem, the temporal version of the problem is mostly unique to reinforcement learning. One fundamental solution method for this problem is *temporal-different learning* (Sutton, 1988). Additionally, almost any temporal-difference method, such as Q-Learning or Sarsa, can be combined with *eligibility traces* to further facilitate temporal credit assignment.

1.5 Contributions

In this thesis we separately study each subproblem of the credit assignment problem. In each case we identify a number of general obstacles and propose ideas for addressing them. These ideas primarily contribute by furthering our understanding about useful mechanisms for estimating action values, with the exception of Chapter 5 which presents ideas that apply more broadly. Furthermore, given our motivation to build agents that learn to act optimally in order to achieve goals, we mainly focus on methods for finding the optimal action-value function and thus an optimal policy. The prime example of such a method is Q-Learning (Watkins, 1989). Hence, throughout this thesis we examine our ideas in combination with Q-Learning, using either lookup tables or function approximation to represent the action-value function.

In the first part of this thesis we study the structural version of the credit assignment problem. We argue that while there is a significant overlap between structural credit assignment in supervised learning and reinforcement learning, the latter poses a number of additional issues that do not normally arise in conventional supervised learning. We identify one such challenge to emerge in environments with multi-dimensional discrete action spaces, where structural credit assignment is rapidly impeded with increasing action-space cardinality. While there exists a myriad of supervised learning methods for addressing the structural credit assignment problem in enormous and combinatorial *state* spaces, tackling similar issues in *action* spaces has received comparatively little attention. This is likely due to the fact that the notion of *agency* is unique to reinforcement learning, and that structural credit assignment is less often explored in the reinforcement learning context. We propose two approaches to bridge this gap.

Firstly, we consider an initial approach for leveraging the combinatorial structure of multi-dimensional discrete action spaces in order to enable fast structural credit assignment from limited data. Our approach relies on the factorisation of multi-dimensional discrete action spaces and learning action values for each action dimension in a decentralised manner (much like a team of cooperative reinforcement learners). We demonstrate that this approach can significantly improve structural credit assignment by enabling more updates for insufficiently-explored, or even unexplored, actions.

Secondly, following the success of our initial approach, we further argue for the usefulness of enabling higher-order combinations of the action dimensions to also contribute to estimation of action values. Moreover, the inclusion of higher-order combinations expands the space of possible action-value functions that can be accurately represented. To enable such a capacity we resort to an alternative approach, formulating the problem as a representation learning one. Specifically, we frame the problem as learning a decomposition of the action-value function that is structured in such a way to leverage the combinatorial structure of multi-dimensional discrete actions at various orders. We present this approach as a framework to flexibly build architectures that leverage such structures. These architectures can be combined in succession with those for learning state representations and trained end-to-end using backpropagation, without imposing any change to the reinforcement learning method. Our results advocate for the general usefulness of leveraging the combinatorial structure of multi-dimensional discrete actions at various orders.

In the second part of this thesis we study the temporal version of the credit assignment problem. We identify and analyse general training scenarios where appropriate temporal credit assignment is hindered either due to using time limits during the interaction process or due to short effective time-horizons (as established by low discount factors). We clarify the root causes for these shortcomings and propose approaches for addressing them.

Firstly, we study scenarios where the agent-environment interaction is broken down into episodes by using time limits, or the maximum amount of time an interaction sequence can last. We identify two ways in which time limits can be interpreted in this context and describe how to appropriately handle each case. Specifically, if the agent must in fact maximise its performance over the time-limited period, we argue that a notion of the remaining time should be included as part of the agent's input. However, if the agent must maximise its performance beyond the time limit, where time limits are only used to diversify the agent's experience, we argue for bootstrapping at states where termination is solely due to reaching a time limit. We discuss the importance of these strategies for correct temporal credit assignment and thus for avoiding learning instabilities and suboptimal policies.

Secondly, we investigate why the the performance of low discount factors tends to fall flat when used in conjunction with function approximation, especially in tasks with long horizons. To do so, we analyse the effect of the discount factor on the optimisation process using a number of experiments. Our analysis leads to refuting a number of common hypotheses about this phenomenon. In turn, we set forth an alternative hypothesis that identifies the difference of the action gap—the optimal value difference between the best and second-best actions—across the state space as the root cause. To test this hypothesis, we introduce a method that achieves more homogeneous action gaps by mapping the action-value estimates to a logarithmic space and performing updates in that space instead. We empirically demonstrate that the proposed method indeed enables successful learning for the combination of low discount factors and function approximation, thus providing supporting evidence for our hypothesis.

1.6 Overview by Chapters

This thesis is partitioned into a shared background chapter followed by two orthogonal parts, each addressing one of the subproblems of credit assignment in reinforcement learning.

Background in Reinforcement Learning

→ Chapter 2 is a brief review of the key concepts in reinforcement learning, spanning from a formal introduction to the reinforcement learning problem to a description of the solution methods that are used throughout the thesis.

Part I Structural Credit Assignment

In this part we present our contributions to the structural credit assignment problem.
→ Chapter 3 explores a decentralised learning approach for exploiting the combinatorial structure of multi-dimensional action spaces at the lowest order.
→ Chapter 4 introduces a general framework for learning action representations that leverage the combinatorial structure of multi-dimensional action spaces at arbitrary orders.

Part II Temporal Credit Assignment

In this part we present our contributions to the temporal credit assignment problem.
→ Chapter 5 formalises the ways in which time limits may be interpreted in reinforcement learning and how they should be handled in each case.
→ Chapter 6 investigates why the combination of low discount factors and function approximation tends to fail in practice, leading us to develop a method for resolving it.

Chapter 2

Background in Reinforcement Learning

In this chapter we will describe the reinforcement learning problem as well as some fundamental solution methods for addressing it. For a broad coverage of reinforcement learning, we refer the reader to the books by Sutton and Barto (2018) and Szepesvári (2010).

2.1 The Reinforcement Learning Problem

The reinforcement learning problem is learning how to map *situations* to actions by interacting with an environment in order to maximise a cumulative numerical reward signal. We refer to the decision-making learner as the *agent*. The *environment* is everything outside of the agent. The *reward signal* defines the goal of a reinforcement learning problem whereby the agent attempts to maximise the total reward it receives over the long run.

We now define the interaction between the agent and environment in a general form. At every given time the agent observes the state of its environment to some extent and in response produces an action according to its behaviour policy. In turn, the environment transitions to a next state with a corresponding next observation and a numerical reward for the agent.

To make the reinforcement learning problem concrete, we need to establish what we mean by "situations" and how they relate to the agent's observations. To do so, we need to define two related notions: the *environment state* and *agent state*. The *environment state* is whatever information the environment uses to transition to a next state and to emit a next reward. The agent can usually observe this information only partially. Therefore, while in actuality the environment state fully represents the situation, it cannot be expected from the agent to learn a mapping from what is not observable. Nevertheless, this need not mean that an agent's interpretation of the situation at a given time can only be as good as its observation at that time. In fact, the agent can use whatever information that is observable to construct an internal state. In other words, the internal state can be any function of the *history*, the sequence of observations, actions, and rewards up to the present time. We refer to any such internal state as the *agent state*.

2.2 The Problem Formulation

The environment of a reinforcement learning problem can be modelled, in a mathematically idealised form, as a *Markov decision process*, or MDP (Puterman, 1994). An MDP is a quintuple $(\mathcal{S}, \mathcal{A}, P, R, P_0)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, P the state-transition distribution, R the reward distribution, and P_0 the initial-state distribution.

The *state space* is the set of all states, or all unique situations, the agent may find itself in while interacting with the environment. In the MDP formalism every state must be a sufficient statistic of the future, or satisfy the *Markov property*. Formally, a state $S_t \in \mathcal{S}$ encountered at time t is said to be *Markov* if and only if

$$Pr\{S_{t+1} \,|\, S_t\} = Pr\{S_{t+1} \,|\, S_0, S_1, \ldots, S_t\}.$$

As such, the environment state is Markov, whereas the agent state may or may not be Markov. Throughout this thesis we enforce, or otherwise assume, that the agent state is Markov and thus no longer distinguish between the environment state and agent state, referring to both simply as the *state*. That is to say, we generally do not consider the problem of how a Markov state can be constructed by the agent from non-Markov observations.

The *action space* is the set of all actions the agent can possibly take in any given state. We should note that the set of possible actions may differ from one state to another, but it is convenient to suppress any such differences by letting $\mathcal{A}$ denote the union of the state-dependent sets of actions:

$$\mathcal{A} \doteq \bigcup_{s \in \mathcal{S}} \mathcal{A}(s).$$

At a given time t the agent uses its behaviour policy $\pi(a \mid s)$, the probability that $A_t = a$ if $S_t = s$, to choose an action to perform in the environment:

$$\pi(a \mid s) \doteq Pr\{A_t = a \mid S_t = s\}.$$

The *state-transition distribution* P specifies the probability of transitioning to a next state s' given a state s and an action a:

$$P(s' \mid s, a) \doteq Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\}.$$

The *reward distribution* R specifies the probability of emitting a numerical reward r on transition to a next state s' from a state s due to an action a:

$$R(r \mid s, a, s') \doteq Pr\{R_{t+1} = r \mid S_t = s, A_t = a, S_{t+1} = s'\}.$$

The *initial-state distribution* P_0 specifies the probability of the interaction starting in a state s:

$$P_0(s) \doteq Pr\{S_0 = s\}.$$

Putting everything together, the interaction between the agent and MDP gives rise to a sequence of states, actions, and rewards that begins like this:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots.$$

2.2.1 Returns and Values

The outcome, or the *return*, of an interaction sequence is defined as some specific function of the reward sequence. In the simplest case the return is the sum of the reward sequence:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \ldots. \tag{2.1}$$

Mostly due to practical reasons, it is very common to adopt a discounted sum of the reward sequence, or the *discounted return*:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots, \tag{2.2}$$

where $0 \le \gamma \le 1$ is called the *discount factor*. Notice that the discounted return subsumes the undiscounted one as a special case for $\gamma = 1$. Hence, throughout this thesis we primarily use the discounted return formulation (2.2) and set $\gamma = 1$ wherever possible to achieve the undiscounted return.

We define the *state-value function* $V_\pi(s)$ as the function that gives the *expected return* from state s due to following policy π:

$$V_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s].$$

Similarly, we define the *action-value function* $Q_\pi(s, a)$ as the function that gives the *expected return* from action a at state s and then following policy π:

$$Q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a].$$

The *goal* of a reinforcement learning agent can now be more formally stated as finding a policy that maximises the values of states $S_0 \sim P_0(s)$.

2.2.2 Episodic and Continuing Tasks

In some cases the agent-environment interaction naturally breaks down into separate *episodes*, where an episode terminates due to reaching a *terminal state*. Tasks with episodes of this kind are referred to as *episodic tasks*. In such cases the episodic sequences of interaction take the following form:

$$S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{H-1}, A_{H-1}, R_H, S_H,$$

where H is a *finite* random variable commonly referred to as the *horizon*. In episodic tasks the goals are often best described using the undiscounted return formulation (2.1). However, whenever using function approximation we generally need to resort to the discounted return formulation (2.2) with $\gamma < 1$ to avoid some optimisation issues.[1]

On the other hand, in other cases the agent-environment interaction goes on *continually* without limit. That is, the horizon H is *infinite*. We call these *continuing tasks*. In such cases the undiscounted return formulation (2.1) is problematic as the sum of an infinite number of rewards could itself easily be infinite. By using $\gamma < 1$ the discounted return formulation (2.2) resolves this optimisation issue by ensuring bounded returns in continuing tasks as long as the rewards $\{R_k\}$ are bounded.

In this thesis we consider both episodic and continuing tasks, but we almost always use a *time limit* as an arbitrary termination condition to break down the agent-environment interaction into partial sequences or episodes.

[1]We refer the reader to the discussions by Durugkar and Stone (2017) and Pohlen et al. (2018) around such issues.

2.3 Optimality of Policies and Value Functions

A value function defines a partial ordering over policies. Specifically, as Sutton and Barto (2018, p. 62) put it: "A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states." There is always at least one policy that is better than or equal to all other policies. We refer to this as an *optimal policy*. While in general there may be more than one optimal policy, we denote all such policies by π^*. The optimal policies share the same state-value function, called the *optimal state-value function V^**:

$$V^*(s) \doteq V_{\pi^*}(s) = \max_{\pi} V_\pi(s),$$

for all $s \in \mathcal{S}$. The optimal policies also share the same action-value function, called the *optimal action-value function Q^**:

$$Q^*(s,a) \doteq Q_{\pi^*}(s,a) = \max_{\pi} Q_\pi(s,a),$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. Notice that indeed an optimal policy in the sense described here also maximises the values of states $S_0 \sim P_0(s)$, which is our goal in a reinforcement learning problem.

2.4 Bellman Equations

A fundamental property of value functions used throughout reinforcement learning is that they satisfy the following recursive relationships:

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s], \tag{2.3}$$

$$Q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]. \tag{2.4}$$

These are known as the *Bellman expectation equations* for V_π and Q_π. Using these equations we can derive the foundational equation underpinning the main reinforcement learning method that permeates throughout this thesis, namely Q-Learning. To this end, we begin by substituting Q_π in Equation (2.4) with Q^*:

$$Q^*(s,a) = \mathbb{E}_{\pi^*}[R_{t+1} + \gamma Q^*(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]. \tag{2.5}$$

As we stated in the previous section, there may be more than one optimal policy π^* which obtains Q^*. A well-known result is that, for any MDP, at least one optimal policy is *deterministic* and that all such policies can be extracted by maximising Q^*:

$$\pi^*(s) \in \arg\max_a Q^*(s,a). \tag{2.6}$$

By assuming a deterministic optimal policy π^*, we can rewrite Equation (2.5) (in which action A_{t+1} is drawn from π^*) as follows:

$$Q^*(s,a) = \mathbb{E}_{\pi^*}[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') \mid S_t = s, A_t = a]. \tag{2.7}$$

This is known as the *Bellman optimality equation* for Q^*.

It is worth mentioning that V^* could also be used to extract a deterministic optimal policy. However, in that case a model of the environment is required for performing one-step-ahead search. As stated in Equation (2.6), with Q^* the agent does not need a model to extract such a policy.

2.5 Temporal-Difference Learning

Temporal-difference learning, or TD learning, is a general method primarily for policy evaluation or *prediction*, the problem of estimating the value functions V_π or Q_π for a given policy π (Sutton, 1984; Sutton, 1988). TD learning uses *bootstrapping* ideas from dynamic programming (Bellman, 1957) to enable learning in an online manner (by updating a guess from a guess) as well as *sample-based value estimation* ideas from Monte Carlo methods (Sutton and Barto, 2018) to enable learning directly from experience without a model of the environment (by eliminating full backups).

In combination with variations of *generalised policy iteration* from dynamic programming, TD learning gives rise to the most widely used methods for *control*, the reinforcement learning problem of finding an optimal policy π^*.

In the next two subsections we will describe tabular *TD(0)*, the simplest TD method for prediction, and tabular *Q-Learning*, a canonical TD method for control. Then we will describe how one could go beyond the tabular case to function approximation and outline a successful procedure for combining Q-Learning with deep neural networks.

2.5.1 TD(0): One-Step TD Learning for Prediction

The simplest TD method is the tabular version of *TD(0)*, or *one-step TD*, for estimating the state-value function V_π of a given policy π. This TD method iteratively improves an estimate $V(S_t)$ of $V_\pi(S_t)$ by making the update

$$V(S_t) \leftarrow V(S_t) + \alpha_t\Big(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big) \tag{2.8}$$

immediately on transition to S_{t+1} and receiving R_{t+1}, due to performing A_t produced by π in S_t. Here α_t is a positive *learning rate.* The expression $\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is known as the (one-step) *TD error.* The expression $Y_t \doteq R_{t+1} + \gamma V(S_{t+1})$ is the *update target* for TD(0), the part of the update rule that distinguishes TD(0) from other TD methods for estimating V_π. In principle, this update rule transforms the Bellman expectation equation for V_π (2.3) into a sample-based iterative method for finding V_π at convergence.

To guarantee convergence with probability one to the ideal predictions V_π, Dayan (1992) (and later Jaakkola et al. (1994)) show that the sequence $\{\alpha_t\}$ must satisfy the Robbins-Monro conditions:

$$\sum_{t=0}^{\infty} \alpha_t = \infty \qquad \text{and} \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty. \tag{2.9}$$

Nevertheless, obtaining a sequence of learning rates that meets the conditions (2.9) with a satisfactory convergence rate is hard. Therefore, we seldom use a learning rate sequence that satisfies these conditions and, instead, use a small constant learning rate α that works well in practice. It is worth mentioning that TD(0) has been proven by Sutton (1988) to converge to V_π in the mean for a constant learning rate α provided that α is sufficiently small.

2.5.2 Q-Learning: Off-Policy TD Learning for Control

Considered one of the early breakthroughs in reinforcement learning, *Q-Learning* (Watkins, 1989) is an elegant TD method for control. The tabular version of Q-Learning iteratively improves an estimate $Q(S_t, A_t)$ of the optimal action value $Q^*(S_t, A_t)$ by making the update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)\Big). \tag{2.10}$$

Notably, this update rule allows for *off-policy* training for control; sample transitions from any behaviour policy can be used to improve the estimates Q towards Q^*.[2] In principle, this update rule transforms the Bellman optimality

[2]More broadly, *off-policy* training refers to learning about a *target policy* (e.g. estimating its corresponding value function) from data generated by a different *behaviour policy.* This is as opposed to *on-policy* training where data has to be generated according to the policy that is being evaluated or improved.

equation for Q^* (2.7) into a sample-based iterative method for finding Q^* at convergence.

Under a variant of the conditions (2.9) and the assumption that all state-action pairs continue to be visited and updated, Q has been shown to converge with probability one to Q^* (Watkins and Dayan, 1992; Jaakkola et al., 1994). To meet the latter assumption, an ε-greedy strategy can be used as the behaviour policy during training. That is, with probability $1 - \varepsilon$ an action that maximises Q is selected, and with probability ε an action is sampled uniformly at random from the set of possible actions. At convergence to Q^* a deterministic optimal policy can be extracted in a model-free manner via Equation (2.6). Similarly to our discussion for TD(0), in practice, we often resort to a small constant learning rate α that works well.

2.5.3 TD Learning with Function Approximation

In environments with continuous or large finite state spaces, it is not possible or practical to learn a value for each individual state or state-action pair. In such cases, it is necessary to represent the state more compactly by using some *feature vector* $\mathbf{x}(s) \doteq (x_1(s), x_2(s), \dots, x_d(s))$ for a given state s. In general, the feature vector should be chosen such that it reflects some structures in the state. For example, in a combinatorial state space with d dimensions, where each state s is the combination of d sub-state components, an obvious choice for the feature vector would be $\mathbf{x}(s) \doteq (s^1, s^2, \dots, s^d)$. Notice that in this case each feature can take the same value across many states, thus enabling the reinforcement learner to leverage the combinatorial structure of the state space and achieve generalisation.

A value function can then be approximated by a function of the features $\mathbf{x}(s)$ and parameters $\boldsymbol{\theta}$. One important special case of function approximation is to use a linear combination of features and parameters to approximate a value function, e.g. $V_{\boldsymbol{\theta}}(s) \doteq \mathbf{x}(s) \cdot \boldsymbol{\theta}$. Notice that in the case of linear function approximators the number of parameters is dictated by, and is equivalent to, the number of features. It is worth mentioning that linear function approximation subsumes table-lookup as a special case. Explicitly, if $\mathbf{x}(s)$ is a one-hot vector with cardinality $|\mathcal{S}|$ (the state-space size) and that for any state s all vector elements are zero except for the sth element, then the linear function approximator reduces to a simple table. Note that in this case, the notion of generalisation becomes irrelevant as an update at one state affects no other. Furthermore, the representational capacity of the function approximator can be increased by using a nonlinear parameterisation such as a neural network.

An approximate value function can be trained using the combination of TD learning, *backpropagation*, and some variation of *stochastic gradient descent*. Specifically, TD learning addresses the temporal credit assignment by providing an appropriate error to minimise. Backpropagation calculates the gradient of a function of the error with respect to the function approximator's parameters. Stochastic gradient descent, or some variation of it, is used to minimise the error by updating the parameters using the gradients calculated via backpropagation. The combination of backpropagation and stochastic gradient descent addresses the structural credit assignment by distributing credit for the error among the many parameters that were involved in producing it.

In the next subsection, we outline the procedure for training Q-Learning with function approximation and the considerations that have been shown to make the training stable, especially with deep neural networks.

2.5.4 Deep Q-Networks

An approximate value function can be trained by updating the parameters of a function approximator after each transition, and solely based on that transition, using TD learning to produce the error for each update. For example, in the case of Q-Learning, an approximate action-value function can be trained by minimising the squared TD error

$$\delta_t^2 \doteq \Big(R_{t+1} + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(S_{t+1}, a') - Q_{\boldsymbol{\theta}}(S_t, A_t)\Big)^2 \tag{2.11}$$

on each individual transition $(S_t, A_t, R_{t+1}, S_{t+1})$ immediately after being experienced. However, the combination of TD learning and function approximation has been shown to be unstable or to even diverge (Tsitsiklis and Van Roy, 1997; Sutton and Barto, 2018). More precisely, due to instabilities during the training process, the convergence of on-policy TD methods (for prediction or control) cannot be guaranteed when used in conjunction with nonlinear function approximators and the convergence of off-policy TD methods (for prediction or control) cannot be guaranteed when combined with linear or nonlinear function approximators (see Maei (2011) for a detailed overview). This instability has numerous causes. Here we outline these causes for Q-Learning, but similar arguments apply to the broader family of TD methods. First, due to the sequential nature of interactions in reinforcement learning, the observations are correlated. This in turn causes a high variance across the updates. Second, small updates to the action-value function may significantly change the behaviour policy. This in turn considerably changes the data distribution. Third, due to bootstrapping, the action values $Q_{\boldsymbol{\theta}}(S_t, A_t)$ and update targets $R_{t+1} + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(S_{t+1}, a')$ are correlated. This implies that updating the action values also changes the target values, thus introducing nonstationarity in the updates.

Mnih et al. (2015) address these instabilities by modifying the Q-Learning procedure in two key ways. First, they use a method called *experience replay* (first studied by Lin (1992)). This method stores the agent's experiences in a memory buffer from which minibatches of experience are drawn uniformly at random to update the action-value function. Note that learning by experience replay is admissible in the case of Q-Learning because it allows for off-policy training. Randomising over the experiences removes correlations in the observation sequence and smooths over changes in the data distribution.

Second, they modify the update targets to be based on an older set of parameters $\boldsymbol{\theta}^-$, thus adding a delay between the time an update to the action values is made and the time the update affects the target values:

$$\delta_t^2 \doteq \Big(R_{t+1} + \gamma \max_{a'} Q_{\boldsymbol{\theta}^-}(S_{t+1}, a') - Q_{\boldsymbol{\theta}}(S_t, A_t)\Big)^2, \tag{2.12}$$

where the target parameters $\boldsymbol{\theta}^-$ are updated with the online parameters $\boldsymbol{\theta}$ every C steps and are held fixed between individual updates. This modification reduces the correlations between the action values and update targets.

This modified Q-Learning procedure, termed *deep Q-networks* (DQN), has been demonstrated to enable training deep neural networks in a stable manner and achieve human-level play on the challenging domain of classic Atari 2600 games (from pixels) (Mnih et al., 2015).

2.6 Policy Gradient Methods

In this chapter we describe *policy gradient methods*, an alternative approach to solving the control problem in reinforcement learning (Sutton et al., 1999). Unlike the action-value methods described in Section 2.5, policy gradient methods do not require action-value estimates for selecting actions. Rather they work by explicitly representing the policy using a parameterised function approximator. However, as we will see later in this section, a value function may still be used to learn the policy parameters.

Given a parameterised policy $\pi_{\boldsymbol{w}}(a \,|\, s)$ with parameters $\boldsymbol{w} \in \mathbb{R}^d$, policy gradient methods seek to maximise a scalar performance measure $J(\boldsymbol{w})$ by updating the policy parameters using stochastic gradient ascent as follows:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \widehat{\nabla J(\boldsymbol{w})}. \tag{2.13}$$

Here $\widehat{\nabla J(\boldsymbol{w})} \in \mathbb{R}^d$ is a stochastic estimate whose expectation approximates the gradient of the performance measure with respect to the policy parameters $\boldsymbol{w}$. One natural choice for the performance measure is the expected state value under the current policy:

$$\begin{aligned} J(\boldsymbol{w}) \doteq \mathbb{E}_{\pi_{\boldsymbol{w}}}\Big[V_{\pi_{\boldsymbol{w}}}(S_t)\Big] &= \mathbb{E}_{\pi_{\boldsymbol{w}}}\left[\sum_{a \in \mathcal{A}} Q_{\pi_{\boldsymbol{w}}}(S_t, a)\, \pi_{\boldsymbol{w}}(a \,|\, S_t)\right] \\ &= \mathbb{E}_{\pi_{\boldsymbol{w}}}\left[Q_{\pi_{\boldsymbol{w}}}(S_t, A_t)\, \frac{\pi_{\boldsymbol{w}}(A_t \,|\, S_t)}{\lfloor \pi_{\boldsymbol{w}}(A_t \,|\, S_t) \rfloor}\right], \end{aligned} \tag{2.14}$$

where $V_{\pi_{\boldsymbol{w}}}$ and $Q_{\pi_{\boldsymbol{w}}}$ are, respectively, the state-value and action-value functions corresponding to policy $\pi_{\boldsymbol{w}}$ and $\lfloor \cdot \rfloor$ denotes the *stop gradient* operation. The *policy gradient theorem* (Sutton et al., 1999) provides an analytic expression for the gradient of this performance measure with respect to the policy parameters as follows:

$$\begin{aligned} \nabla J(\boldsymbol{w}) &= \nabla \mathbb{E}_{\pi_{\boldsymbol{w}}}\left[Q_{\pi_{\boldsymbol{w}}}(S_t, A_t)\, \frac{\pi_{\boldsymbol{w}}(A_t \,|\, S_t)}{\lfloor \pi_{\boldsymbol{w}}(A_t \,|\, S_t) \rfloor}\right] \\ &\propto \mathbb{E}_{\pi_{\boldsymbol{w}}}\left[Q_{\pi_{\boldsymbol{w}}}(S_t, A_t)\, \frac{\nabla \pi_{\boldsymbol{w}}(A_t \,|\, S_t)}{\pi_{\boldsymbol{w}}(A_t \,|\, S_t)}\right] \\ &= \mathbb{E}_{\pi_{\boldsymbol{w}}}\Big[Q_{\pi_{\boldsymbol{w}}}(S_t, A_t)\, \nabla \ln \pi_{\boldsymbol{w}}(A_t \,|\, S_t)\Big], \end{aligned} \tag{2.15}$$

where the symbol $\propto$ denotes “proportional to.” In episodic tasks the constant of proportionality is the average length of an episode, while in continuing tasks it is 1 (i.e. the relationship becomes an equality) (Sutton and Barto, 2018).

In practical scenarios this expectation cannot be evaluated exactly, rather it needs to be estimated from data generated by following the policy $\pi_{\boldsymbol{w}}$ in the environment. A key challenge is then how to best estimate this expression from limited data. In the next subsection we will describe a commonly used method that addresses this challenge by combining several algorithmic components.

2.6.1 Proximal Policy Optimisation

Proximal Policy Optimisation (PPO) (Schulman et al., 2017) is one of the most commonly used on-policy algorithms at the time of writing this thesis. In this subsection we will briefly describe the main algorithmic components that PPO combines to obtain good estimates of $\nabla J(\boldsymbol{w})$ from limited data.

Baseline Function

The policy gradient theorem can be generalised to include a comparison of the action value to an arbitrary *baseline* $b(s)$ (Williams, 1992):

$$\nabla J(\boldsymbol{w}) \propto \mathbb{E}_{\pi_{\boldsymbol{w}}}\Big[\big(Q_{\pi_{\boldsymbol{w}}}(S_t, A_t) - b(S_t)\big)\,\nabla \ln \pi_{\boldsymbol{w}}(A_t \mid S_t)\Big]. \tag{2.16}$$

The baseline can generally be any function as long as it does not change the expectation on the right-hand side (i.e. one that does not introduce *statistical bias*). This can be satisfied by ensuring that the baseline function does not vary with actions. For an action-independent baseline $b(s)$, Equation (2.16) remains valid because the subtracted quantity is zero in expectation:

$$\begin{aligned}
\mathbb{E}_{\pi_{\boldsymbol{w}}}\Big[b(S_t)\,\nabla \ln \pi_{\boldsymbol{w}}(A_t \mid S_t) \mid S_t = s\Big] &= \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{w}}(a \mid s)\, b(s)\, \nabla \ln \pi_{\boldsymbol{w}}(a \mid s) \\
&= \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{w}}(a \mid s)\, b(s)\, \frac{\nabla \pi_{\boldsymbol{w}}(a \mid s)}{\pi_{\boldsymbol{w}}(a \mid s)} \\
&= \sum_{a \in \mathcal{A}} b(s)\, \nabla \pi_{\boldsymbol{w}}(a \mid s) \\
&= b(s)\, \nabla \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{w}}(a \mid s) \\
&= b(s)\, \nabla 1 \\
&= 0.
\end{aligned} \tag{2.17}$$

Notice that in continuous action spaces the summation over discrete actions in the equation above turns into an integration over the continuous actions. While the baseline leaves the expected value of the update unchanged (as established above), it can greatly reduce its variance. One natural choice for the baseline is the state-value function $V_{\pi_{\boldsymbol{w}}}$, or in practice a learned estimator for it V. Subtracting the state-value function from the action-value function leaves a quantity that is commonly referred to as the *advantage function*:

$$A_{\pi_{\boldsymbol{w}}}(S_t, A_t) \doteq Q_{\pi_{\boldsymbol{w}}}(S_t, A_t) - V_{\pi_{\boldsymbol{w}}}(S_t), \tag{2.18}$$

where $\mathbb{E}_{\pi_{\boldsymbol{w}}}[A_{\pi_{\boldsymbol{w}}}(S_t, A_t)] = 0$. Intuitively, the advantage function gives a relative measure of the importance of each action at any given state. PPO estimates this quantity instead of the action values.

Likelihood Ratio

The policy gradient updates are restricted to data generated according to the agent's current policy $\pi_{\boldsymbol{w}}$. As such, the moment the policy gets updated by applying one step of gradient ascent, all interaction data up until then needs to be discarded as they no longer correspond to the agent's policy. This is highly sample inefficient. The primary variant of PPO optimises a clipped surrogate objective to address this shortcoming:

$$J_{\text{clip}}(\boldsymbol{w}) \doteq \mathbb{E}_{\pi_{\boldsymbol{\mu}}}\Big[\min\Big\{A_{\pi_{\boldsymbol{\mu}}}(S_t, A_t)\,\rho(\boldsymbol{w}),\ A_{\pi_{\boldsymbol{\mu}}}(S_t, A_t)\,\text{clip}\big(\rho(\boldsymbol{w}), 1-\epsilon, 1+\epsilon\big)\Big\}\Big], \tag{2.19}$$

where ϵ is a small scalar constant and $\rho(\boldsymbol{w}) \doteq \frac{\pi_{\boldsymbol{w}}(A_t\,|\,S_t)}{\lfloor \pi_{\boldsymbol{\mu}}(A_t\,|\,S_t)\rfloor}$ is the likelihood ratio. This altered objective serves by enabling *multiple* minibatch updates on the same batch of data, as opposed to performing a *single* update on the entire batch. More precisely, a small ϵ prohibits the policy from diverging far from the data-generating policy $\pi_{\boldsymbol{\mu}}$, thus asserting that the data remains approximately consistent with the policy after an update is applied. This clipped surrogate objective, which is the most distinctive feature of the PPO algorithm, has been empirically shown to improve sample complexity in comparison to the standard policy gradient objective.

Advantage Estimation

To achieve variance-reduced estimation of the advantage function $A_{\pi_{\boldsymbol{\mu}}}$, PPO makes use of a truncated variant of the *generalised advantage estimator* or *GAE*(λ) (Schulman et al., 2016) given by:

$$A(S_t, A_t) \doteq \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots + (\gamma\lambda)^{K-t-1}\delta_{K-1}, \tag{2.20}$$

where $A(S_t, A_t)$ is an estimate of $A_{\pi_{\boldsymbol{\mu}}}(S_t, A_t)$ and $\delta_i \doteq R_{i+1} + \gamma V(S_{i+1}) - V(S_i)$ is a one-step TD error, in which $V(S_i)$ is an estimator for $V_{\pi_{\boldsymbol{\mu}}}(S_i)$. The above equation calculates advantage estimates as a geometrically-weighted average of truncated k-step returns with decay parameter λ and truncation horizon K (i.e. maximum length of the trajectory segment). GAE(λ) is generally best viewed as a *TD(λ)* method for estimating the advantage function, with its truncated variant (discussed above) being analogous to the *truncated TD(λ)* algorithms (Sutton and Barto, 2018).

One natural choice for obtaining state-value estimates to compute the advantage estimates is to learn an estimator of the state value using the update target

$$Y_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{K-t-1} R_K + \gamma^{K-t} V(S_K). \tag{2.21}$$

Using this update target a tabular estimator can be trained by making the update $V(S_t) \leftarrow V(S_t) + \alpha(Y_t - V(S_t))$ and an approximate estimator can be trained by minimising the squared error $(Y_t - V_{\boldsymbol{\theta}}(S_t))^2$.

Part I

Structural Credit Assignment

This part of the thesis describes ideas for addressing the *structural* version of the credit assignment problem in reinforcement learning. That is to say, we study some aspects of the credit assignment problem that have to do with determining *which* internal structures deserve credit for influencing the performed behaviour (Sutton, 1984). In particular, the ideas described in this part of the thesis address this problem in the specific context of reinforcement learning in environments with *multi-dimensional discrete action spaces.*

We now explicate the target problem setting by revisiting the MDP formulation of the environment in reinforcement learning (presented in Section 2.2). The MDP formalism abstracts away the combination of sub-actions that are activated when a discrete action a is chosen. Specifically, in a problem with an N-dimensional action space a given action a maps onto an N-tuple $(a^1, a^2, \dots, a^N)$, where each a^i represents a discrete sub-action from the ith sub-action space. As such, the action space could have an underlying combinatorial structure whereby the set of actions is formed as a Cartesian product of the sub-action spaces. To make this property apparent in our notations, throughout this part of the thesis we express the action space as $\mathcal{A} \doteq \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_N$, where each $\mathcal{A}_i$ is a finite set of sub-actions. Moreover, we amend our notation for the actions a into $\mathbf{a}$ (in bold) to reflect that actions are generally the combinations of several sub-actions.

The theme of our contributions is the repurposing of existing approaches for cooperative multi-agent reinforcement learning (Panait and Luke, 2005; Buşoniu et al., 2008; Bloembergen et al., 2015) to address a number of challenges that are confronted by individual reinforcement learners in environments with multi-dimensional discrete action spaces. Particularly, we outline the specific challenges that arise in such environments, which are caused by the *curse of dimensionality* (Bellman, 1957), and address the aspects that are related to structural credit assignment. We argue that the key is in leveraging the combinatorial structure of such action spaces to enable rapid generalisation from limited data, thus leading us to introduce two approaches for estimating action values with such a capacity. Furthermore, we show that our approaches not only improve generalisation but also unleash significant benefits concerning space and time complexity.

Chapter 3

Action Branching Methods

3.1 Introduction

Solving the reinforcement learning problem in multi-dimensional discrete action spaces is plagued by the *curse of dimensionality* (Bellman, 1957): the number of actions increases exponentially with the number of action dimensions. For example, a system with 17 action dimensions and 5 sub-actions per action dimension implies an action-space size of 5^{17}, or approximately 800 billion actions. Such enormous action spaces, which are not uncommon, pose great challenges for exploration and credit assignment even in small finite state spaces. These challenges arise given the amount of experience that can be collected is often substantially smaller than the number of possible state-action pairs. In the example above if there is only one state (as in a multi-armed bandit problem) and if sampling a single transition takes 0.01 seconds, then trying each possible action only *once* takes more than 250 years. This is clearly not possible to achieve. Regardless, this number of samples is hardly enough for learning in multi-armed bandit problems, let alone in reinforcement learning problems with numerous states.

The limitations imposed by the curse of dimensionality are not specific to multi-dimensional action spaces. In fact, multi-dimensionality is similarly problematic in state spaces; take for example an agent that receives images as input, where the number of pixels gives the dimensionality with each pixel having a set of possible values. Nevertheless, advances in deep learning have enabled efficient *structural credit assignment* in such high-dimensional state spaces. In turn, deep Q-networks (DQN) achieve human-level performance in Atari 2600 games by learning from pixel images (Mnih et al., 2015). The key idea is exploiting the underlying structure in such spaces to enable fast generalisation from limited data. In the case of DQN this is made possible through the use of convolutional networks over a factorised representation of the state. Interestingly, the success of DQN is not due to using a sophisticated exploration strategy to efficiently discover useful experiences; on the contrary, DQN uses the simplest exploration strategy there is, ε-greedy. The success of DQN in such high-dimensional state spaces is due to combining the standard Q-Learning algorithm for *temporal* credit assignment with deep neural networks for *structural* credit assignment.

We note that while models such as that of DQN are able to cope with combinatorial (discrete or continuous) state spaces by using function approximators that feature useful *structural inductive biases*, their representation of the action space is always simply *tabular* over the composite space of actions. In this chapter[1] we take a first step towards leveraging the combinatorial structure of multi-dimensional discrete action spaces in order to enable fast generalisation from limited data. Our approach relies on learning a separate estimator of the action-value function for each sub-action space via *independent learning* (Tan, 1993). This approach can significantly improve structural credit assignment by enabling further updates for insufficiently-explored, or even unexplored, actions: updating one action's value would update the values of all actions with a shared subset of sub-actions.

In addition to its primary benefit of improving structural credit assignment, our approach addresses the main computational issues (with respect to the space and time complexity) that arise due to the curse of dimensionality in multi-dimensional discrete action spaces. Firstly, because we do not represent each composite action using a unique output (thus a unique set of weights), our approach enables substantially smaller function approximators. Explicitly, using our approach the number of outputs grows linearly in the number of action dimensions, whereas using the conventional approach it grows exponentially. Secondly, the substantially smaller number of parameters proportionately reduces the amount of computations needed for evaluating the action values and updating the parameters. Thirdly, the actions that maximise the

[1]This chapter is based on the following paper: **Tavakoli, A.**, Pardo, F., and Kormushev, P. (2018). Action branching architectures for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4131–4138.

action-value function could now be identified by maximising the estimates for each sub-action space independently. This decentralised maximisation operation has linear time complexity versus exponential time complexity for the standard approach. The benefit of this substantially reduced computational demand is critical for Q-Learning (in which maximisation operation is required for computing the update target as well as for action selection), but it is also important for methods such as Sarsa (Rummery and Niranjan, 1994) which require maximisation of the action-value function for action selection.

We combine our approach with DQN and test the resulting agent in discretised physical control benchmarks, which feature action spaces of diverse dimensionality. Our results show the effectiveness of our approach in multi-dimensional discrete action spaces, especially in problems with a larger number of action dimensions. Remarkably, our approach achieves effective learning in a highly challenging case with nearly 800 billion actions (just as that in our example from the beginning of this section). Interestingly, these improvements are achieved using the simple ε-greedy for exploration.

3.2 Methods

Our approach is to apply *independent learning* (Tan, 1993), as an approach to cooperative multi-agent reinforcement learning, so as to address the aforementioned challenges faced by individual reinforcement learners in environments with multi-dimensional discrete action spaces. However, given the differences in motivation and problem setting, we refer to our approach as *action branching* to distinguish it from conventional independent learning. The difference in motivation is that we are interested in leveraging the combinatorial structure of multi-dimensional action spaces to improve structural credit assignment and computational scalability, while independent learning is a natural paradigm for a team of agents trying to maximise a shared reward signal. The difference in problem setting is that independent learning primarily deals with heterogeneous partial observations across multiple agents, whereas action branching is for single-agent reinforcement learning and as such deals with a single observation at any given time.

3.2.1 Branching Q-Learning

In principle, action branching can be combined with any action-value method, such as Q-Learning or Sarsa. We illustrate this general possibility using Q-Learning as an example. Consider a problem with action-space dimensionality N and M^i sub-actions for the ith action dimension. We need N action-value estimators Q^i with M^i outputs. Note that this differs from standard Q-Learning in which the only action-value estimator Q has $\prod_{i=1}^{N} M^i$ outputs. We modify the Q-Learning update rule (2.10) by replacing the composite actions with sub-actions along a single action dimension and by replacing the estimator Q (over the composite action space) with Q^i (over the ith sub-action space):

$$Q^i(S_t, A_t^i) \leftarrow Q^i(S_t, A_t^i) + \alpha\Big(R_{t+1} + \gamma \max_{a^{i\prime}} Q^i(S_{t+1}, a^{i\prime}) - Q^i(S_t, A_t^i)\Big). \tag{3.1}$$

In each iteration we update all estimators Q^i in parallel, where $i \in \{1, 2, \ldots, N\}$. This is essentially Independent Q-Learning (Tan, 1993) with the only difference being that of the problem setting which we discussed earlier.

For a given state s and composite action $\mathbf{a} \doteq (a^1, a^2, \ldots, a^N)$, we note that all estimators $Q^i(s, a^i)$ attempt to predict the same value. To explicitly incorporate this notion into the updates we can modify the *independent* update target formulation from Equation (3.1)

$$Y^i_{t,\,\text{independent}} \doteq R_{t+1} + \gamma \max_{a^{i\prime}} Q^i(S_{t+1}, a^{i\prime}) \tag{3.2}$$

into

$$Y_{\text{t, mean}} \doteq R_{t+1} + \gamma \frac{1}{N} \sum_{i=1}^{N} \max_{a^{i\prime}} Q^i(S_{t+1}, a^{i\prime})\,, \tag{3.3}$$

which sets the mean update target across all estimators as the update target for each estimator.

3.2.2 Branching Deep Q-Networks

We described how our approach can be combined with Q-Learning in the tabular case. The combination with other tabular action-value methods, such as Sarsa, or their extensions to function approximation, such as DQN, is similarly straightforward. For the purpose of our experiments we combine our approach with DQN, the canonical action-value method for model-free control in high-dimensional state spaces. We call this agent *Branching DQN* (BDQN). To realise our agent's model we should note that while independent learning generally deals with independent models across the agents (without inter-agent communication or parameter sharing), in our approach we can share parameters to any extent that is desired. We choose to use a shared set of parameters $\boldsymbol{\psi}$ to jointly learn an encoder for state and, thereafter, use a unique set of parameters $\boldsymbol{\theta}^i$ for each estimator. Thus, we denote each estimator Q^i with $Q_{\boldsymbol{\psi},\boldsymbol{\theta}^i}$ to clearly reflect such parameterisation. For all practical purposes, this is a multi-head neural network. All such estimators can then be trained in parallel by minimising their respective sequences of squared TD errors

$$\delta_t^{i2} \doteq \left(R_{t+1} + \gamma \max_{a^{i\prime}} Q_{\boldsymbol{\psi},\boldsymbol{\theta}^i}(S_{t+1}, a^{i\prime}) - Q_{\boldsymbol{\psi},\boldsymbol{\theta}^i}(S_t, A_t^i) \right)^2 \tag{3.4}$$

over samples $\{(S_k, \mathbf{A}_k, R_{k+1}, S_{k+1})\}$, where $\mathbf{A}_k = (A_k^1, A_k^2, \ldots, A_k^N)$. We can modify this expression to incorporate mean update targets as before. To stabilise the training process with function approximation we apply the same considerations as in DQN (which we outlined in Section 2.5.4).

Same as in DQN, we use an ε-greedy strategy as the behaviour policy to balance exploration and exploitation. For greedy action selection (with probability $1 - \varepsilon$), we find sub-actions that maximise each $Q_{\boldsymbol{\psi},\boldsymbol{\theta}^i}$ separately and then concatenate them to obtain the composite greedy action. Otherwise (with probability ε), we sample sub-actions uniformly at random across all action dimensions. Similarly to DQN, we linearly decay ε over time.

3.3 Experiments

3.3.1 Experimental Setup

We test our approach in several physical control benchmark environments that feature continuous action spaces of diverse dimensionality, providing us with a range of combinatorial action spaces. Table 3.1 shows the dimensionality and size of the action spaces in these environments, where the latter is obtained by discretising each action space with the granularity of five sub-actions per joint (just as in our experiments). Given that the continuous sub-actions in our environments belong in the range $[-1, 1]$, discretizing them into five equally spaced sub-actions yields discrete sub-action spaces of the form $\{-1, -0.5, 0, 0.5, 1\}$. This allows for learning even finer behaviours than bang-bang and bang-off-bang solutions (i.e. policies over extremal sub-actions, respectively, excluding and including the neutral 0 sub-action), two choices of discrete-policy representation which have been argued to be sufficient for solving numerous continuous control benchmarks (Seyde et al., 2021). The first five environments (Reacher, Hopper, HalfCheetah, Walker2D, and Ant) feature the full suite of PyBullet benchmarks (Coumans and Bai, 2019) excluding those with one-dimensional action spaces (InvertedPendulum and InvertedDoublePendulum) or without a stable implementation (Humanoid). For the missing Humanoid benchmark we use the one from OpenAI Gym (Brockman et al., 2016), which is simulated using MuJoCo (Todorov et al., 2012). These environments use predefined time limits: 1000 steps for all locomotion tasks and 150 steps for the single manipulation task (Reacher). See Appendix A.2 for a brief overview of these environments.

We base our implementation of BDQN on the Dopamine framework (Castro et al., 2018), which provides a reliable open-source code for DQN. We simplify the network architecture of DQN (originally for Atari 2600 games) by replacing the convolutional networks with two hidden layers of 600 and 400 rectifier units to reflect the non-pixel nature of states in our physical control benchmarks. We share these layers across all network heads. We also adapt the final hidden layer from 512 to 400 rectifier units, dividing them equally across the number of network heads for BDQN. We remark that this simple heuristic does not achieve similar numbers of parameters with respect to our discrete action baselines in environments with large action spaces. Nonetheless, this is inevitable given that the standard models require a set of unique parameters for representing each possible action. Table 3.2 lists the hyperparameters used for BDQN and DQN in our physical control benchmarks.

All reported learning curves are generated by evaluating the agents after every 10,000 steps during the training using an ε-greedy policy with $\varepsilon = 0.001$, each time for a minimum of 5000 steps (until the last evaluation episode ends by reaching a terminal state or the time limit).

Domain	**Dimensionality**	**Size**
Reacher	2	25
Hopper	3	125
HalfCheetah	6	15,625
Walker2D	6	15,625
Ant	8	390,625
Humanoid	17	762,939,453,125

Table 3.1: Dimensionality and size (using five sub-actions per joint) of the action spaces in our physical control benchmarks.

Hyperparameter	**Value**
minibatch size	64
replay memory size	100,000
agent history length	1
target network update frequency	2000
discount factor	0.99
action repeat	1
update frequency	1
optimiser	Adam
learning rate	0.00001
$\hat{\epsilon}$ (a constant used for numerical stability in Adam)	0.0003125
β_1 (1^{st} moment decay rate used by Adam)	0.9
β_2 (2^{nd} moment decay rate used by Adam)	0.999
loss function	mean-squared error
initial exploration	1
final exploration	0.05
final exploration step	50,000
replay start size	10,000

Table 3.2: Hyperparameters used for BDQN and DQN in our physical control benchmarks.

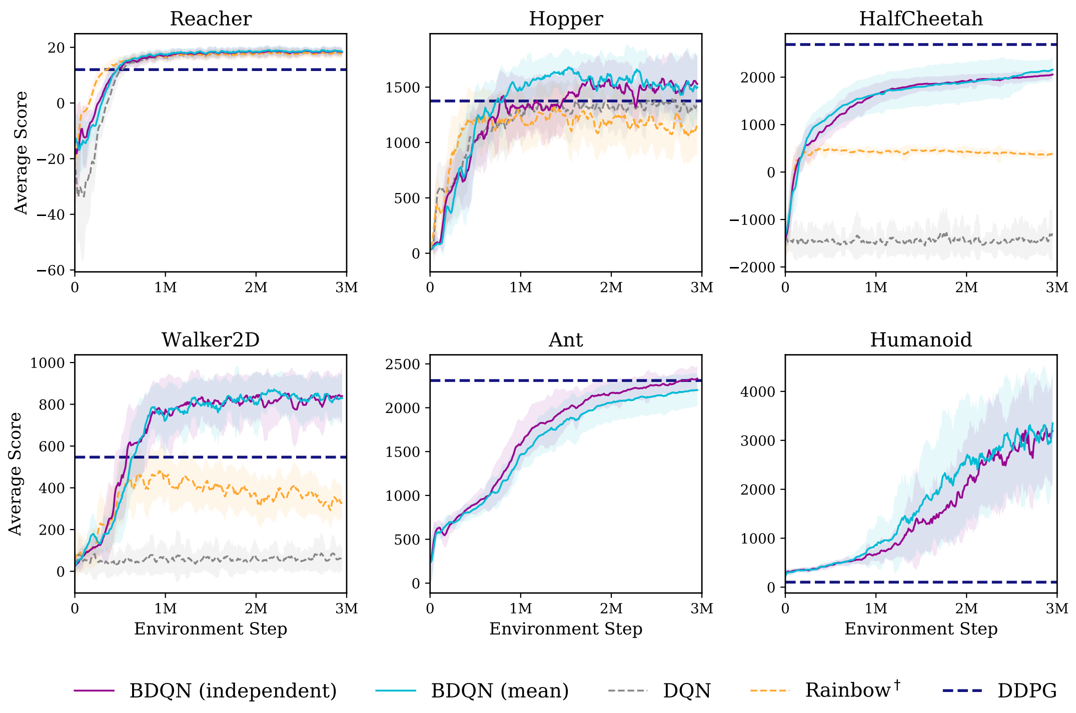


Figure 3.1: Learning curves for BDQN (with independent and mean update targets), DQN, and a simplified version of Rainbow in our physical control benchmarks (nine random seeds). Shaded regions indicate standard deviation. Average performance of DDPG trained for three million environment steps is provided for illustration purposes.

3.3.2 Results

Figure 3.1 shows the learning curves for BDQN with independent and mean update targets as well as DQN. In Reacher with a two-dimensional action space we do not see any significant difference. In Hopper with a three-dimensional action space we start to observe moderate improvements by both BDQN variants over DQN. Remarkably, in HalfCheetah and Walker2D with six-dimensional action spaces the performance of DQN falls flat, whereas both BDQN variants manage to achieve high levels of performance. In Ant and Humanoid with higher-dimensional action spaces we were unable to run DQN as it imposes significant computational demands which render it intractable. On the other hand, BDQN successfully scales to such high-dimensional action spaces without a significant increase in the required computational resources (in particular, memory and time requirements). This demonstrates the utility of leveraging the combinatorial structure of multi-dimensional discrete action spaces to enable fast generalisation from limited data as well as computational scalability.

We also provide the performance of DQN combined with prioritised replay

(Schaul et al., 2016), dueling networks (Wang et al., 2016), multi-step learning (Hessel et al., 2018), and Double Q-Learning (van Hasselt et al., 2016) (denoted Rainbow†; see Hessel et al. (2018) for an overview). The significant gap between the performances of this agent and vanilla BDQN supports the orthogonality of our approach with respect to these extensions.

Moreover, we include the average final performance of DDPG (Lillicrap et al., 2016) to give a sense of the performances achievable by a continuous control method, particularly by one that is closely related to DQN. The reported DDPG results are based on the implementation by Achiam (2018). The final performances (at three million environment steps) were obtained by averaging test performances (with no action noise) over the last 100,000 steps of training (20 random seeds). The network architecture and hyperparameters match those reported in Lillicrap et al. (2016). Interestingly, the performance of BDQN across our environments is either on par with or significantly better than that of DDPG. This is despite the fact that, unlike DDPG, BDQN does not use a specialised exploration strategy to exploit ordinality of the underlying continuous action space or perform temporally-extended exploration which is useful in physical environments with inertia. Remarkably, in Humanoid with a 17-dimensional action space both BDQN variants achieve a high performance level, whereas DDPG falls flat. This may be due to local optimisation issues in DDPG which can lead to sub-optimal policies in multi-modal value landscapes (Metz et al., 2017; Tessler et al., 2019).

Lastly, we do not observe a significant difference between the performances of our BDQN variants, with only a moderately better overall performance for the variant with mean update targets. Nevertheless, evaluations on a broader set of environments are necessary to establish whether using mean update targets benefits learning with action branching.

3.4 Related Work

To the best of our knowledge, the idea of modelling the problem of single-agent reinforcement learning in multi-dimensional discrete action spaces as a cooperative multi-agent one was first mentioned by Buşoniu et al. (2006), where decentralised value iteration was used to learn a control policy in a problem with a two-dimensional action space. Nonetheless, the reported improvements were mostly in terms of reduced computational requirements and not sample complexity. Similar observations were made by Troost et al. (2008) but this time in a model-free reinforcement learning context, based on Q(λ)-Learning and Sarsa(λ). A more extensive examination in this direction was performed, concurrently to our work, by Leottau et al. (2018) which showed that improved sample complexity can be obtained in environments with higher-dimensional action spaces via decentralised reinforcement learning (using a family of such methods including independent learning) as compared to the centralised counterparts while requiring less computational resources. Nonetheless, this work also remains limited to environments with relatively low-dimensional action spaces (with the highest being four-dimensional). The work presented in this chapter serves as the first empirical validation for scaling single-agent reinforcement learning to such high-dimensional discrete action spaces using decentralised reinforcement learning, as well as the first large-scale examination of such an approach in the context of deep reinforcement learning.

Our results could also serve to further corroborate the effectiveness of Independent DQN (Tampuu et al., 2017) in cooperative multi-agent reinforcement learning problems by providing empirical evidence for its success at scale (with up to 17 independent learners) in a related setting.

Some alternative approaches for Q-Learning in environments with high-dimensional discrete action spaces have been explored. Concurrent to the work presented in this chapter, Metz et al. (2017) proposed factoring the action space and predicting action values sequentially across the sub-action spaces using an autoregressive model architecture. More recently, Van de Wiele et al. (2020) proposed a method based on combining Q-Learning with a sampling-based approach for maximising the action values. An approximate maximisation as such enables Q-Learning to computationally scale to environments with high-dimensional discrete or continuous action spaces. These methods have been shown to enable Q-Learning to perform competitively against DDPG or D3PG (a distributed version of DDPG), respectively, in numerous discretised physical control benchmarks.

Several methods have been proposed for learning in large discrete action spaces that have an associated underlying continuous action representation by using a continuous control method (van Hasselt and Wiering, 2009; Dulac-

Arnold et al., 2015). Nevertheless, these methods are not intended for enabling the application of discrete action methods such as Q-Learning to environments with high-dimensional discrete action spaces. For example, the approach of Dulac-Arnold et al. (2015) relies on DDPG.

3.5 Conclusion

In this chapter we demonstrated that fast generalisation from limited data can be achieved in environments with multi-dimensional discrete action spaces by assigning credit to the action's sub-actions as opposed to solely the action for which an update is performed. The improvements, which are gained without modifying the methods of exploration or temporal credit assignment, are due to better structural credit assignment. Our results illustrate that exploiting the combinatorial structure of multi-dimensional action spaces is a key ingredient in enabling sample-efficient structural credit assignment and, thus, fast generalisation. We also demonstrated that representing action values over a factorised action space rather than its composite form has significant computational benefits due to achieving substantially smaller tables or number of parameters. Moreover, in the case of Q-Learning, action-value maximisation can be done over the factorised action space as opposed to the composite one, thus achieving linear time complexity versus exponential time complexity for the conventional approach. (This also applies to other methods that require maximisation of action values for action selection such as Sarsa.) In the high-dimensional Humanoid environment BDQN achieved great learning performance in only three million environment steps despite the underlying action space having nearly 800 billion actions.

To enable learning action values over the factorised action space, we resorted to independent learning, or action branching as we call it in our case. While in practice this approach shows great success, in theory it is subject to numerous challenging *coordination* issues. These coordination issues are well known and extensively studied in the context of cooperative multi-agent reinforcement learning (Claus and Boutilier, 1998; Panait et al., 2008). The literature identifies five challenges responsible for the non-coordination of independent learners: Pareto-selection, nonstationarity, stochasticity, alter-exploration, and shadowed equilibria (see Matignon et al. (2012) for a detailed overview). The BDQN agent does not explicitly address any of these coordination issues. However, a certain choice which comes naturally in a single-agent reinforcement learning context could be responsible for the effectiveness of BDQN. Specifically, as described in Section 3.2.2, our ε-greedy policy is such that the independent learners explore or exploit in unison. This is conjectured in the literature as a way of overcoming certain coordination issues (Troost et al., 2008; Matignon et al., 2012). In addition, the relative success of independent learning in cooperative multi-agent reinforcement learning problems is often attributed to using a GLIE[2] exploration strategy (Matignon et al., 2012). Similarly to the standard DQN agent, we linearly annealed ε during training from 1.0 to 0.05 over the first 50,000 environment steps, and fixed it at 0.05 thereafter. Notice that while this is not exactly a GLIE strategy, it achieves reduced exploration frequency over time to a good degree.

Given the remarkable effectiveness of BDQN in multi and high-dimensional action spaces, an interesting direction for future work is to apply explicit mechanisms to resolve the coordination issues. In the next chapter we take such a step forward, providing an alternative approach with a more general formulation and more appealing properties.

[2]GLIE: Greedy in the Limit with Infinite Exploration (Singh et al., 2000).

Chapter 4

Action Hypergraph Networks

4.1 Introduction

In the previous chapter we demonstrated that factorising multi-dimensional discrete action spaces and learning action values over the sub-action spaces in an independent manner unleashes significant benefits with regard to sample, time, and space complexity. The key was in enabling a capacity to leverage the combinatorial structure of such action spaces. We only investigated leveraging the combinatorial structure at the level of the sub-action spaces. Nonetheless, we argue that enabling higher-order combinations of the sub-action spaces to also contribute to the estimation of action values can further facilitate generalisation. Moreover, the inclusion of higher-order combinations increases the representational capacity of the model or, in other words, expands the space of possible action-value functions that can be accurately represented. Increasing the representational capacity in this sense allows us to reduce the coordination issues, or even bypass them altogether in moderate action spaces. Note that this is not readily possible with independent learners, whereby the control of each sub-action space is delegated to a separate estimator. In order to enable such a capacity, and more generally improve coordination, in this chapter[1] we resort to an alternative approach, formulating the problem as a representation learning one.

Representation learning methods have helped shape the recent progress in reinforcement learning by enabling a capacity for learning good representations of state. This is in spite of the fact that representation learning was tradition-

[1]This chapter is based on the following paper: **Tavakoli, A.**, Fatemi, M., and Kormushev, P. (2021). Learning to represent action values as a hypergraph on the action vertices. In *Proceedings of the International Conference on Learning Representations.*

ally less often explored in the reinforcement learning context. As such, the de facto representation learning techniques which are widely used in reinforcement learning were developed under other machine learning paradigms (Bengio et al., 2013). Nevertheless, reinforcement learning brings some unique problems to the topic of representation learning, with exciting headway being made in identifying and exploring them.

Action-value estimation is a critical component of the reinforcement learning paradigm. Hence, how to effectively learn estimators for action value from training samples is one of the major problems studied in reinforcement learning. We set out to study this problem through the lens of representation learning, focusing particularly on learning representations of action in multi-dimensional discrete action spaces. While action values are conditioned on both state and action and, as such, good representations of both would be beneficial, there has been comparatively little research on learning action representations.

We frame this problem as learning a decomposition of the action-value function that is structured in such a way to leverage the combinatorial structure of multi-dimensional discrete actions. This structure is an inductive bias which we incorporate in the form of architectural assumptions. We present this approach as a framework to flexibly build architectures for learning representations of multi-dimensional discrete actions by leveraging various orders of their underlying sub-action combinations. These architectures can be combined in succession with any architecture for learning state representations and trained end-to-end using backpropagation, without imposing any change to the reinforcement learning algorithm. We remark that designing representation learning methods by incorporating some form of structural inductive biases is highly common in deep learning, resulting in highly-publicised architectures such as recurrent, convolutional, and graph networks (see Battaglia et al. (2018) for a detailed discussion of these inductive biases). Moreover, even without the inclusion of higher-order combinations, framing the problem as learning a decomposition of the action-value function generally enables better coordination than that of independent learners; this is because it allows each estimator to contribute the utility of its own sub-actions to the joint estimate. This is based on a finding by Sunehag et al. (2017) in the context of cooperative multi-agent reinforcement learning.

We first demonstrate the effectiveness of our approach in illustrative, structured prediction problems with only one state (as in multi-armed bandits). We then argue for the ubiquity of similar structures and test our approach in standard reinforcement learning problems, including our physical control benchmarks from the previous chapter. The results advocate for the general usefulness of leveraging the combinatorial structure of multi-dimensional discrete action spaces at various orders.

4.2 Definition of Hypergraph

A *hypergraph* (Berge, 1989) is a generalisation of a graph (West, 1996) in which an edge, also known as a *hyperedge*, can join any number of vertices. Let $V \doteq \{\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_{N^v}\}$ be a finite set representing the set of vertices $\mathcal{A}_i$. A hypergraph on V is a family of subsets or hyperedges $H \doteq \{E_1, E_2, \dots, E_{N^e}\}$ such that

$$E_j \neq \emptyset \quad (j = 1, 2, \dots, N^e), \tag{4.1}$$

$$\bigcup_{j=1}^{N^e} E_j = V. \tag{4.2}$$

According to Equation (4.1), each hyperedge E_j is a member of $\mathcal{E} \doteq \mathcal{P}(V) \setminus \emptyset$ where $\mathcal{P}(V)$, called the *powerset* of V, is the set of possible subsets on V. The rank r of a hypergraph is defined as the maximum cardinality of any of its hyperedges. We define a *c-hyperedge*, where $c \in \{1, 2, \dots, N^v\}$, as a hyperedge with cardinality or order c. The number of possible c-hyperedges on V is given by the binomial coefficient $\binom{N^v}{c}$. We define a *c-uniform* hypergraph as one with only c-hyperedges. As a special case, a *c-complete* hypergraph, denoted K^c, is one with all possible c-hyperedges.

In this chapter we treat each sub-action space $\mathcal{A}_i$ as a vertex, or an *action vertex*. In this way, V denotes the set of action vertices, N^v the number of action dimensions, and N^e the number of hyperedges in a hypergraph on the action vertices.

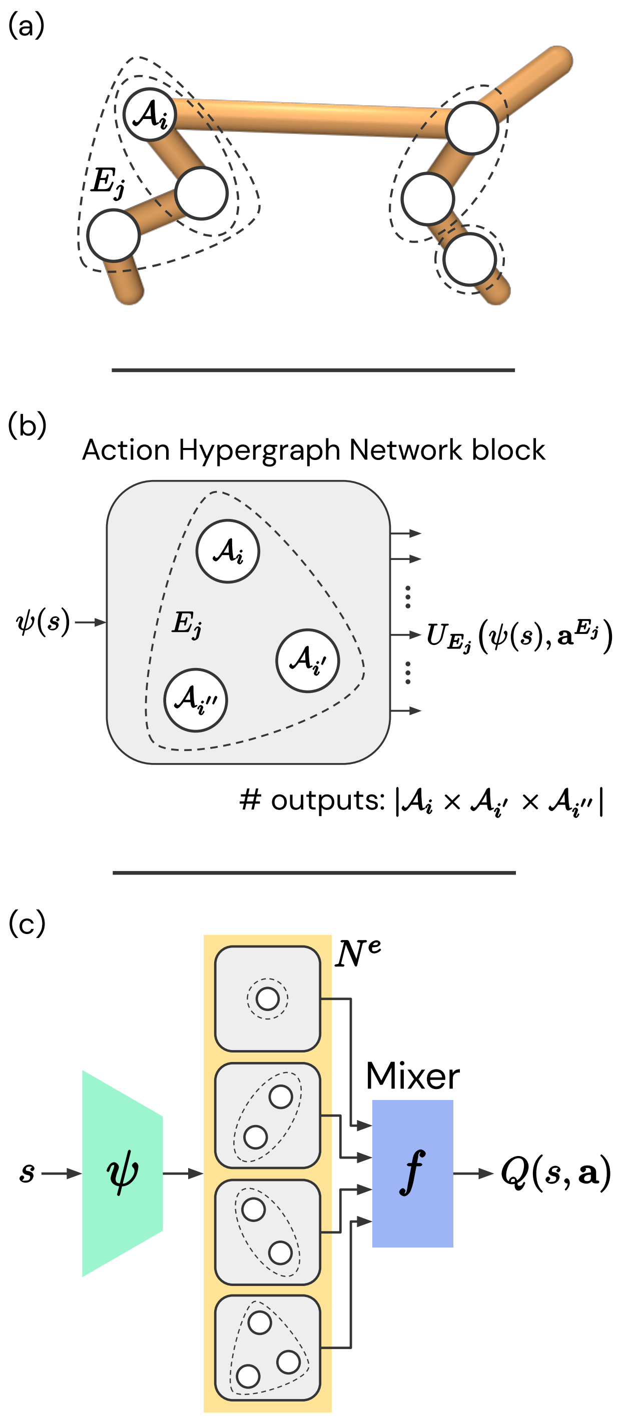

(a)
$\mathcal{A}_i$
E_j
(b)
Action Hypergraph Network block
$\psi(s)$
$\mathcal{A}_i$
E_j
$\mathcal{A}_{i'}$
$\mathcal{A}_{i''}$
$U_{E_j}(\psi(s), \mathbf{a}^{E_j})$
# outputs: $|\mathcal{A}_i \times \mathcal{A}_{i'} \times \mathcal{A}_{i''}|$
(c)
N^e
Mixer
s
ψ
f
$Q(s, \mathbf{a})$

Figure 4.1: (a) A sample hypergraph overlaid on a physical system with six action vertices. (b) An instance building block of our framework for a sample hyperedge. (c) An architecture is realised by stacking several building blocks, one for each hyperedge in the hypergraph.

4.3 Action Hypergraph Networks Framework

We now describe our framework using the example in Figure 4.1. Consider the sample physical system of Figure 4.1a with six action vertices (solid circles). A sample hypergraph is depicted for this system with four hyperedges (dashed shapes), featuring a 1-hyperedge, two 2-hyperedges, and a 3-hyperedge. Note that this set of hyperedges constitutes a hypergraph as there are no empty hyperedges and the union of hyperedges spans the set of action vertices, respectively satisfying conditions (4.1) and (4.2).

We wish to enable learning a representation of each hyperedge in an arbitrary hypergraph. To achieve this, we create a parameterised function U_{E_j} (e.g. a neural network) for each hyperedge E_j which receives a state representation $\boldsymbol{\psi}(s)$ as input and returns as many values as the possible combinations of the sub-actions for the action vertices enclosed by its respective hyperedge. In other words, U_{E_j} has as many outputs as the cardinality of a Cartesian product of the action vertices in E_j. Each such hyperedge-specific function U_{E_j} is a building block of our *action hypergraph networks* framework.[2] Figure 4.1b depicts the block corresponding to the 3-hyperedge from the hypergraph of Figure 4.1a. We remark that for any action $\mathbf{a} \doteq (a^1, a^2, \dots, a^{N^v})$ from the action space $\mathcal{A}$, each block has only one output that corresponds to $\mathbf{a}$ and, thus, contributes exactly one value as a representation of its respective hyperedge at the given action. This output $U_{E_j}\big(\boldsymbol{\psi}(s), \mathbf{a}^{E_j}\big)$ is identified by $\mathbf{a}^{E_j}$ which denotes the combination of sub-actions in $\mathbf{a}$ that correspond to the action vertices enclosed by E_j.

We can realise an architecture within our framework by composing several of such building blocks, one for each hyperedge in a hypergraph of our choosing. Figure 4.1c shows an instance architecture corresponding to the sample hypergraph of Figure 4.1a. The forward view through this architecture is as follows. A shared representation of the input state s is fed into multiple blocks, each of which features a unique hyperedge. A representation vector of size N^e is then obtained for each action $\mathbf{a}$, where N^e is the number of hyperedges or blocks. These action-specific representations are then mixed (on an action-by-action basis) using a function f (e.g. a fixed non-parametric function or a

[2]Throughout this chapter we assume linear units for the block outputs. However, other activation functions could be more useful depending on the task or choice of mixing function.

neural network). The output of this mixing function is our estimator for action value at the state-action pair $(s, \mathbf{a})$. Concretely,

$$Q(s, \mathbf{a}) \doteq f\Big(U_{E_1}\big(\boldsymbol{\psi}(s), \mathbf{a}^{E_1}\big), U_{E_2}\big(\boldsymbol{\psi}(s), \mathbf{a}^{E_2}\big), \ldots, U_{E_{N^e}}\big(\boldsymbol{\psi}(s), \mathbf{a}^{E_{N^e}}\big) \Big). \quad (4.3)$$

While only a single output from each block is relevant for an action, the reverse is not the case. That is, an output from a block generally contributes to more than a single action's value estimate. In fact, the lower the cardinality of a hyperedge, the larger the number of actions' value estimates to which a block output contributes. This can be thought of as a form of combinatorial generalisation. Particularly, action value can be estimated for an insufficiently-explored or unexplored action by mixing the action's corresponding representations which have been trained as parts of other actions. Moreover, this structure enables a capacity for learning faster on lower-order hyperedges by receiving more updates and learning slower on higher-order ones by receiving less updates. This is a desirable intrinsic property as we would ideally wish to learn representations that exhaust their capacity for representing action values using lower-order hyperedges before they resort to higher-order ones.

4.3.1 Mixing Function Specification

The mixing function receives as input a state-conditioned representation vector for each action. We view these action-specific representation vectors as *action representations*. Explicitly, these action representations are learned as a decomposition of the action-value function under the mixing function. Without a priori knowledge about its appropriate form, the mixing function should be learned by a universal function approximator. However, joint learning of the mixing function together with a good decomposition under its dynamic form could be challenging. Moreover, increasing the number of hyperedges expands the space of possible decompositions, thereby making it even harder to reach a good one. The latter is due to the lack of identifiability in value decomposition in that there is not a unique decomposition to reach. Nevertheless, this issue is not unique to our setting as representations learned using neural networks are in general unidentifiable. We demonstrate the potential benefit of using a universal mixer (a neural network) in our illustrative bandit problems. However, to allow flexible experimentation without re-tuning standard agent implementations, in our reinforcement learning benchmarks we choose to use summation as a non-parametric mixing function. This boils down the task of learning action representations to reaching a good linear decomposition of the action-value function under the summation mixer. We remark that the summation mixer is commonly used with value-decomposition methods in the context of cooperative multi-agent reinforcement learning (Sunehag et al., 2017). In an informal evaluation in our reinforcement learning benchmarks, we did not find any advantage for a universal mixer over the summation one. Nonetheless, this could be a matter of tuning the learning hyperparameters.

4.3.2 Hypergraph Specification

We now consider the question of how to specify a good hypergraph. In actuality, there is not an all-encompassing answer to this question. For example, the choice of hypergraph could be treated as a way to incorporate a priori knowledge about a specific problem. Nonetheless, we can outline some general rules of thumb. In principle, including as many hyperedges as possible enables a richer capacity for discovering useful structures. Correspondingly, an ideal representation is one that returns neutral values (e.g. near-zero inputs to the summation mixer) for any hyperedge whose contribution is not necessary for accurate estimation of action values (i.e. the lower-order hyperedges are able to represent its contribution). However, as described in Section 4.3.1, having many mixing terms could complicate reaching a good decomposition due to the lack of identifiability. Taking these into consideration, we frame hypergraph specification as choosing a rank r whereby we specify the hypergraph that comprises all possible hyperedges of orders up to and including r:

$$H \doteq \bigcup_{c=1}^{r \leq N^v} K^c, \tag{4.4}$$

where hypergraph H is expressed as the union of c-complete hypergraphs K^c. Figure 4.2 depicts a standard model against a class of models in our framework where the space of possible hyperedges $\mathcal{E}$ is ordered by cardinality.

In general, not including the highest-order (N^v) hyperedge limits the representational capacity of an action-value estimator. This could introduce statistical bias due to estimating N actions' values using a model with $M < N$

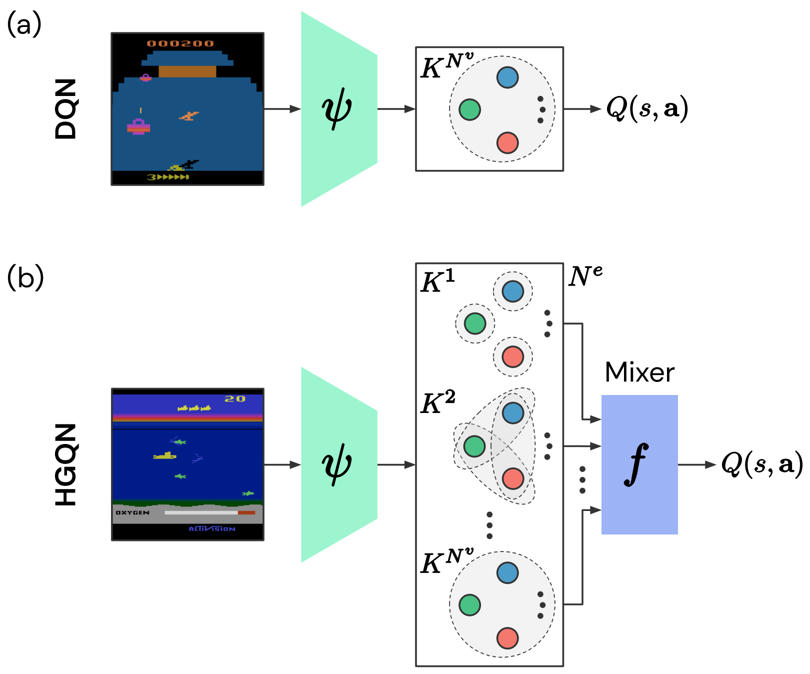


Figure 4.2: (a) A standard model. (b) A class of models in our framework, depicted by the ordered space of possible hyperedges $\mathcal{E}$. Our class of models subsumes the standard one as an instance.

unique outputs.[3] In a structured problem $M < N$ could suffice, otherwise such bias is inevitable. Consequently, choosing any $r < N^v$ could affect the estimation accuracy in a prediction problem and cause sub-optimality in a control problem. Thus, we preferably wish to use hypergraphs of rank $r = N^v$. We remark that this bias is additional to (and as such should be distinguished from) the bias of function approximation which also affects methods such as DQN. We can view this in terms of the bias-variance tradeoff with r acting as a knob: lower r means more bias but less variance, and vice versa. Notably, when the N^v-hyperedge is present even the simple summation mixer can be used without causing bias. However, when this is not the case, the choice of mixing function could significantly influence the extent of bias. In this work we generally fix the choice of the mixing function to summation and instead try to include higher-order hyperedges.

[3]The notions of *statistical bias* and *inductive bias* should be distinguished from one another: an inductive bias does not necessarily imply a statistical bias. In fact, a good inductive bias is one that enables a better generalisation capacity but causes little or no statistical bias. In this chapter we use "bias" as a convenient shorthand for statistical bias.

4.4 Experiments

4.4.1 Illustrative Prediction Problems

In this section we set out to illustrate the essence of problems in which we expect improvements by our approach. To do so, we minimise confounding effects in both our problem setting and learning method. Given that we are interested purely in studying the role of learning representations of *action* (and not of state), we consider a multi-armed bandit (one-state MDP) problem setting. We specify our bandits such that they have a combinatorial action space of three dimensions, where we vary the action-space sizes by choosing the number of sub-actions per action dimension from $\{5, 10, 20\}$.

The reward functions are deterministic but differ for each random seed. Despite using a different reward function in each independent trial, they are all generated to feature a good degree of decomposability with respect to the combinatorial structure of the action space. That is to say, by design, there is generally at least one possible decomposition that has nonzero values on all possible hyperedges. To achieve this we create each reward function by randomly initialising a hypergraph model with all possible hyperedges. Explicitly, in each trial, we sample as many values as the total number of outputs across all possible hyperedges: sampling uniformly from $[-10, 10]$ for the 1-hyperedges, $[-5, 5]$ for the 2-hyperedges, and $[-2.5, 2.5]$ for the 3-hyperedges. This results in structured reward functions that can be decomposed to a good degree on the lower-order hyperedges but still need the highest-order hyperedge for a precise decomposition. Next, we generate a random mixing function by uniformly sampling the parameters of a single-hidden-layer neural network from $[-1, 1]$. The number of hidden units and the activation functions are sampled uniformly from $\{1, 2, \ldots, 5\}$ and $\{\text{ReLU, tanh, sigmoid, linear}\}$, respectively. Lastly, a deterministic reward for each action is generated by mixing the action's corresponding subset of values.

We train predictors for reward (equivalently, the optimal action values) using minimalist parameterised models that resemble tabular ones as closely as possible. As such, our baseline corresponds to a standard tabular model in which each action value is estimated by a single unique parameter. In contrast, our approach does not always require a unique parameter for each action as it relies on mixing multiple action-representation values to produce an action-value estimate. Hence, for our approach we instantiate as many unique parameters as the total number of outputs from each model's hyperedges. Moreover, we consider summation and universal mixers as well as increasingly more complete hypergraphs, specified using Equation (4.4) by varying rank r from 1 to 3. For any model using a universal mixer we additionally use a single hidden layer of 10 rectifier units for mixing the action-representation values, initialised using the Xavier uniform method (Glorot and Bengio, 2010). Each predictor is trained in a supervised learning manner, using backpropagation and stochastic gradient descent to minimise the mean-squared prediction error. We repeatedly sample minibatches of 32 rewards (with replacement) to update a predictor's parameters, with each training iteration comprising 100 such updates.

The number of hyperedges in our hypergraphs of interest (according to Section 4.3.2) can be expressed in terms of binomial coefficients as

$$|H| \doteq \sum_{c=1}^{r \leq N^v} \binom{N^v}{c},$$

where hypergraph H is specified by its rank r according to Equation (4.4). As we described in Section 4.3, each action's value estimator is formed by mixing as many action-representation values as there are hyperedges in the model (see Equation (4.3)). In our simple models for the bandit problems, each such value corresponds to a single parameter. Therefore, each learning update per action involves updating as many parameters as the number of hyperedges. To ensure fair comparisons, we adapt the learning rates across different models in our study based on the number of parameters involved in updating each action's value estimate. Concretely, we set a single effective learning rate across all models in our study. We then obtain each model's individual learning rate α via

$$\alpha \doteq \frac{\text{effective learning rate}}{|H|}.$$

We use an effective learning rate of 0.0007 in our study. While this achieves the same actual effective learning rate for both the baseline and our models using the summation mixer, the same does not hold exactly for our models using a universal mixer. Nonetheless, this still serves as a useful heuristic to

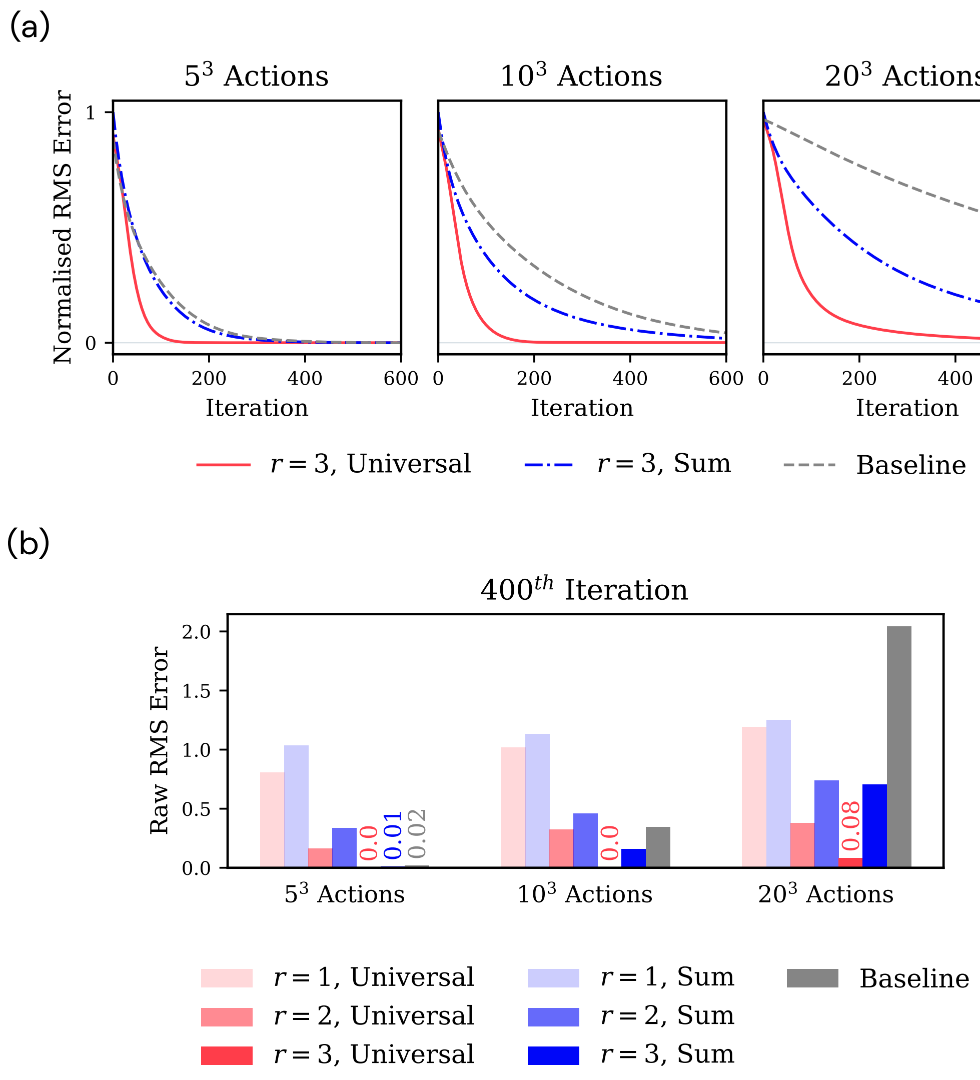


Figure 4.3: Prediction error in our illustrative multi-armed bandits with three action dimensions and increasing action-space sizes. Each variant is run on 64 reward functions in any action-space size. (a) Normalised average RMS error curves. (b) Average RMS errors at the 400th training iteration.

obtain similar effective learning rates across all models. It is important to note that the baseline receives the largest individual learning rate across all other models, one that improves its performance with respect to any other learning rates used by the variants of our approach.[4]

Figure 4.3a shows normalised RMS (root mean square) prediction error curves (averaged over 64 reward functions) for two variants of our approach that leverage all possible hyperedges versus our tabular baseline. The learning

[4]The best learning rate for standard tabular learning in deterministic environments is $\alpha = 1$. Nevertheless, the variants of our approach undergo more complex learning dynamics (due to learning value decompositions) and thus require a lower learning rate. Our systematic adjustment of the learning rate allows us to fairly study sample complexity under a similar effective learning rate for all.

curve for each independent trial is generated by calculating the RMS prediction error across all possible actions after each training iteration. We see that the ordering of the curves remains consistent with the increasing number of actions, where the baseline is outperformed by our approach regardless of the mixing function. Nevertheless, our model with a universal mixer performs significantly better in terms of sample efficiency. Moreover, the performance gap becomes significantly wider as the action-space size increases, attesting to the utility of leveraging the combinatorial structure of actions for scaling to more complex action spaces. Figure 4.3b shows average RMS prediction errors at the 400th training iteration (as a proxy for "final" performance) for all variants in our study. As expected, including higher-order hyperedges and/or using a more generic mixing function improves final prediction accuracy in every case.

Going beyond these simple prediction problems to our control benchmarks in reinforcement learning we anticipate a general advantage for our approach, following the same pattern of yielding more significant improvements in larger action spaces.

4.4.2 Atari 2600 Games

We now test our approach in the Atari 2600 games of the Arcade Learning Environment (ALE) (Bellemare et al., 2013) (see Appendix A.1 for a brief overview). The action space of Atari 2600 is determined by a digital joystick with three degrees of freedom: three positions for each axis of the joystick, plus a button. This implies that these games can have a maximum of 18 discrete actions, with many not making full use of the joystick capacity. To focus our compute resources on games with a more complex action space we limit our tests to 29 Atari 2600 games from the literature that feature 18 valid actions.

For the purpose of our control experiments we combine our approach with deep Q-networks (DQN) (Mnih et al., 2015). The resulting agent, which we dub *hypergraph Q-networks* (HGQN), deploys an architecture based on our action hypergraph networks (similar to that shown in Figure 4.2b) using the summation mixer. Given that the action space has merely three dimensions, we instantiate our agent's model based on a hypergraph including the seven possible hyperedges. We realise this model by modifying the DQN's final hidden layer into a multi-head one, where each head implements a block from our framework for a respective hyperedge. To achieve a fair comparison with DQN we ensure that our agent's model has roughly the same number of parameters as DQN by making the sum of the hidden units across its seven heads to match that of the final hidden layer in DQN. Specifically, we implement HGQN by replacing the final hidden layer of 512 rectifier units in DQN with seven network heads, each with a single hidden layer of 74 rectifier units. Our agent is trained end-to-end using backpropagation in the same way as DQN.

Just as in Chapter 3, we base our implementation of HGQN on the Dopamine framework (Castro et al., 2018). Dopamine provides a reliable open-source code for DQN as well as enables standardised benchmarking in the ALE under the best known evaluation practices (see, e.g., Bellemare et al. (2013) and Machado et al. (2018)). As such, we conduct our Atari 2600 experiments without any modifications to the agent or environment parameters with respect to those outlined in Castro et al. (2018), except for the network architecture change for HGQN described above. Our DQN results are based on the published Dopamine baselines.

The human-normalised scores reported in this section are given by the formula (similarly to van Hasselt et al. (2016) and Dabney et al. (2018))

$$\frac{\text{score}_{\text{agent}} - \text{score}_{\text{random}}}{\text{score}_{\text{human}} - \text{score}_{\text{random}}},$$

where $\text{score}_{\text{agent}}$, $\text{score}_{\text{human}}$, and $\text{score}_{\text{random}}$ are the per-game scores (undiscounted returns) for the given agent, a reference human player, and random agent baseline. We use Table 2 from Wang et al. (2016) to retrieve the human player and random agent scores. The relative human-normalised score of HGQN versus DQN in each game is given by the formula (similarly to Wang et al. (2016))

$$\frac{\text{score}_{\text{HGQN}} - \text{score}_{\text{DQN}}}{\max(\text{score}_{\text{DQN}}, \text{score}_{\text{human}}) - \text{score}_{\text{random}}},$$

where $\text{score}_{\text{HGQN}}$ and $\text{score}_{\text{DQN}}$ are computed by averaging over their respective learning curves.

Figure 4.4a shows the relative human-normalised score of HGQN (three random seeds) versus DQN (five random seeds) for each game. Figure 4.4b shows median and mean human-normalised scores across the 29 Atari games for HGQN and DQN. Our results indicate improvements over DQN on the majority of the games (Figure 4.4a) as well as in overall performance (Figure 4.4b).

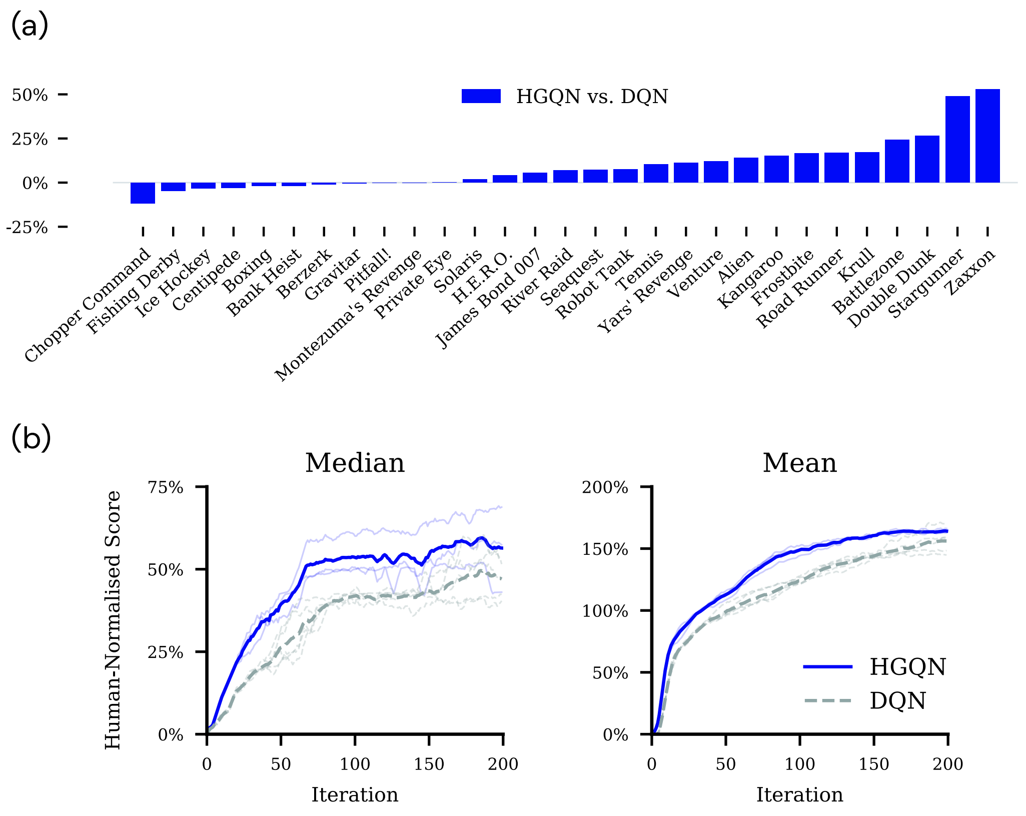


Figure 4.4: (a) Difference in human-normalised score for 29 Atari 2600 games with 18 valid actions, HGQN versus DQN over 200 training iterations (positive % means HGQN outperforms DQN). (b) Human-normalised median and mean scores across the same set of games. Random seeds are shown as traces.

Notably, these improvements are both in terms of sample complexity and final performance (Figure 4.4b). The consistency of improvements in these games has the promise of greater improvements in tasks with larger action spaces. See Figure B.1 for full learning curves across these 29 Atari 2600 games.

To further demonstrate the versatility of our approach, we combine it with a simplified version of Rainbow (Hessel et al., 2018) that includes prioritised replay (Schaul et al., 2016), dueling networks (Wang et al., 2016), multi-step learning (Hessel et al., 2018), and Double Q-Learning (van Hasselt et al., 2016). We do not include C51 (Bellemare et al., 2017) as it is not trivial how it can be combined effectively with our approach. We also do not combine with noisy networks (Fortunato et al., 2018) as they are not implemented in Dopamine. We denote the simplified Rainbow by Rainbow† and the version combined with our approach by HG-Rainbow†.

We run these agents on the three best and worst-performing games from Figure 4.4a. We conjecture that the games in which HGQN outperforms DQN

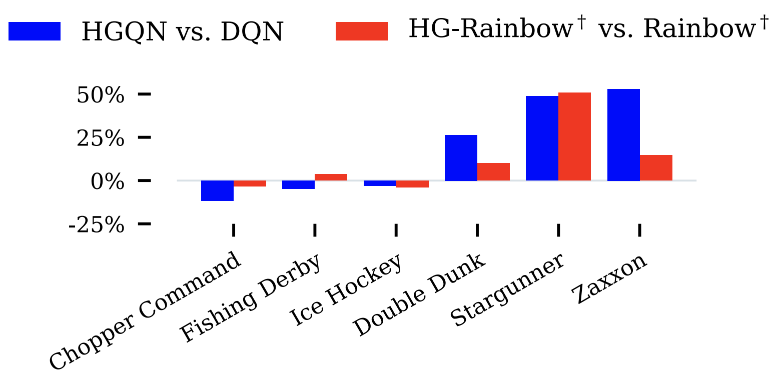


Figure 4.5: Difference in human-normalised score for six Atari 2600 games with 18 valid actions, featuring the three best and worst-performing games from Figure 4.4a (positive % means HGQN outperforms DQN or HG-Rainbow† outperforms Rainbow† over 200 training iterations).

most significantly should feature a kind of structure that is exploitable by our approach. Therefore, given that our approach is notionally orthogonal to the extensions in Rainbow†, we can also expect to see improvements by HG-Rainbow† over Rainbow†. Figure 4.5 shows the relative human-normalised score of HG-Rainbow† versus Rainbow† (three random seeds in each case) along with those of HGQN versus DQN from Figure 4.4a for these six games, sorted according to Figure 4.4a (blue bars). We see that the signs of relative scores for Rainbow†-based runs are mostly aligned with those for DQN-based runs. Notably, in Stargunner the relative improvements are on par in magnitude. See Figure B.2 for full learning curves across these six select Atari 2600 games.

The latter experiment mainly serves to demonstrate the practical feasibility of combining our approach with several DQN extensions. However, as each extension in Rainbow† impacts the learning process in certain ways, we believe that substantial work is required to establish whether such extensions are theoretically sound when combined with the learning dynamics of value decomposition. In fact, a recent study on the properties of linear value-decomposition methods in cooperative multi-agent reinforcement learning could hint at a potential theoretical incompatibility with certain replay schemes (Wang et al., 2021). We defer such a study to future work.

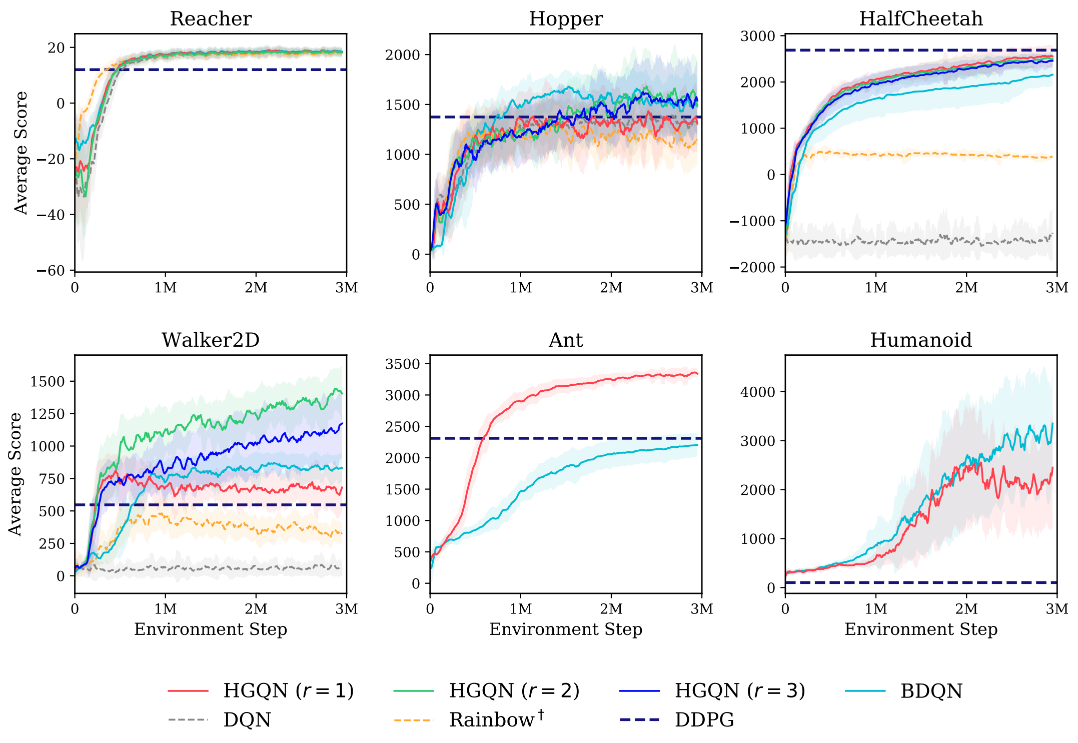


Figure 4.6: Learning curves for HGQN, BDQN (with mean update targets), DQN, and a simplified version of Rainbow in our physical control benchmarks (nine random seeds). Shaded regions indicate standard deviation. Average performance of DDPG trained for three million environment steps is provided for illustration purposes.

4.4.3 Physical Control Benchmarks

We test our approach in problems with larger action spaces by returning to the physical control benchmarks from the previous chapter, under the same settings used to produce the results of Figure 3.1. (See Appendix A.2 for a brief overview of these environments.) The implementation and hyperparameters of HGQN closely follow those of BDQN (see Section 3.3.1). Once again, we apply the same heuristic of dividing the final hidden layer equally across the number of network heads. In this way, the architecture of a 1-complete HGQN model becomes equivalent to that of BDQN for each environment. Therefore, the memory requirements for the 1-complete HGQN models and BDQN are exactly the same.

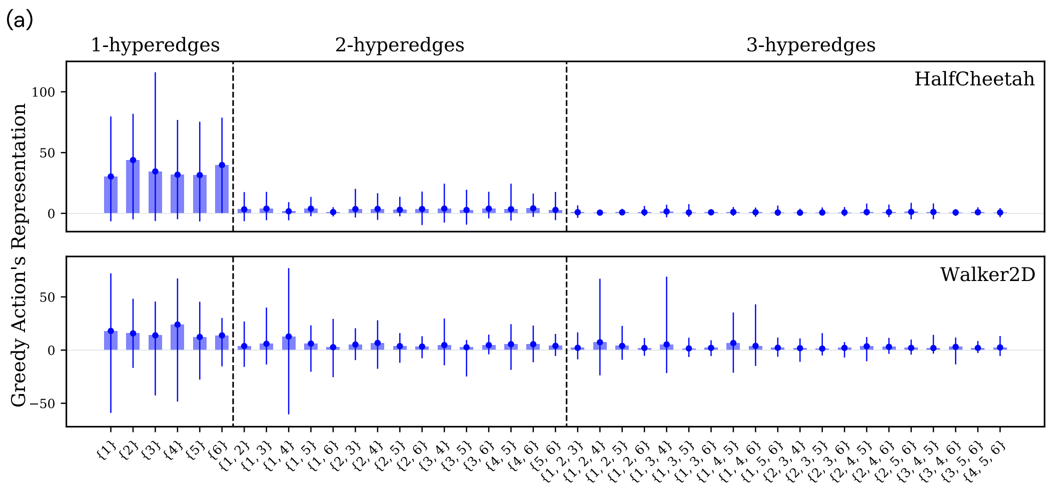
We run HGQN with increasingly more complete hypergraphs, specified using Equation (4.4) by varying rank r from 1 to 3. Figure 4.6 shows the learning curves for HGQN, BDQN (with mean update targets), and our baselines from Section 3.3.2. We see that HGQN is generally competitive with BDQN. Notably, in the Ant environment BDQN is significantly outperformed by the

1-complete HGQN agent using the same model capacity. This could be due to the better coordination properties of the latter approach (related to findings by Sunehag et al. (2017)). We also see that the 1-complete HGQN models in our experiments are able to scale computationally to high-dimensional action spaces. In fact, as we shall discuss in Section 4.6, the computational complexity of our 1-complete HGQN models using the summation mixer (or any strictly monotonic mixer) is the same as that of BDQN.

We also see that HGQN significantly outperforms the other baselines. Remarkably, HGQN outperforms DDPG in almost all cases despite using the training routine of DQN, without involving policy gradients or a specialised exploration strategy. Moreover, the significant gap between the performances of Rainbow† and vanilla HGQN supports the orthogonality of our approach with respect to the extensions in Rainbow†.

In Walker2D we see a clear advantage for including the 2-hyperedges, but additionally including the 3-hyperedges relatively degrades the performance. This suggests that a rank-2 hypergraph strikes the right balance between bias and variance, where including the 3-hyperedges causes higher variance and, potentially, overfitting. In HalfCheetah we see little difference across hypergraphs, despite it having the same action space as Walker2D. This suggests that the 1-complete model is sufficient in this case for learning a good action-value estimator.

Figure 4.7 shows average per-hyperedge representation of the greedy action learned by our approach. Specifically, we evaluated nine trained rank-3 HGQN models using a greedy policy for 10,000 steps in each case. We collected the greedy action's corresponding representation at each step (i.e. one action-representation value for each hyperedge) and averaged them per hyperedge across steps. The error bars show the maxima and minima of these representations. In HalfCheetah the significant representations are on the 1-hyperedges. In Walker2D, while 1-hyperedges generally receive higher average representations, there is one 2-hyperedge that receives a comparatively significant average representation. The same 2-hyperedge also receives the highest variation of values across steps. This 2-hyperedge, denoted $\{1, 4\}$, corresponds to the left and right hips in the agent's morphology (Figure 4.7b). A good bipedal walking behaviour, intuitively, relies on the hip joints acting in unison.



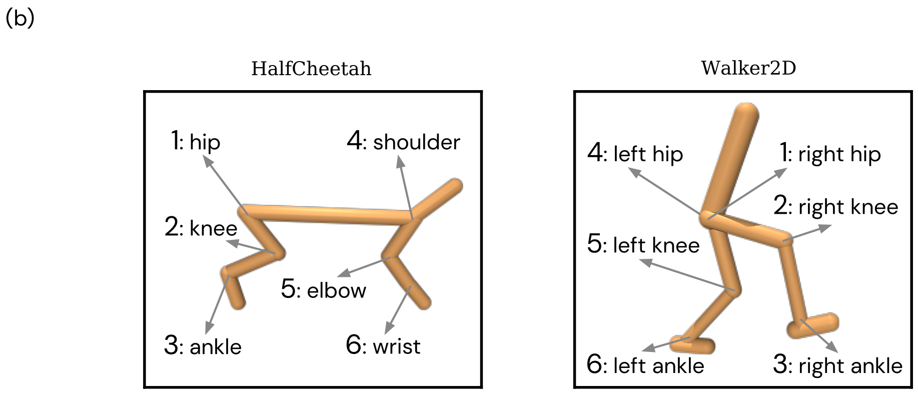


Figure 4.7: (a) Average (bars) and minimum-maximum (error bars) per-hyperedge representation of the greedy action over 90,000 steps (nine trained rank-3 HGQN models, 10,000 steps each). (b) The actuation morphology of each respective domain is provided for reference.

Therefore, modelling their joint interaction explicitly enables a way of obtaining coordinated representations. Moreover, the significant representations on the 3-hyperedges correspond to those including the two hip joints. This perhaps suggests that the 2-hyperedge representing the hip joints is critical in achieving a good walking behaviour, and that any representations learned by the 3-hyperedges are only learned due to the lack of identifiability in value decomposition (see Section 4.3).

Notice that, similarly to the 1-complete HGQN agent, BDQN is outperformed by higher-order HGQN models in Walker2D, where coordination between the hip joints seems to be key in reaching a higher performance level. This further demonstrates the importance of addressing the coordination issue by representing higher-order combinations, a general possibility with action hypergraph networks that cannot be obtained by action branching methods.

4.5 Related Work

Value decomposition has been studied in cooperative multi-agent reinforcement learning under the paradigm of centralised training but decentralised execution. In this context, the aim is to decompose joint action values into agent-wise values such that acting greedily with respect to the local values yields the same joint actions as those that maximise the joint action values. A sufficient but not necessary condition for a decomposition that satisfies this property is the strict monotonicity of the mixing function (Rashid et al., 2018). An instance is VDN (Sunehag et al., 2017) which learns to decompose joint action values onto a sum of agent-wise values. QMIX (Rashid et al., 2018) advances this by learning a strictly increasing nonlinear mixer, with a further conditioning on the (global) state using hypernetworks (Ha et al., 2017). These are multi-agent counterparts of our 1-complete models combined with the respective mixing functions. To increase the representational capacity of these methods, higher-order interactions need to be represented. In a multi-agent reinforcement learning context this means that fully-localised maximisation of joint action values is no longer possible. By relaxing this requirement and allowing some communication between the agents during execution, *coordination graphs* (Guestrin et al., 2002) provide a framework for expressing higher-order interactions in a multi-agent setting. Recent works have combined coordination graphs with neural networks and studied them in multi-agent one-shot games (Castellini et al., 2019) and multi-agent reinforcement learning benchmarks (Böhmer et al., 2020). Our work repurposes coordination graphs from cooperative multi-agent reinforcement learning as a method for action representation learning in standard reinforcement learning.

Sharma et al. (2017) proposed a model on par with a 1-complete model in our framework and evaluated it in multiple Atari 2600 games. In contrast to HGQN, their model shows little improvement beyond DQN. This is likely due to not including higher-order hyperedges, which in turn limits the representational capacity of their model.

Graph networks (Scarselli et al., 2009) have been combined with policy gradient methods for training modular policies that generalise to new agent morphologies (Wang et al., 2018; Pathak et al., 2019; Huang et al., 2020).

The global policy is expressed as a collection of graph policy networks that correspond to each of the agent's actuators, where every policy is only responsible for controlling its corresponding actuator and receives information from only its local sensors. The information is propagated based on some assumed graph structure among the policies before acting. Our work differs from this literature in many ways. Notably, our approach does not impose any assumptions on the state structure and, thus, is applicable to any problem with a multi-dimensional discrete action space. The closest to our approach in this literature is the concurrent work of Kurin et al. (2021) in which they introduce a transformer-based approach to bypass having to assume a specific graph structure.

4.6 Conclusion

In this chapter we described a class of models for learning action representations that enable fast generalisation from limited data by leveraging the combinatorial structure of multi-dimensional actions at various orders. We showed that, by only representing the lowest-order hyperedges, our approach can yield a class of models that achieve competitive performance with BDQN and have the same computational requirements. Notably, this class of models removes the need for independent learners and enables us to use unified updates without imposing any change to the reinforcement learning method. As such, much like a convolutional network which can be seamlessly incorporated into an agent's model, our proposed models can be combined in succession with those for state representation learning to leverage the combinatorial structure in actions. Additionally, our approach enables higher-order combinations of the sub-action spaces to also contribute to the estimation of action values. While this comes at a higher computational cost, it presents the possibility to achieve faster generalisation in problems with moderate action dimensionality without sacrificing accuracy, or lowering the representational capacity. In a sense, our approach spans a spectrum with the conventional approach at one end and action-branching style approaches at the other, thus allowing us to choose our models depending on the task complexity or to incorporate a priori knowledge about the task.

We demonstrated the potential benefit of using a more generic mixing function in our bandit problems (Section 4.4.1). However, in an informal study, on a subset of our reinforcement learning benchmarks we did not see any improvements beyond our non-parametric summation mixer. Further exploration of more generic (even state-conditioned) mixing functions is an interesting direction for future work.

The requirement to maximise over the set of possible actions limits the applicability of Q-Learning in environments with high-dimensional action spaces. In the case of approximate Q-Learning, this limitation is partly due to the cost of evaluating all possible actions' values before an exact maximisation can be performed. Our approach can bypass these issues in certain cases, e.g. when using a 1-complete hypergraph with a strictly monotonic mixer (Rashid et al., 2018). To more generally address such issues, we can maximise over a sampled set of actions instead of the entire set of possible actions (Van de Wiele et al., 2020). Such approximate maximisation will enable including higher-order hyperedges in environments with high-dimensional action spaces.

The above discussion is specific to the combination of our approach with Q-Learning. Nevertheless, our approach can be combined with other action-value methods. For instance, it can be used to learn a critic in an actor-critic method where an action drawn from the actor only needs to be evaluated by the critic in a single forward pass through the model.

We demonstrated the practical feasibility of combining our approach with a simplified version of Rainbow. An extensive empirical study of the impact of these combinations, as well as combining with further extensions, is left for future work. Moreover, a better understanding of how value decomposition affects the learning dynamics of approximate Q-Learning could help establish which extensions are theoretically compatible with our approach.

An intriguing direction for future research is exploring regularisation strategies for promoting the emergence of useful sparse hypergraph structures (see, e.g., Sakryukin et al. (2020)). Another interesting direction is to formulate the problem of hypergraph specification as architecture search in a meta-learning problem setting.

Part II

Temporal Credit Assignment

This part of the thesis describes ideas for addressing the *temporal* version of the credit assignment problem in reinforcement learning. In other words, we study some aspects of the credit assignment problem that have to do with determining *when* the behaviour that deserves credit for influencing an outcome occurred during the sequence of interactions (Sutton, 1984). In particular, we identify and analyse general training scenarios that hinder temporal credit assignment either due to using a *time limit* during the interaction process or a *low discount factor* (thus establishing a short effective time-horizon).

To address the first matter, we characterise the ways in which time limits may be interpreted in reinforcement learning. This leads us to formalise how they should be handled in each case in order to obtain correct temporal credit assignment. To address the second matter, we investigate why approximate solution methods tend to entirely fail when combined with low discount factors. We produce a possible explanation for this phenomenon which, in turn, leads us to develop a method that enables a much larger range of discount factors by rectifying the hypothesised root cause.

Chapter 5

Time Limits in Reinforcement Learning

5.1 Introduction

The interaction between a reinforcement learning agent and its environment is commonly broken down into sequences, or episodes, by using *time limits*—the maximum amount of time an interaction sequence can last. In this case the discounted return formulation (2.2) can be explicitly rewritten as follows:

$$G_{t:T} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T\,, \qquad (5.1)$$

where T denotes the time limit which is generally a finite random variable (i.e. T may vary from one episode to another). Nonetheless, for ease of exposition and without loss of generality, throughout our discussions we assume that T is *fixed* across episodes.

In this chapter[1] we formalise how time limits should be handled and explain why not doing so can cause state-aliasing and invalidate experience replay, leading to suboptimal policies and learning instability. In particular, we identify two ways in which time limits can be interpreted in the context of reinforcement learning and describe how to appropriately handle each case. We now introduce the two cases and outline our consideration for each case.

Optimising directly for the expectation of the return formulation (5.1) is suitable for naturally *time-limited tasks* in which the agent has to maximise its expected return over a time-limited horizon. Under this notion of optimality, the objective of the agent does not go beyond the time limit. Hence, in this

[1]This chapter is based on the following paper: Pardo, F., **Tavakoli, A.**, Levdik, V., and Kormushev, P. (2018). Time limits in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 4045–4054.

case the agent could, for example, learn to take more risky actions that lead to higher expected returns as approaching the time limit. In Section 5.2 we study this case and illustrate that, due to timeout terminations, the remaining time is part of the environment state and essential to its Markov property. As such, we argue for the inclusion of a notion of the remaining time in the agent's input. We refer to this approach as *time-awareness*.

On the other hand, optimising for the expectation of the standard return formulation (2.2) is relevant for *time-unlimited tasks* in which the interaction is not limited in time by nature. In this case the agent has to maximise its expected return over an indefinite (possibly infinite) horizon. Nevertheless, it is often desirable to use time limits to diversify the agent's experience. In Section 5.3 we show that, in order to learn good time-unlimited policies, it is important to distinguish between terminations that are due to reaching a time limit and those due to reaching a terminal state. Specifically, for bootstrapping methods (dynamic programming and TD learning), we argue for bootstrapping at states where termination is due to reaching a time limit or, more generally, any other arbitrary causes. We refer to this approach as *partial-episode bootstrapping*.

We first demonstrate the effectiveness of our considerations in an illustrative control problem using tabular Q-Learning. Then, we evaluate the impact of our considerations on a range of physical control problems using Proximal Policy Optimisation (PPO) (Schulman et al., 2017), demonstrating significant improvements for both of our considerations. Lastly, we show that the negative impact of large experience replay buffers from Zhang and Sutton (2017) can be vastly reduced if time limits are properly handled.

5.2 Time-Awareness for Time-Limited Tasks

In tasks that are time-limited by nature, the learning objective is to directly optimise the expectation of the return $G_{t:T}$ from Equation (5.1). Interactions are systematically terminated at a predetermined time T if no environmental termination occurs earlier. Such time-wise terminations should be interpreted as transitioning to a terminal state whenever the time limit is reached. Thus, the states of the environment, formally an MDP, contain a notion of the remaining time used by the state-transition distribution. This time-dependent MDP can be thought of as a stack of T time-independent MDPs followed by one that only transitions to a terminal state. Therefore, at each time step $t \in \{0, 1, \ldots, T-1\}$ actions result in transitioning to a next state in the next MDP in the stack.

In this case, a time-unaware agent has to effectively act in a partially observable environment, where the states that only differ by their remaining time appear identical. This phenomenon is a form of *state-aliasing* (Whitehead and Ballard, 1991) which is known to lead to suboptimal policies and learning instability due to the infeasibility of appropriate *temporal credit assignment*. In the case of a time-unaware agent in a naturally time-limited task, the issue arises specifically from the fact that timeout terminations can only be attributed to stochasticity in the environment; in which case the time-unaware agent could perceive an additional chance of transitioning to a terminal state from any given state in the underlying MDP. In fact, the perceived termination distribution depends on the agent's behaviour policy and, as such, is *nonstationary* from the agent's perspective. For example, consider an MDP with a fixed underlying initial state and zero probability for transitioning from the initial state to a terminal state. If a time-unaware agent always chooses to remain in its initial state during the entire course of an episode, it would then perceive the probability of termination from that initial state to be $1\,/\,T$. On the contrary, if the agent always chooses to move away from its initial state, it would then perceive said probability to be zero.

In view of the above, we consider *time-awareness* for reinforcement learning agents in time-limited tasks by directly including the remaining time $T-t$ in the agent's input or, more generally, by providing a way to infer it. For example, for a given policy π, we write the state-value function at state s and time t for a time-aware agent in a time-limited environment with time limit T like this:

$$V_\pi(s, T-t) \doteq \mathbb{E}_\pi \left[G_{t:T} \,|\, S_t = s \right], \tag{5.2}$$

where s denotes a non-Markov state that becomes Markov only when it is combined with a notion of the remaining time $T-t$. This consideration resolves the issues of time-unawareness for bootstrapping (dynamic programming and TD learning) and non-bootstrapping methods (Monte Carlo methods).

We now identify three additional issues of time-unawareness that are relevant specifically in the context of bootstrapping methods. To do so, we start off by showing time-awareness for the simplest bootstrapping method: one-step TD prediction, or TD(0), for learning an estimator of state value. By letting V denote an estimate of the state-value function, the one-step TD update for a time-aware agent is as follows:

$$V(s, T-t) \leftarrow (1-\alpha)V(s, T-t) + \alpha y\,, \tag{5.3}$$

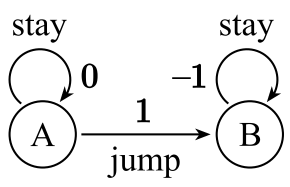
where y is the one-step TD target:

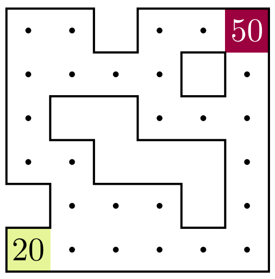
$$y \doteq \begin{cases} r & \text{at all terminations} \\ r + \gamma V(s', T-t-1) & \text{otherwise} \end{cases}\,. \tag{5.4}$$

Similarly to before, s and s' denote non-Markov states that become Markov only when they are combined with a notion of the remaining time. A time-aware agent uniquely identifies all possible underlying states and appropriately decides when to perform bootstrapping. On the other hand, a time-unaware agent undergoes conflicting updates (sometimes bootstrapping and sometimes not) for the same perceived state transition s to s'. These conflicting updates are problematic. Firstly, they can cause delusive value estimates due to a leakage of values to out-of-reach states. Secondly, they can yield a high estimation variance whenever the difference between bootstrapping and non-bootstrapping targets is large. Thirdly, in control problems (where value estimates are commonly used to infer or update the behaviour policy) the ratio of these conflicting updates for a given state s is nonstationary as the termination distribution changes with the behaviour policy.

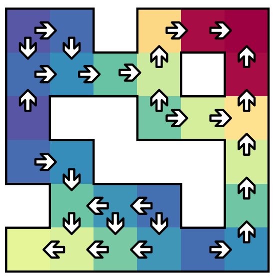
5.2.1 The Last Moment Problem

To give a simple example of where the optimal policy is time-dependent (i.e. time-awareness is critical for both bootstrapping and non-bootstrapping methods), we consider an MDP consisting of two states A and B. The agent always starts in A and has the possibility to choose to *stay* in place with a reward of 0 or *jump* to B with a reward of +1. However, B is a trap state with no exit where the only possible action provides a penalty of −1. The episodes terminate after a fixed number of time steps T. Thus, assuming the true objective is not discounted, the optimal deterministic policy is to jump right before the time limit is reached. For

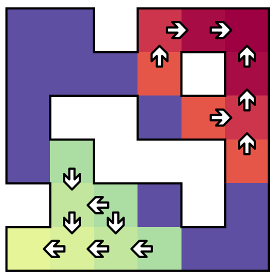




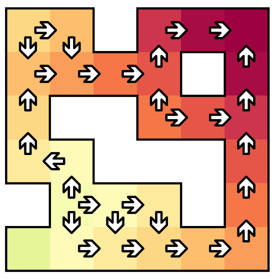
(a) Environment



(b) Standard



(c) Time-awareness



(d) Partial-episode bootstrapping

Figure 5.1: An illustration of the Two-Goal Gridworld problem ($T = 3$) together with colour-coded state values and policies learned by tabular Q-Learning with and without our considerations. (a) The environment. (b) The standard agent is time-unaware and treats timeouts as environmental terminations. It learns to always go for the nearest goal even if there is not enough time to reach it. (c) The time-aware agent maximises its return over the fixed horizon and learns to stay in place when there is not enough time to reach a goal. (d) The agent with partial-episode bootstrapping maximises its return over an indefinite horizon and learns to go for the most rewarding goal.

a time-unaware agent, this task is impossible to solve optimally for any $T > 1$, with the best learnable deterministic policy being to stay in place. In contrast, a time-aware agent can learn to stay in place for $T - 1$ steps and then jump.

5.2.2 The Two-Goal Gridworld Problem

To further show the impact of state-aliasing in time-unaware agents, we consider a deterministic gridworld environment with two possible goals rewarding 50 for reaching the top-right and 20 for the bottom-left grids (Figure 5.1a). The agent has five actions to choose from: moving in the four cardinal directions or staying in place. A movement in any direction incurs a penalty of -1, while staying in place generates a reward of 0. Episodes terminate on reaching a goal or after 3 time steps. The initial state is uniformly sampled at the start of each episode from the set of possible grids excluding the goals.

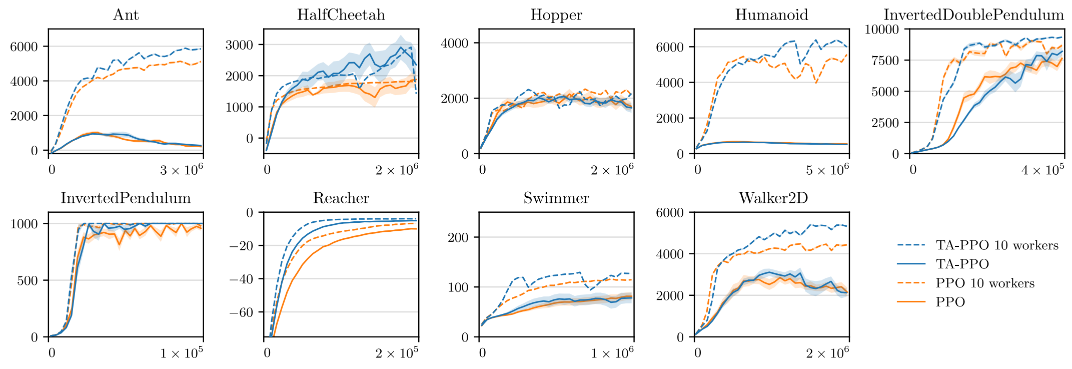
We train tabular Q-Learning with and without time-awareness until convergence using a uniform random behaviour policy, a decaying learning rate, and a discount factor of 0.99. As shown in Figure 5.1c, the time-aware agent quickly learns the optimal policy, which is to go for the closest goal only when there is enough time or to stay in place otherwise. On the other hand, as shown in Figure 5.1b, the standard time-unaware agent learns a policy that always tries to go for the closest goal even if there is not enough time to reach it. Interestingly, the optimal policy in this example can be represented without any information about the remaining time. Nonetheless, this example clearly shows that the conflicting updates during training due to state-aliasing can cause a leakage of values to out-of-reach states and lead to suboptimal policies. We remark that non-bootstrapping methods are not susceptible to such leakage as they use complete returns as opposed to bootstrapping.

5.2.3 Physical Control Benchmarks

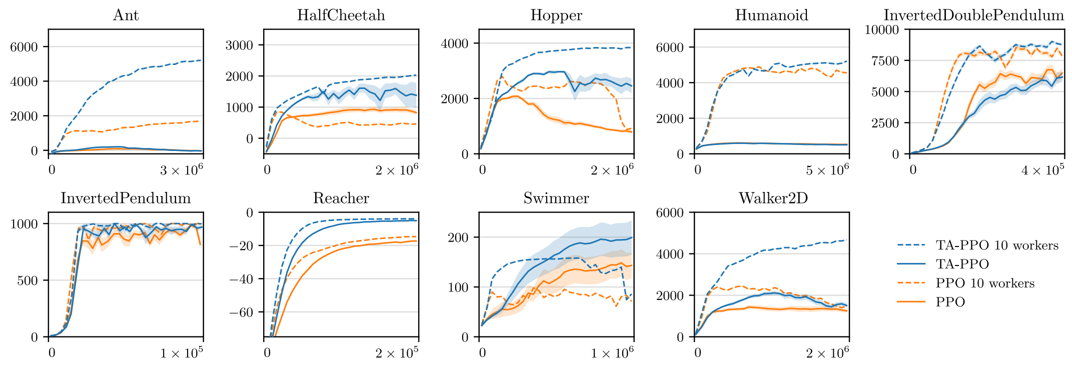
We now evaluate the performance of Proximal Policy Optimisation (PPO) (Schulman et al., 2017) with and without inclusion of the remaining time as part of the agent's input on nine physical control benchmarks from the OpenAI Gym's MuJoCo collection (Todorov et al., 2012; Brockman et al., 2016). We refer the reader to Section 2.6.1 for a description of the PPO algorithm and to Appendix A.2 for a brief overview of the environments. We use the PPO implementation from OpenAI Baselines (Dhariwal et al., 2017) with a diagonal-covariance Gaussian policy distribution parameterisation, using the hyperparameters reported by Schulman et al. (2017) (see Table 5.1). For the time-aware PPO, we concatenate the remaining time represented by a scalar (normalised from -1 to 1) to the MuJoCo states. By default, these environments use predefined time limits and do not distinguish between timeout and

Hyperparameter	**Value**
λ (decay parameter)	0.95
K (truncation horizon)	2048
ϵ (clipping parameter)	0.2
minibatch size	64
number of minibatch updates	10
optimiser	Adam
learning rate	0.0003

Table 5.1: PPO hyperparameters.



(a) $\gamma = 0.99$



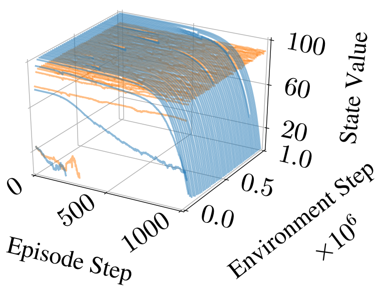
(b) $\gamma = 1$

Figure 5.2: Learning curves for PPO with and without the remaining time in input on several physical control benchmarks from the OpenAI Gym's MuJoCo collection ($T = 1000$ for all except Reacher with $T = 50$). The averaged sum of rewards and standard errors are shown against the number of environment steps (10 random seeds). Results are shown for two discount factors: (a) $\gamma = 0.99$ and (b) $\gamma = 1$. The time-aware PPO (TA-PPO) generally outperforms the standard PPO, especially for the case with a higher discount factor of 1.

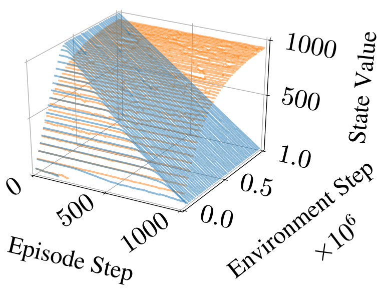
environmental terminations.

Figure 5.2 shows that time-awareness significantly improves the learning performance of PPO. To better understand the differences between the time-aware and time-unaware agents, we now highlight some observations.

As shown in Figure 5.2a, for a discount factor of 0.99, the standard time-unaware PPO agent is oftentimes initially on par with the time-aware PPO agent but later reaches a lower final performance. This is due to the fact that in some domains (e.g. Humanoid) the agents only start to experience timeout terminations more frequently as they become better, at which point the time-unaware agent begins to perceive inconsistent returns for seemingly similar states. The advantage of time-awareness becomes even clearer in the case of a discount factor of 1 (Figure 5.2b) where the time-unaware PPO agent often diverges drastically. This is mainly because, in this case, the time-



(a) $\gamma = 0.99$



(b) $\gamma = 1$

Figure 5.3: Learned state-value estimates on InvertedPendulum ($T = 1000$) by the time-aware (blue) and the standard time-unaware (orange) PPO agents. The time-aware agent quickly learns an accurate estimator for state value, while the standard time-unaware agent slowly learns a constant estimate. Results are shown for two discount factors: (a) $\gamma = 0.99$ and (b) $\gamma = 1$.

unaware agent suffers from much more significant conflicts as returns are now the undiscounted sum of rewards.

Figure 5.3 shows the learned state-value estimates in InvertedPendulum over episodes of interaction during training, perfectly illustrating the difference between time-aware and time-unaware agents in terms of their estimated expected returns. While time-awareness enables PPO to learn an accurate decay of the expected return with respect to the remaining time, the time-unaware PPO only learns a constant estimate. We must note that both agents learn good policies that quickly reach a goal state early on during training. In other words, shortly after the training process begins, the agents learn to quickly reach a goal state and then maintain that state for the remainder of the episode. As such, the graphs of Figure 5.3 show, for the most part, the learned state-value estimates of a goal state over episodic time steps.

In naturally time-limited tasks where the agents have to maximise their performance for a limited time, time-aware agents can demonstrate interesting ways of achieving this objective. Figure 5.4 shows the average final pose of the time-aware and time-unaware PPO agents in Hopper using $T = 300$. Here, the time-aware agent robustly learns to jump forward as approaching the time limit in order to maximise its expected return, achieving a “photo finish.” Moreover, Figure 5.4 shows a failure mode demonstrated by the time-unaware PPO agent in the case of $\gamma = 1$ where the learned behaviour is to actively stay in place in order to accumulate the bonus for not falling. This suboptimality is caused by the high learning instability due to the conflicting updates.

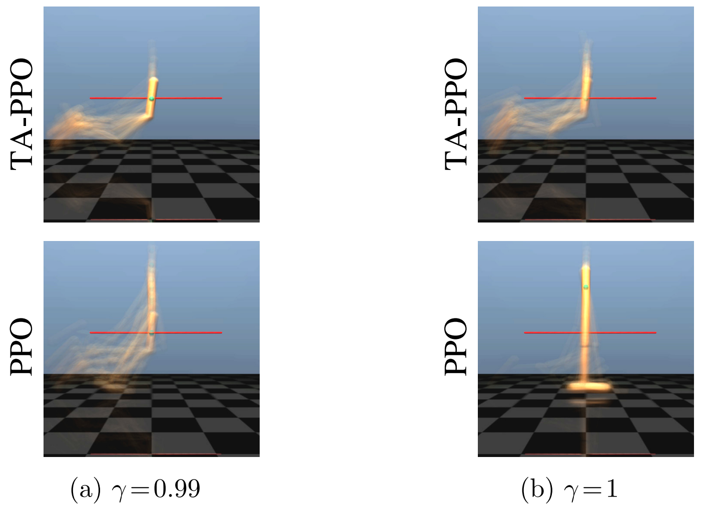


Figure 5.4: Average last pose on Hopper ($T = 300$) with the vertical termination threshold of 0.7 meters in red. The time-aware agent (TA-PPO) learns to jump forward close to the time limit in order to maximise its traversed distance. The time-unaware PPO agent does not learn this behaviour and its training is significantly destabilised when the discount factor is high. Results are shown for two discount factors: (a) $\gamma = 0.99$ and (b) $\gamma = 1$.

5.3 Partial-Episode Bootstrapping for Time-Unlimited Tasks

In tasks that are *not* time-limited by nature, the learning objective is to optimise the expectation of the return G_t from Equation (2.2). While the agent has to maximise its expected return over an indefinite (possibly infinite) horizon, it is often desirable to regularly reset the environment by using time limits in order to increase the diversity of the agent's experience. However, a common misconception is to then treat the auxiliary terminations due to such time limits as environmental terminations. As such, the agent falsely treats its partial episodes of interaction as concrete ones, not accounting for the possible future rewards that could have been experienced if no time limits were used.

In view of the above, in the specific context of bootstrapping methods, we argue for continuing to bootstrap at states where termination is due to the time limit. We refer to this consideration as *partial-episode bootstrapping*. To formalise this, let us consider the example of predicting a state-value function using TD(0) as the simplest bootstrapping method. We remark that the state-value function for a given policy π can be rewritten in terms of the time-limited return $G_{t:T}$ and the value from the last state in the interaction sequence $V_\pi(S_T)$

as below:

$$V_\pi(s) \doteq \mathbb{E}_\pi\left[G_{t:T} + \gamma^{T-t} V_\pi(S_T) \,|\, S_t = s\right] . \tag{5.5}$$

By letting V denote an estimate of the state-value function, the one-step TD update on transition to s' and receiving r is:

$$V(s) \leftarrow (1-\alpha)V(s) + \alpha y\,, \tag{5.6}$$

where y is the one-step TD target for a partial-episode bootstrapping agent:

$$y \doteq \begin{cases} r & \text{at environmental terminations} \\ r + \gamma V(s') & \text{otherwise (including timeouts)} \end{cases} . \tag{5.7}$$

An agent without partial-episode bootstrapping does not bootstrap at timeout terminations. This causes conflicting updates for estimating the value of the same state s, leading to similar issues as those we discussed for conflicting updates in the context of time-unaware agents (Section 5.2).

5.3.1 The Two-Goal Gridworld Problem

We revisit the gridworld environment from Section 5.2.2. While previously the agent's task was to learn an optimal policy for a given time limit, we now consider how an agent can learn a good policy for an indefinite horizon from partial episodes of interaction. We use the same setup and tabular Q-Learning as in Section 5.2.2, but, instead of treating terminations due to time limits as environmental ones, we now maintain bootstrapping from the nonterminal states that are reached at the time limits. This modification allows the agent to learn the time-unlimited optimal policy of always going for the most rewarding goal (Figure 5.1d). On the other hand, while the standard agent which does not perform bootstrapping at nonterminal timeout states has values from out-of-reach states leaking into its learned value function (Figure 5.1b), these updates do not occur in the appropriate proportion to let the agent learn the time-unlimited optimal policy.

5.3.2 Physical Control Benchmarks

For the next experiments we again use PPO but with a modification for partial-episode-bootstrapping to enable the agent to bootstrap when the environment is reset due to time limits and no terminal states are encountered. This involves modifying the implementation of the generalised advantage estimator,

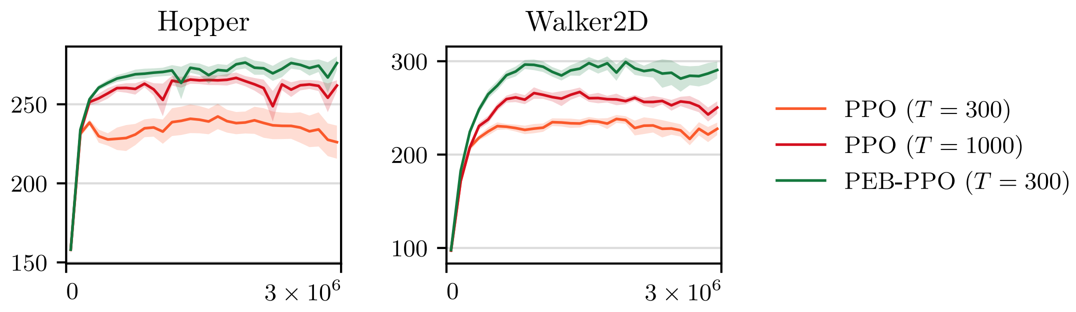


Figure 5.5: Performance comparison of PPO with and without partial-episode bootstrapping (PEB) using $\gamma = 0.99$ on two physical control benchmarks (10 random seeds). The averaged discounted sum of rewards and standard errors are shown against the number of environment steps. The training episodes are limited to 300 or 1000 time steps (indicated in parentheses). The evaluation episodes are limited to one million time steps.

or GAE(λ), from Section 2.6.1. While the latter uses a geometrically-weighted average of truncated k-step returns for bootstrapping which is more complex than the one-step bootstrapping described in Equation (5.7), the same idea of continuing to bootstrap from the nonterminal timeout states is the only modification required for the considered approach.

We consider the Hopper and Walker2D environments from Section 5.2.3 but instead aim to learn a policy that maximises the agent's expected return over a time-unlimited horizon. The goal here is to show that by continuing to bootstrap from nonterminal states at timeout terminations it is possible to learn good policies for time-unlimited tasks from time-limited episodes of interaction. Figure 5.5 shows the learning curves for PPO with and without partial-episode bootstrapping. The training episodes were limited to a maximum of 300 time steps, whereas during evaluation the episodes were limited to a higher maximum of one million time steps. For comparison, we also trained the standard PPO agent with a training time limit of 1000 steps. The results show that partial-episode bootstrapping enables the agent to learn better policies using shorter training episodes of interaction.

5.3.3 The Infinite Cube Pusher Problem

To demonstrate the ability of our agent in optimising for an infinite-horizon objective (as in a continuing task), we introduce a novel physical control environment consisting of a torque-controlled ball on a horizontal plane that is used to push a cube to specified target positions (Figure 5.6a). Every time the cube

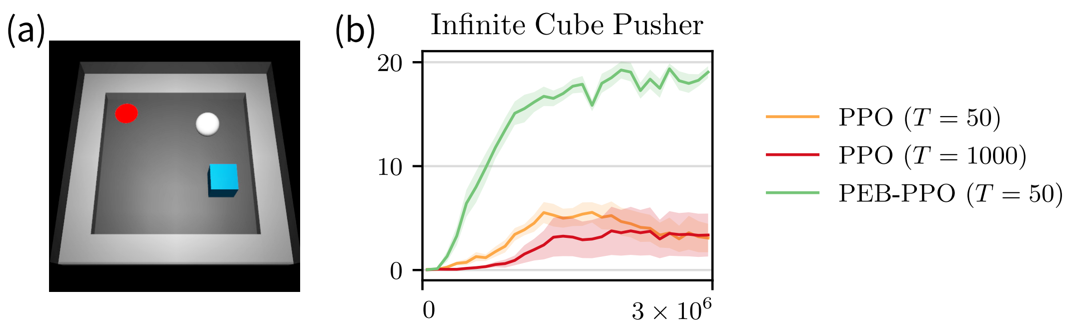


Figure 5.6: Performance comparison of PPO with and without partial-episode bootstrapping (PEB) using $\gamma = 0.99$ on the Infinite Cube Pusher problem (10 random seeds). (a) A sample image of the environment. (b) The averaged number of reached targets and standard errors are shown against the number of environment steps. The training episodes are limited to 50 or 1000 time steps (indicated in parentheses). The evaluation episodes are limited to 1000 time steps.

reaches a target, the agent receives a reward of +1 and the target is moved away from the cube to a new random position. The terrain is surrounded by fixed bounding walls. We force the inner edges of the walls to obstruct the cube, but not the ball, in order to let the agent move the cube even if it is in a corner. The environment state consists of: the coordinates of the ball, cube, and target; the velocities of the ball and cube; and the rotation of the cube.

Due to the infinite-horizon nature of this task and the absence of dense rewards, it is highly useful to break down the interaction during training into partial episodes to diversify experience and improve learning performance. To this end, we use a time limit of 50 steps during training which is sufficient to push the cube to an individual target in most cases. However, during evaluation 1000 time steps were used to allow reaching multiple targets. Figure 5.6b shows that partial-episode bootstrapping significantly improves the learning performance.

5.4 The Effect on Experience Replay

Sampling and training on batches of transitions from a buffer of past experience has proved to be highly effective in stabilising the training of neural networks by decorrelating updates and avoiding the rapid forgetting of rare experiences (Mnih et al., 2015; Schaul et al., 2016). This is known as *experience replay* (Lin, 1992). However, we argue that the perceived non-stationarity that is induced by improper handling of time limits is incompatible with experience

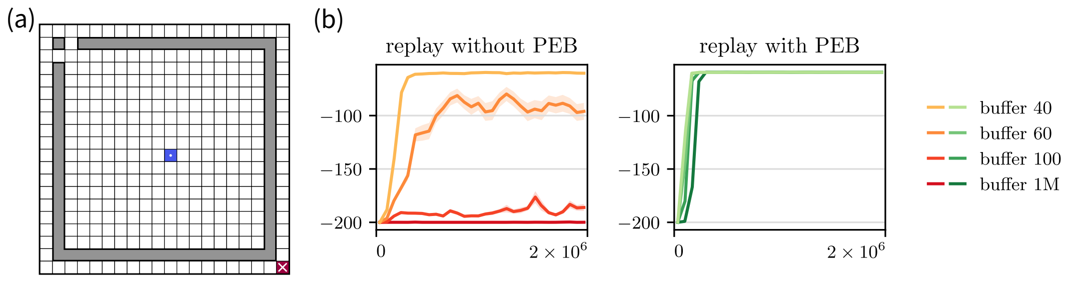


Figure 5.7: Performance comparison of tabular Q-Learning with and without partial-episode bootstrapping (PEB) using $\gamma = 1$ on the Difficult Gridworld problem ($T = 200$) presented by Zhang and Sutton (2017). (a) The environment. (b) The averaged sum of rewards and standard errors are shown against the number of environment steps (30 random seeds). Experience replay significantly hurts the performance where timeout terminations are not properly handled, while by simply continuing to bootstrap whenever a timeout termination is encountered the learning is much faster and varying the buffer size has almost no effect.

replay. Indeed, the distribution of timeout terminations changes with the agent's behaviour and, thus, past transitions become obsolete.

While both time-awareness and partial-episode bootstrapping provide ways to solve this issue, we only illustrate the effect of partial-episode bootstrapping on one of the tasks presented by Zhang and Sutton (2017). In the latter work, the authors demonstrate that experience replay can significantly hurt the learning process if the size of the replay buffer is not tuned well. One of the environments used is a deterministic gridworld with a fixed initial state and a fixed terminal goal state (Figure 5.7a). The agent receives a penalty of -1 for each time step at a nonterminal state and 0 otherwise. As proposed by the authors, tabular Q-Learning is used with values initialised to 0, no discounting, an ε-greedy policy with a fixed 10% chance of random actions, and a time limit of 200 steps. Figure 5.7b shows the learning curves averaged over 30 random seeds. We successfully reproduce the results from the paper showing that the performance deteriorates very quickly with increasing buffer size. We then demonstrate that by simply bootstrapping from nonterminal timeout states the effect of the buffer size is vastly diminished.

5.5 Related Work

The importance of time-awareness for optimising a time-limited objective is well-established in the dynamic programming and optimal control literature. However, as we discussed in this chapter, it has been widely overlooked in the reinforcement learning literature and in the design of popular benchmarks that are aimed at evaluating the progress in reinforcement learning. To the best of our knowledge, the importance of the inclusion of a notion of time in time-limited tasks was first demonstrated in the reinforcement learning literature by Harada (1997), yet it seems to have been widely overlooked. Nonetheless, a major difference between the approach of Harada (1997), namely $\mathrm{Q_T}$-Learning, and the one described in this chapter is that we considered a more general class of time-dependent MDPs, where the reward and the transition distributions may also be time-dependent.

With the aim of unifying task specification in reinforcement learning, White (2017) introduces a way to view episodic tasks as continuing ones through transition-based discounting. While they do not explicitly discuss this possibility, partial-episode bootstrapping can be equivalently obtained through transition-based discounting. Other than this, despite the fact that using auxiliary time limits is highly common, the reinforcement learning literature has been unclear about whether partial-episode bootstrapping is used or not.

In deep reinforcement learning it is highly common to use a stack of previous observations or use recurrent neural networks to address partial observability (Wierstra et al., 2010). These solutions can help when a notion of the remaining time is not included as part of the agent's input. However, including this information is much simpler and allows for better interpretability of the learned policies. Furthermore, the latter should enable better generalisation for dealing with varying time limits.

5.6 Conclusion

We considered the problems of learning optimal policies in time-limited and time-unlimited tasks using time-limited episodes of interaction. When learning policies for naturally time-limited tasks, we showed that it is important for correct temporal credit assignment to include a notion of the remaining time as part of the agent's input. On the other hand, when learning policies for time-unlimited tasks, we showed that it is important for correct temporal credit assignment to continue bootstrapping at the end of partial episodes—whenever termination is purely due to time limits, or more generally any early termination conditions other than reaching a terminal state. In both cases we reported significant improvements in the learning performance for our considerations.

Nevertheless, in many problems, agents that do not incorporate these considerations still manage to perform relatively well. This could be due to numerous reasons. For example, the impact of the conflicting updates can be negligible if the time limits are so large that timeout terminations are hardly experienced or if the discount factor is small enough. Moreover, in the case of time-unaware agents in naturally time-limited tasks there could be observable features that are correlated with time (e.g. the forward distance) or it may be unlikely to observe the same states at different remaining times.

Lastly, we discussed time-awareness and partial-episode bootstrapping as separate cases. However, if the task is time-limited in nature but the episodes of interaction are cut shorter in time due to any arbitrary termination conditions, then the two approaches should be used in conjunction.

Chapter 6

Logarithmic Reinforcement Learning

6.1 Introduction

Using a discount factor $0 < \gamma < 1$ to exponentially decay the present value of future rewards is highly common for task specification in reinforcement learning (see Section 2.2.1 for the discounted return formulation). Sometimes this is intended to specify the true objective one should aim to optimise, in which case γ is to be treated as part of the MDP. However, more often than not, γ serves as a mathematically convenient means to satisfy theoretical convergence conditions (Bertsekas and Tsitsiklis, 1996) or, more generally, as a hyperparameter of the optimisation (Prokhorov and Wunsch, 1997; Xu et al., 2018; Amit et al., 2020). This is because the magnitude of γ can greatly impact the stability of the learning process. For example, when used in conjunction with function approximation, too high of a discount factor is known to generally lead to unstable learning due to overgeneralisation (Durugkar and Stone, 2017; Pohlen et al., 2018) and the performance of low discount factors tends to simply fall flat in practice. As such, if the true objective is best specified using some γ, we often need to resort to a different γ that works well in practice. In fact, having a mismatch between the true objective and one that the agent optimises is not restricted to only having different discount factors. It is nonetheless important to enable using a wider range of discount factors in order to reduce the performance gap due to any such mismatch.

In this chapter[1] we start off by exemplifying some scenarios in which the discount factor plays an important role in reducing the performance gap due to a mismatch between the true objective and optimisation objective (Section 6.2). (These examples also expand our discussions from the previous chapter on the role of time limits in reinforcement learning.) We then look deeper into the effect of the discount factor on the optimisation process. We analyse why, in practice, the performance of low discount factors tends to fall flat when used in combination with function approximation, especially in tasks with long horizons. In particular, we refute a number of common hypotheses and present an alternative one, identifying the primary culprit to be the difference of the *action gap*—the optimal value difference between the best and second-best actions—across the state space. To test this hypothesis, we introduce a new method that yields more homogeneous action gaps in tasks with sparse rewards. This is achieved by mapping the update target to a logarithmic space and performing updates in that space instead. Finally, we demonstrate empirically that our method achieves much better performances for low discount factors than previously possible, providing supporting evidence for our new hypothesis.

6.2 Discounting Effects on Performance Gap

In this section we present examples to substantiate our earlier claims regarding the importance of γ in reducing the performance gap wherever there is a mismatch between the true objective and one that the agent optimises. In Section 5.2 we established that time-awareness is generally needed for the learning of optimal policies in time-limited tasks. However, it is still very common to indirectly optimise for the true time-limited objective through the proxy of a time-unlimited objective. (This is perhaps because a stationary, or time-independent, policy is generally simpler to compute.) In fact, most commonly according to the literature, the true objective is time-limited and undiscounted, while the optimisation objective is time-unlimited and discounted (e.g. as in the seminal work of Mnih et al. (2015)). We consider this common scenario,

[1]This chapter is based on the following paper: van Seijen, H., Fatemi, M., and **Tavakoli, A.** (2019). Using a logarithmic mapping to enable lower discount factors in reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 14134–14144.

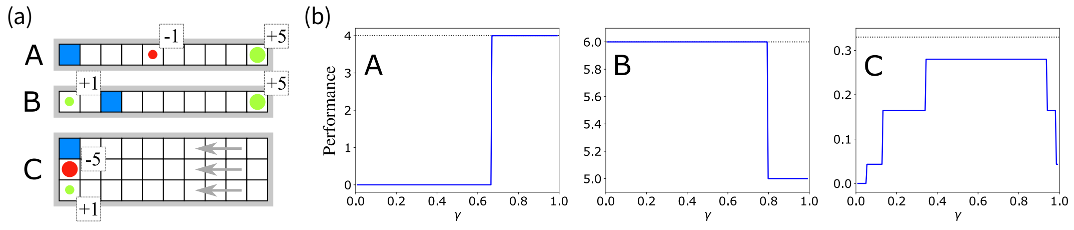


Figure 6.1: (a) Illustrations of our three environments, where the blue grid denotes the fixed initial position, numbers indicate the reward for collecting each object, and arrows signify the wind direction. (b) Performance of the optimal policies for the true objective (black) and optimisation objective (blue) as evaluated with respect to the true objective as a function of γ. The difference between the two represents the performance gap.

with a clear objective mismatch, as the basis for constructing our examples. Explicitly, we design three simple environments (Figure 6.1a) and evaluate the performances of the optimal policies for the true objective (time-limited and undiscounted) and optimisation objective (time-unlimited and discounted) with respect to the true objective and as a function of $0 < \gamma < 1$ (Figure 6.1b). We use a time limit of 12 steps for the true objective.

In each environment, starting from the blue grid, the agent is rewarded for collecting the green objects and penalised for collecting the red objects. The transition dynamics is deterministic in environments A and B. In environment C wind blows along the direction of the arrows, making the agent move towards left with a 40% chance regardless of the performed action.

In environment A, where a small negative reward (-1) has to be traded off for a large positive reward ($+5$) received later, *high* discount factors result in a smaller performance gap. By contrast, in environment B *low* discount factors result in a smaller performance gap. The reason is that for high discount factors the optimal policy (with respect to the optimisation objective) takes the longer route of first collecting the large positive reward ($+5$) before going for the small positive reward ($+1$). However, with a time limit of 12 steps during evaluation, there is not enough time to collect both objects by taking this long route. Using a low discount factor results in the agent taking the shorter route of first collecting the small positive reward and then the large positive reward; thereby the agent manages to collect both positive rewards in time. In environment C a tradeoff needs to be made between taking the shorter route and risk being pushed to the negative reward (-5), due to state-transition stochasticity, versus a longer detour to minimise such a risk. In this environment the optimal policy for the true objective is time-dependent (i.e. time-awareness is critical for representing the optimal policy). Hence, given that the agent optimises a time-unlimited objective and, as such, does not have time-awareness, the performance gap cannot be reduced to 0 for any γ. In this case, discount factors that minimise the performance gap are neither too high nor too low. These examples demonstrate that, as long as there is a mismatch between the true objective and optimisation objective, the best discount factor is problem-specific and can be anywhere in the range between 0 and 1.

It is worth noting that the optimal policy for the time-unlimited optimisation objective can be learned from time-limited episodes of interaction during training. For example, let us consider using Q-Learning to learn the optimal policy for the optimisation objective in any of the environments from Figure 6.1. With a uniform random behaviour policy and training episodes of 12 time steps (just as the time limit of the true objective), there is a nonzero probability for each possible state-action pair being visited in an episode. Hence, with an appropriate schedule for decaying the learning rate, convergence in the limit can be guaranteed (Jaakkola et al., 1994). A key detail to enable this is to utilise partial-episode bootstrapping (Section 5.3). Similarly, the optimal policy for the true objective can be learned using Q-Learning by utilising time-awareness (Section 5.2).

6.3 Discounting Effects on Optimisation

In the previous section our discussion involved the theoretical minimum of the performance gap for each possible γ. That is, we evaluated the performance gap with respect to the true objective using the optimal policies for the true and optimisation objectives. However, γ also affects the optimisation process whereby finding an optimal policy could be more challenging for some discount factors than others. In this section, using the chain walk problem shown in Figure 6.2, we evaluate the correlation between the choice of γ and how challenging it could be to find an optimal policy. Specifically, we use a discounted objective for the purpose of optimisation with different discount factors and measure the proportion of random trials that learn the optimal policy of choosing the left action a_L in every state. In theory, the optimal performance of +1

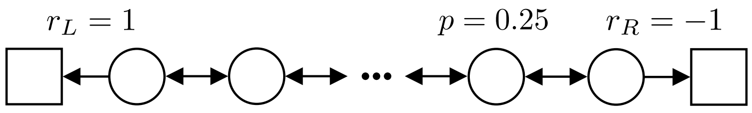


Figure 6.2: Our chain walk problem consisting of 50 nonterminal states and two terminal ones. Each nonterminal state has two actions: a_L which results in transitioning to the left with probability $1-p$ or to the right with probability p, and vice versa for a_R. All rewards are 0, except for transitioning to the far-left or far-right terminal states which result in r_L and r_R, respectively.

is achievable for any choice of γ (i.e. the performance gap can be minimised to 0 for any γ).

To study the optimisation effects under function approximation we use linear function approximation with features constructed by tile coding (Sutton, 1996), using tile widths of 1, 2, 3, and 5 together with an offset of 1. Tile coding a state using a tile width of w and an offset of 1 corresponds to a set of binary features that are nonzero for w neighbouring states and zero elsewhere. We set the number of tilings such that all states receive a full set of features. With a tile coding scheme as such, any value function can be represented. That is, in principle, error-free reconstruction of the optimal action-value function is possible for any discount factor. Note that for $w = 1$ the representation becomes equivalent to a tabular one. To keep our experiment as simple as possible, we remove the role of exploration by performing update sweeps over the entire state-action space (using a learning rate of 0.001) and measure the performance at the end of each full-sweep update.

Figure 6.3a shows the performance of Q-Learning early on during the training (average performance over the first 10,000 sweeps) as well as the final performance (average between the 100,000 and 110,000 sweeps). This experiment demonstrates a common empirical observation: when used in conjunction with function approximation, the performance of low discount factors tends to fall flat in sparse reward problems. More specifically, we have three main observations: (1) there is a sharp drop in the final performance for discount factors below some threshold value; (2) the threshold value depends on the tile width, with larger tile widths resulting in worse (i.e. higher) threshold values; and (3) the tabular representation performs well across all discount factors.

The *action gap* of a state s, denoted by $\Delta_{\text{AG}}(s)$, is defined as the value difference between the best and second-best actions at s for the optimal action-

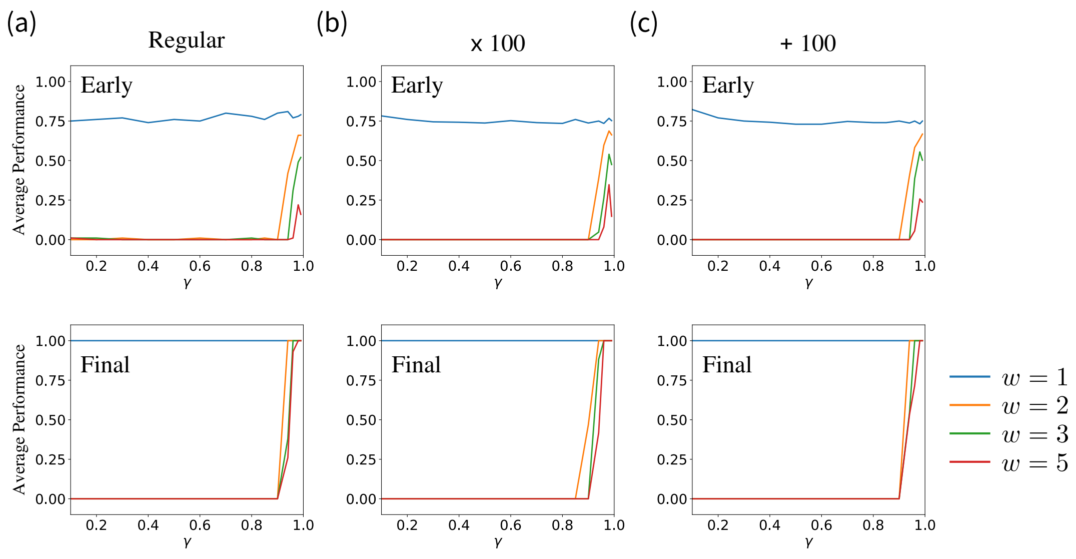


Figure 6.3: Early and final performance of Q-Learning on three variations of the chain walk problem. (a) Performance on the unaltered environment. (b) Performance on the environment variation with 100 times larger action values (i.e. larger action gaps). (c) Performance on the environment variation with action values pushed up by 100 (i.e. lower relative action gaps). All environment variations result in similar performance graphs.

value function Q^*:

$$\Delta_{\mathrm{AG}}(s) \doteq Q^*(s, a^*) - Q^*(s, a^\dagger), \tag{6.1}$$

where a^* is the best action and $a^\dagger$ is the second-best action. Now, if the error for an estimate of the optimal action-value function is smaller than half the action gap in every state, then the greedy policy with respect to such action-value estimates is guaranteed to be equal to the optimal policy (Farahmand, 2011). For this reason, merely having small action gaps is commonly held responsible for the numerous failure modes of reinforcement learning with function approximation (Bellemare et al., 2016), such as that of low discount factors. To examine this common belief, we start by evaluating two straightforward hypotheses involving the action gap: (1) lower discount factors cause poor performance because they result in smaller action gaps; and (2) lower discount factors cause poor performance because they result in smaller *relative* action gaps (i.e. the action gap of s divided by the maximum action-value at s). Given that both hypotheses are supported by the results from Figure 6.3a, we perform more directed experiments to test each hypothesis. To test the first hypothesis, we perform the same experiment as before but with rewards that are a factor of 100 larger, thus increasing the action gaps by a factor of 100.

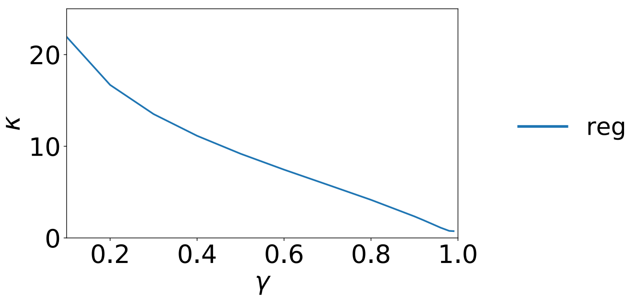


Figure 6.4: Action-gap deviation as a function of the discount factor in the chain walk problem of Figure 6.2, illustrating that the difference in action gaps across the state space becomes larger with lower discount factors.

Therefore, for the first hypothesis to hold, this change should improve (i.e. lower) the threshold value below which the performance falls flat. To test the second hypothesis, we push all action values up by 100 through extra rewards (i.e. adding $100\,(1-\gamma)$ and 100 to the reward on any transition to a nonterminal and terminal state, respectively), thus reducing the relative action gaps. Therefore, for the second hypothesis to hold, this change should degrade the performance. Figures 6.3b and 6.3c show the performance of Q-Learning on these environment variations, together with the performance on the unaltered environment (Figure 6.3a). The performance on both of these environment variations is roughly the same as the performance on the unaltered environment, thus invalidating both hypotheses.

Inspired by the observation that larger tile widths (which perform averaging over a larger set of states) result in a worse performance on lower discount factors (Figure 6.3), we set forth a third hypothesis: lower discount factors cause poor performance with function approximation because they result in a larger difference in action gaps across the state space. To illustrate our statement about the difference in action gaps, we show what such differences look like as a function of γ in the chain walk problem of Figure 6.2. To do so, we introduce a metric, which we call *action-gap deviation* and denote by κ, for capturing a notion of the difference in action gaps across the state space. Explicitly, let S be a random variable with a uniform distribution over $\mathcal{S}^\dagger \subseteq \mathcal{S}$, the set of all nonterminal states with nonzero action gaps. Moreover, let $X \doteq \log_{10}\left(\Delta_{\mathrm{AG}}(S)\right)$. We now define κ to be the standard deviation of the random variable X. Figure 6.4 shows κ as a function of the discount factor for the chain walk problem of Figure 6.2, illustrating our statement that lower discount factors result in a larger difference in action gaps across the states.

To test this new hypothesis, we have to develop a method that reduces the action-gap deviation κ for low discount factors without changing the optimal policy. We do so in the next section.

6.4 Logarithmic Q-Learning

In this section we introduce our new method, called *Logarithmic Q-Learning*, which reduces the action-gap deviation κ in sparse reward problems. We present this method in three steps, in each step adding a layer of complexity to extend the generality of the method. In Appendix C we present the proof of convergence for this method in its most general form. As the first step, we now consider deterministic environments with rewards that are nonnegative.

6.4.1 Deterministic Domains with Nonnegative Rewards

Our method is based on the general approach of mapping the update target to a different space and performing updates in that space instead. We denote the mapping function by f and its inverse by f^{-1}. Just as in Pohlen et al. (2018), with Q-Learning as the basis, values in the mapping space are updated as follows:

$$\widetilde{Q}(S_t, A_t) \leftarrow (1-\alpha)\widetilde{Q}(S_t, A_t) + \alpha f\left(R_{t+1} + \gamma \max_{a'} f^{-1}\Big(\widetilde{Q}(S_{t+1}, a')\Big)\right), \quad (6.2)$$

where $\widetilde{Q}$ is an estimate of the optimal action-value function in the mapping space. To obtain a regular action value Q the inverse mapping has to be applied to $\widetilde{Q}$. Because the updates occur in the mapping space, the action gap of a state s from Equation (6.1) is now defined in the mapping space:

$$\widetilde{\Delta}_{\text{AG}}(s) \doteq \widetilde{Q}^*(s, a^*) - \widetilde{Q}^*(s, a^\dagger)\,. \quad (6.3)$$

Consequently, κ is also measured in the mapping space.

To reduce κ, we propose using a *logarithmic mapping*. Explicitly, we propose the following mapping function:

$$f(x) \doteq c\,\ln(x+\epsilon) + d\,, \quad (6.4)$$

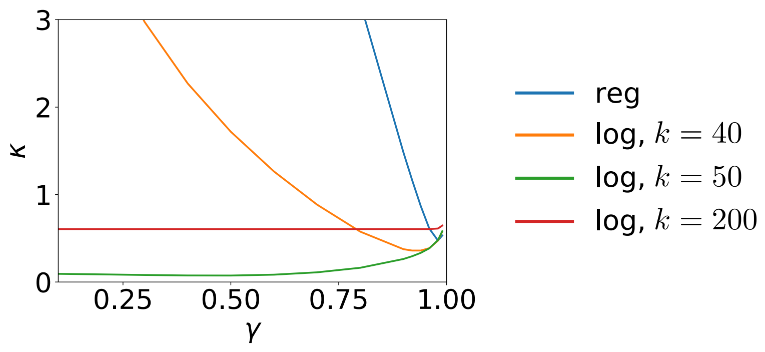


Figure 6.5: Action-gap deviation as a function of the discount factor for a modified version of the chain walk problem from Figure 6.2 with $r_L\!=\!1$, $r_R\!=\!0$, and $p\!=\!0$.

with the inverse function:

$$f^{-1}(x) = e^{(x-d)/c} - \epsilon\,, \tag{6.5}$$

where c, d, and ϵ are the mapping hyperparameters.

To understand the effect of the logarithmic mapping (6.4) on the action-gap deviation κ, we plot κ based on action gaps in the regular space and in the logarithmic space for a modified version of the chain walk problem from Figure 6.2 (with $r_L = 1$, $r_R = 0$, and $p = 0$). In the case of the logarithmic mapping (6.4), we set $\epsilon\!=\!\gamma^k$ with $k \in \{40, 50, 200\}$. We will explain shortly our reasoning behind this choice. Figure 6.5 shows that, with an appropriate value for ϵ ($k\!=\!50$), the action-gap deviation κ can almost be reduced to zero for low discount factors. Setting ϵ too low ($k\!=\!200$) increases the action-gap deviation a little, while setting ϵ too high ($k = 40$) increases the action-gap deviation by a lot for low discount factors. The reason for this is that, effectively, ϵ controls the smallest value in the regular space for which the action gap in the logarithmic space is still significant enough. In the modified version of our chain walk problem this value is about γ^k, the value of the state from which it takes k time steps to experience the reward of $+1$ under the optimal policy. (This is why $k\!=\!50$ works best in our chain walk problem with 50 nonterminal states.) Therefore, by setting $\epsilon = \gamma^k$, we achieve roughly the same effective horizon of k time steps across discount factors.

The parameters c and d, respectively, scale and shift values in the logarithmic space and do not have any effect on the action-gap deviation. The parameter d controls the initialisation of the action values. Setting d as

$$d = -c\,\ln(q_{init} + \epsilon) \tag{6.6}$$

ensures that $f^{-1}(0) = q_{init}$ for any choice of c, k, and γ. This can be useful in practice, e.g. when using neural networks to represent $\widetilde{Q}$, as it enables standard initialisation methods (which produce output values around 0) while still ensuring that the initialised $\widetilde{Q}$ values correspond to q_{init} in the regular space. The parameter c scales values in the logarithmic space. For most tabular and linear methods, scaling values does not affect the optimisation process. However, such scaling can significantly impact the optimisation process for deep reinforcement learning methods. In all our experiments, except for the deep reinforcement learning ones, we use $c = 1$ and set d according to Equation (6.6) using $q_{init} = 0$.

In stochastic environments the approach described in this section is problematic. This is because averaging over stochastic samples in the logarithmic space produces an underestimate as compared to averaging in the regular space and then mapping the outcome to the logarithmic space. More precisely, in deterministic environments a state-action pair (s, a) results deterministically in a next state s' and a next reward r. As such, $y = r + \gamma \max_{a'} Q(s', a')$, or the averaging target for $Q(s, a)$, is deterministic. In this case we have:

$$\mathbb{E}\left[\ln(y)\right] = \ln\left(\mathbb{E}[y]\right) = \ln(y).$$

On the other hand, in environments with stochastic transition dynamics or rewards either or both of the resultant next state or next reward could be random variables, thus denoted respectively by capital letters S' and R to reflect this. As such, $Y = R + \gamma \max_{a'} Q(S', a')$, or the averaging target for $Q(s, a)$, is stochastic. In this case, from Jensen's inequality, we have:

$$\mathbb{E}\left[\ln(Y)\right] \leq \ln\left(\mathbb{E}[Y]\right).$$

Fortunately, within our specific context, there is a way around this limitation that we discuss in the next section.

6.4.2 Stochastic Domains with Nonnegative Rewards

The learning rate α generally conflates two forms of averaging: averaging of stochastic update targets due to environment stochasticity and, in the case of function approximation, averaging over different states. In order to amend our method for stochastic environments, ideally, we would separate these two forms of averaging and perform the averaging over stochastic update targets in the regular space and the averaging over different states in the logarithmic space. (This idealised notion is related to our observation in Section 6.3 that larger tile widths, which perform averaging over a larger set of states, result in a worse performance on lower discount factors.) While such a separation is hard to obtain, the approach presented below achieves many of the same benefits. In particular, it enables convergence of $\widetilde{Q}$ to $f(Q^*)$ even when the environment is stochastic.

Let $\beta_{\log}$ and β_{reg} be the learning rates for averaging in the logarithmic space and regular space, respectively. We amend our method from the previous section by computing an alternative update target that is based on performing an averaging operation in the regular space. Specifically, the update target U is transformed into an alternative update target $\hat{U}$ as follows:

$$\hat{U} \leftarrow f^{-1}\left(\widetilde{Q}(S_t, A_t)\right) + \beta_{\text{reg}}\left(U - f^{-1}\left(\widetilde{Q}(S_t, A_t)\right)\right), \tag{6.7}$$

where

$$U \doteq R_{t+1} + \gamma \max_{a'} f^{-1}\left(\widetilde{Q}(S_{t+1}, a')\right). \tag{6.8}$$

The modified update target $\hat{U}$ is used for updates in the logarithmic space:

$$\widetilde{Q}(S_t, A_t) \leftarrow \widetilde{Q}(S_t, A_t) + \beta_{\log}\left(f(\hat{U}) - \widetilde{Q}(S_t, A_t)\right). \tag{6.9}$$

The effective learning rate for the combination of these updates is the product $\alpha = \beta_{\text{reg}}\,\beta_{\log}$. Notice that if $\beta_{\text{reg}} = 1$, then $\hat{U} = U$ and the update (6.9) reduces to the update (6.2) with $\alpha = \beta_{\log}$.

We now fix the effective learning rate $\alpha = \beta_{\text{reg}}\,\beta_{\log}$ to 0.001 and test different combinations of values for β_{reg} and $\beta_{\log}$. Figure 6.6 shows RMS (root mean square) error curves, as a measure of the difference between $f^{-1}(\widetilde{Q}(s,a))$ and $Q^*(s,a)$ over all state-action pairs, on a modified version of the chain walk problem from Figure 6.2 with $r_L = 1$, $r_R = 0$, and $p = 0.25$. The results are based on a tile width of 1 (i.e. tabular representation) and $k = 200$. These results show that β_{reg} should be sufficiently low to reduce the TD errors in the logarithmic space, thus keeping the underestimation of values due to averaging in the logarithmic space under control. Note that for $\beta_{\text{reg}} = 1$, which reduces the revised method to that in the previous section, the error does not reduce to near zero.

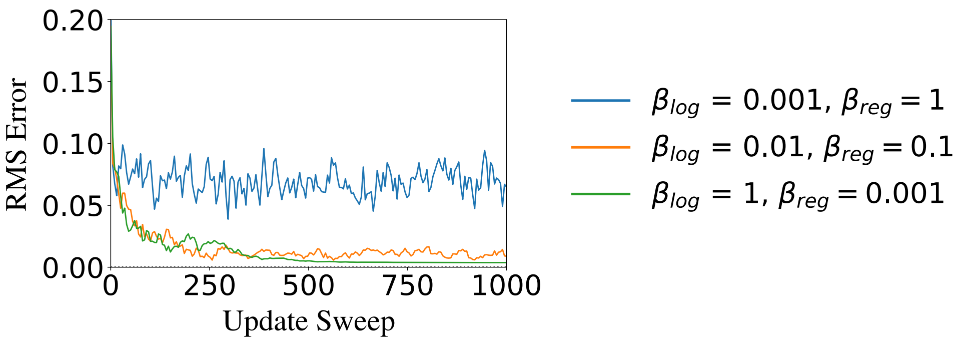


Figure 6.6: RMS error curves, based on the difference between $f^{-1}(\widetilde{Q}(s,a))$ and $Q^*(s,a)$ over all state-action pairs, on a modified version of the chain walk problem from Figure 6.2 with $r_L\!=\!1$, $r_R\!=\!0$, and $p\!=\!0.25$. A low enough β_{reg} is necessary to keep the underestimation of values due to averaging in the logarithmic space under control.

6.4.3 Stochastic Domains with General Rewards

We now consider the general case of stochastic environments with rewards that can be any real number. Consider the decomposition of the reward r into two components r^+ (nonnegative) and r^- (nonpositive) as follows:

$$\begin{aligned} r^+ &\doteq \begin{cases} r & \text{if } r > 0 \\ 0 & \text{otherwise} \end{cases} \\ r^- &\doteq \begin{cases} |r| & \text{if } r < 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned} . \tag{6.10}$$

Note that the decomposed rewards r^+ and r^- are always nonnegative and that

$$r = r^+ - r^- \tag{6.11}$$

at all times. By decomposing the rewards in this way, two separate utility functions, $\widetilde{Q}^+$ and $\widetilde{Q}^-$, can be trained in the logarithmic space using each reward component, r^+ and r^-. The regular action value function Q can then be obtained using $\widetilde{Q}^+$ and $\widetilde{Q}^-$ as follows:

$$Q(s,a) \doteq f^{-1}\left(\widetilde{Q}^+(s,a)\right) - f^{-1}\left(\widetilde{Q}^-(s,a)\right). \tag{6.12}$$

Importantly, we use this action-value function Q to find maximising actions for performing our updates as well as for greedy action selection. Therefore, $\widetilde{Q}^+$ and $\widetilde{Q}^-$ are not intended to estimate the optimal action-value functions for r^+ and r^- in the mapping space. Rather it is only their composition via Equation (6.12) into Q, our estimator for Q^* of the original reward signal r,

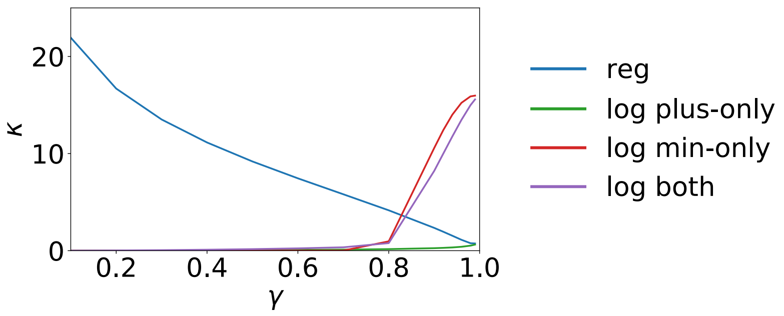


Figure 6.7: Action-gap deviation as a function of the discount factor in the original chain walk problem from Figure 6.2 calculated based on the action gaps of $\widetilde{Q}^+$ ("log plus-only"), $\widetilde{Q}^-$ ("log min-only"), both $\widetilde{Q}^+$ and $\widetilde{Q}^-$ ("log both"), and Q^* ("reg").

that is meaningful. Putting it all together, to perform the update (6.7) we obtain two separate update targets for each reward component:

$$\begin{aligned} U^+ &\doteq R^+_{t+1} + \gamma f^{-1}\left(\widetilde{Q}^+(S_{t+1}, a')\right) \\ U^- &\doteq R^-_{t+1} + \gamma f^{-1}\left(\widetilde{Q}^-(S_{t+1}, a')\right) \end{aligned}, \tag{6.13}$$

where

$$a' \doteq \arg\max_{a'} \left(f^{-1}\Big(\widetilde{Q}^+(S_{t+1}, a')\Big) - f^{-1}\Big(\widetilde{Q}^-(S_{t+1}, a')\Big)\right). \tag{6.14}$$

In turn, these update targets are transformed using the update (6.7) into $\hat{U}^+$ and $\hat{U}^-$, respectively, which are then transformed using the update (6.9) into $\widetilde{Q}^+$ and $\widetilde{Q}^-$, respectively.

We now demonstrate κ for the original version of the chain walk problem with $r_L = 1$, $r_R = -1$, and $p = 0.25$ (Figure 6.2). Because there are two functions, $\widetilde{Q}^+$ and $\widetilde{Q}^-$, we need to generalise our definition of κ for this situation. We consider three variants: (1) κ is based on the action gaps of $\widetilde{Q}^+$ ("log plus-only"); (2) κ is based on the action gaps of $\widetilde{Q}^-$ ("log min-only"); and (3) κ is based on the action gaps of both $\widetilde{Q}^+$ and $\widetilde{Q}^-$ ("log both"). Figure 6.7 shows such κ variants (using $k = 200$) together with κ based on the action gaps of Q^* ("reg"), same as that in Figure 6.4. Interestingly, only for the "log plus-only" variant κ is small across discount factors. This is because under the optimal policy (computed by acting greedily with respect to the composition of $\widetilde{Q}^+$ and $\widetilde{Q}^-$ using Equation (6.12) on convergence to Q^*) the chance that the agent moves from a state near the positive terminal state to the negative terminal state is very small. Consequently, $k = 200$ is too small to make the action gaps for $\widetilde{Q}^-$ homogeneous across the state space. Nevertheless, as we

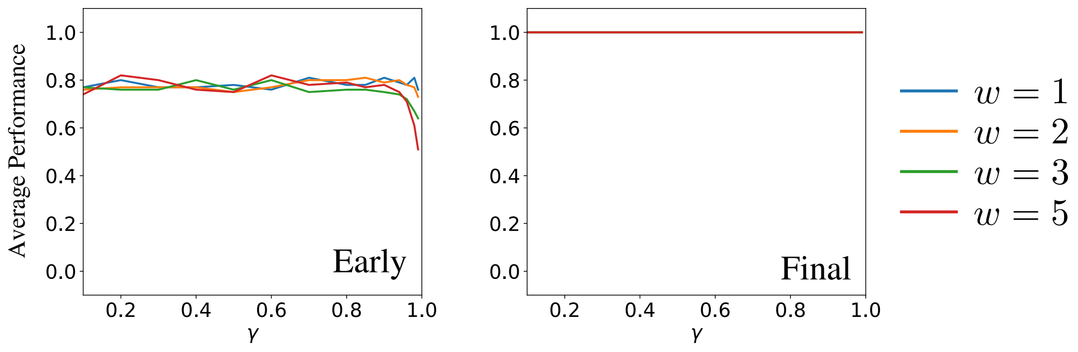


Figure 6.8: Early and final performance of Logarithmic Q-Learning on our original chain walk problem.

see in the next section, $k = 200$ results in a good performance across discount factors, suggesting that not having homogeneous action gaps for $\widetilde{Q}^-$ is not a huge issue. We think this is due to the different nature of behaviours for positive and negative rewards; that is, it may be worthwhile to traverse a long distance to obtain a positive reward, but avoiding a negative reward typically requires only a short effective horizon.

6.5 Experiments

6.5.1 Chain Walk

We test our method by returning to the original version of the chain walk problem (Figure 6.2), under the same settings used to produce the results of Figure 6.3a. We use $k = 200$ as well as $\beta_{\text{reg}} = 0.1$ and $\beta_{\text{log}} = 0.01$, for which the effective learning rate $\beta_{\text{reg}}\beta_{\text{log}}$ is equal to the α used to produce the results of Figure 6.3a. Figure 6.8 shows the performance of Logarithmic Q-Learning early on during the training as well as its final performance. Comparing these graphs with the graphs from Figure 6.3a shows that Logarithmic Q-Learning has successfully resolved the optimisation issues of regular Q-Learning related to the use of low discount factors in conjunction with function approximation.

6.5.2 Atari 2600 Games

We test our approach in Atari 2600 games (see Appendix A.1 for a brief overview), using identical experimentation settings as in Section 4.4.2. For this purpose, we combine our approach with deep Q-networks (Mnih et al., 2015), referring to the resulting agent as *Logarithmic DQN* (LogDQN). Similar to Section 4.4.2, we base our implementation on the Dopamine framework (Castro et al., 2018). We realise our model for LogDQN by doubling the size of the DQN's output layer, using half of it to estimate $\widetilde{Q}^+$ and the other half to estimate $\widetilde{Q}^-$. All the other layers are shared between $\widetilde{Q}^+$ and $\widetilde{Q}^-$ and remain unchanged from DQN. Given that both $\widetilde{Q}^+$ and $\widetilde{Q}^-$ are updated using the same samples, the replay memory does not require any modification and thus the memory footprint does not change. Moreover, because $\widetilde{Q}^+$ and $\widetilde{Q}^-$ are evaluated simultaneously in a single pass through the model, the computational cost of LogDQN remains similar to DQN. While Dopamine provides baselines for 60 games in total, we only consider the subset of 55 games for which human scores have been published.

We optimised the hyperparameters of LogDQN using a subset of six games: Alien, Zaxxon, Breakout, Double Dunk, Space Invaders, and Fishing Derby. For the discount factor we tried $\gamma \in \{0.84, 0.92, 0.96, 0.98, 0.99\}$ and for c we tried $c \in \{0.1, 0.5, 1.0, 2.0, 5.0\}$. Throughout, we used a fix $k = 100$. We also tried the same γ values for DQN and found the default $\gamma = 0.99$ to perform best. For LogDQN, $\gamma = 0.96$ and $c = 0.5$ performed best. We also tried lower γ values, such as $\gamma = 0.1$ and $\gamma = 0.5$, but they did not improve the overall performance over these six games. We set the learning rates of LogDQN to $\beta_{\text{log}} = 0.0025$ and $\beta_{\text{reg}} = 0.1$ so that the effective learning rate $\alpha = \beta_{\text{reg}}\,\beta_{\text{log}}$ is the same as that of DQN ($\alpha = 0.00025$). We set d based on Equation (6.6) with $q_{init} = 1$ for the positive head and $q_{init} = 0$ for the negative head.

Figure 6.9a shows the relative human-normalised score of LogDQN versus DQN over the last 10% of their respective learning curve for each game. Figure 6.9b shows median and mean human-normalised scores across the 55 Atari games for LogDQN and DQN. The results indicate significant improvements in final performance over DQN in a number of games (Figure 6.9a) as well as in overall performance (Figure 6.9b). See Figure B.3 for full learning curves across these 55 Atari 2600 games.

6.6 Conclusion

Our results provide strong evidence for our hypothesis that large differences in action gaps are detrimental to the performance of approximate reinforcement

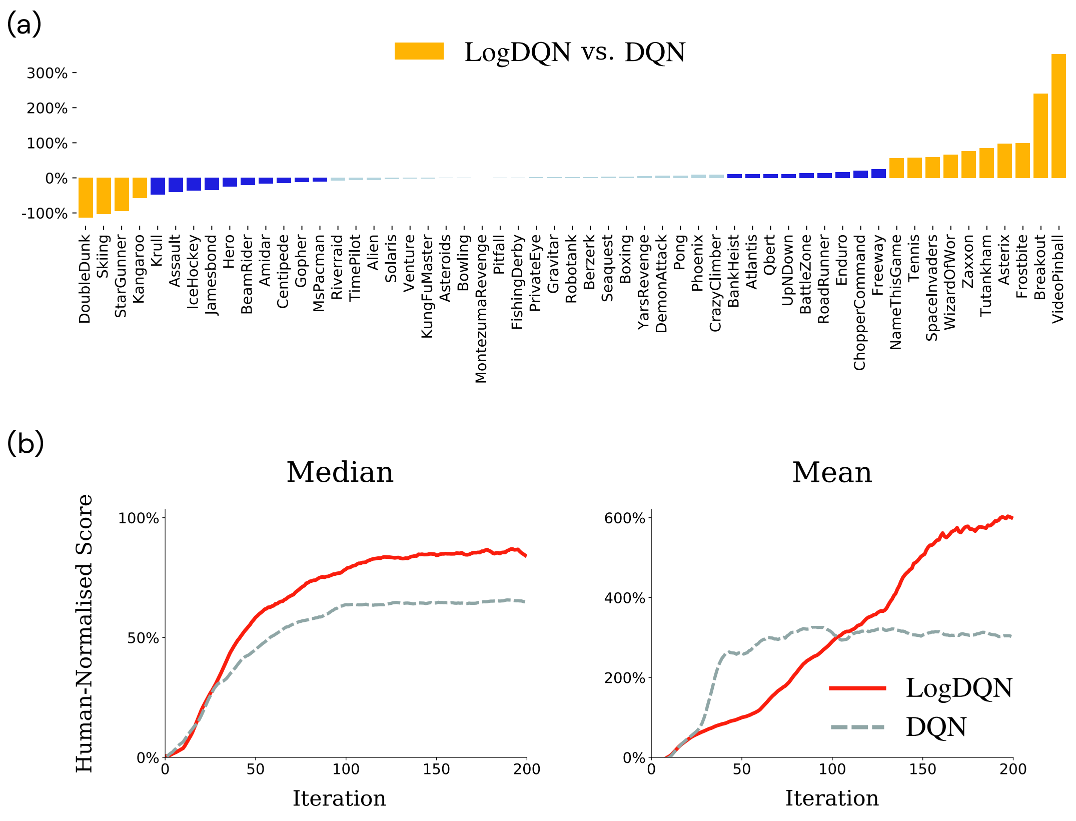


Figure 6.9: (a) Difference in human-normalised score for 55 Atari 2600 games, LogDQN versus DQN over the last 10% of their respective learning curve (positive % means LogDQN outperforms DQN). Orange bars indicate a performance difference larger than 50%, dark-blue bars indicate a performance difference between 10% and 50%, and light-blue bars indicate a performance difference smaller than 10%. (b) Human-normalised median and mean scores across the same set of games.

learning methods. A possible explanation could be that optimising on the squared Bellman error in the conventional approach might drive towards an average squared error that is similar across the state space. However, the error landscape required to bring the approximation error below the action gap in all states has a very different shape if the action gap is different by orders of magnitude across the state space. This mismatch between the required error landscape and that produced by the conventional Bellman error may lead to an ineffective use of the function approximator.

The strong performance we observed for $\gamma = 0.96$ in the deep reinforcement learning setting is unlikely solely due to a difference in the performance gap. We suspect that there are also other effects at play that make LogDQN as effective as it is. On the other hand, at (much) lower discount factors, the performance was not as good as it was for the high discount factors. We had expected that at least for some of the six games a small discount factor (e.g. between $\gamma = 0.1$ and $\gamma = 0.5$) would be better. We believe a possible reason could be that since such low values are very different than the original DQN settings, some of the other DQN hyperparameters might no longer be ideal in the low discount factor region. An interesting future direction would be to reevaluate some of the other hyperparameters for such low discount factors.

Chapter 7

Conclusions and Outlook

This thesis focused on the credit assignment problem in reinforcement learning and described several contributions to each of its main subproblems separately. This chapter discusses the essence of the findings and suggests general directions for future research.

In the first part of the thesis we reviewed the issues that prevent the application of classical reinforcement learning methods to domains with combinatorial and enormous action spaces. We looked at the issues pertaining to credit assignment in such domains and identified the main barrier to be structural, as opposed to temporal. This in turn led us to develop two general classes of methods in order to remedy those issues. The key was to embrace the compositionality in combinatorial action spaces and leverage it for enabling fast generalisation from limited data. This idea proved critical in moderate or high-dimensional action spaces, and improved sample complexity and final performance even in low-dimensional action spaces. In addition, we showed that our methods not only address the structural credit assignment problem but also unleash significant benefits concerning space and time complexity. This in turn allows them to computationally scale to domains far beyond what is possible by the conventional approach. These ideas suggest numerous lines of inquiry for future research. We outline a few of them below.

While leveraging compositionality of combinatorial states underpins nearly all recent success stories of reinforcement learning, similar ideas are not widely explored in the context of combinatorial actions. To a good extent, this is due to the fact that, generally, structural credit assignment is less often studied by the reinforcement learning community. This evokes the notion that there likely are many other unique problems concerning structural credit assignment which need to be studied in the specific context of reinforcement learning; given they do not arise normally in conventional supervised learning. We especially believe that action representation learning needs to be explored in great depth and more generally, beyond only leveraging the compositional structure in combinatorial action spaces. We hope that the success of our work in this direction instigates a wider attention to this topic.

We demonstrated that our methods enable DQN to significantly outperform DDPG on a set of continuous action problems via discretisation. Remarkably, this was accomplished using the training procedure of DQN, without involving policy gradients or a specialised exploration strategy. This manifests the great potential of classical action-value methods in tackling continuous action problems, a problem setting in which policy gradient methods have been the dominant choice. This begs the question of what more could be done to further improve the performance of action-value methods in such domains. Here we prescribe a few possibilities. Firstly, employing exploration strategies that exploit the ordinality of the underlying continuous structure instead of using ε-greedy could prove useful. Moreover, DDPG uses an Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930) to generate temporally-correlated action noise for achieving exploration efficiency in physical control problems with inertia. Using a similar inductive bias could also be useful for discrete action methods. An important aspect of using discrete action methods for discretised continuous action problems is the myriad of available techniques, most of which can be readily employed. For instance, temporally-extended ε-greedy strategies could be used to achieve persistent exploration (Dabney et al., 2021). Another interesting opportunity is using a curriculum of progressively growing action spaces where a coarse discretisation helps exploration and a fine discretisation allows for a more optimal policy (Farquhar et al., 2020).

The intuitions behind our methods, which scaled action-value methods to high-dimensional discrete action domains, also apply to policy gradient methods (Sutton et al., 1999). Recently, a similar idea to our action branching and 1-complete action hypergraph models has been studied in the context of discrete policy gradient methods, yielding superior performance in physical control benchmarks over the Gaussian policy counterparts (Tang and Agrawal, 2020). The same idea has also been successfully applied to the Shadow Dexterous Hand, a humanoid robotic hand with a 20-dimensional action space (Andrychowicz et al., 2020). Given the effectiveness of policy gradient methods with factorised discrete policies, an interesting direction for future work would be to study policies that additionally represent the higher-order combinations of the sub-action spaces (similarly to our action hypergraph models).

We discussed in Chapter 3 that action branching, similarly to independent learning in multi-agent reinforcement learning, is in theory subject to numerous coordination issues. To this end, we introduced action hypergraph networks which reduce such coordination issues by allowing to incorporate higher-order combinations, even bypassing the issues altogether when the highest-order combination is included. Nonetheless, as discussed in Chapter 4, including higher-order hyperedges leads to higher computation and memory demands that quickly become intractable with increasing action dimensionality. The high computational demand stems from the maximisation operation in Q-Learning. The high memory demand is because of having to instantiate a separate network head to represent each hyperedge. One potential solution to address the prohibitive computational demand is to substitute the exact maximisation operation with an approximate one. The memory demand could also be alleviated by sharing parameters among all hyperedges of the same order, thus avoiding the need to store a unique set of parameters for each possible hyperedge. The combination of these ideas with action hypergraph networks is indeed an important direction for future work. Furthermore, exploring other explicit mechanisms to resolve the coordination issues for when higher-order hyperedges are not present would be another interesting direction for future research. For instance, we empirically showed that our 1-complete hypergraph models are able to outperform the action branching methods, which have the same representational capacity. We conjecture that these improvements are due to better coordination properties of the value-decomposition approach (in action hypergraph networks) over independent learning (in action branching).

In the second part of the thesis we studied the two related notions of time limits and discount factors in reinforcement learning. First, we identified two ways in which using a time limit to break down the agent-environment interaction can be interpreted: (1) the time limit indicates the actual period over which the performance should be maximised; or (2) the time limit is arbitrary whereby the performance should be maximised beyond the time limit. We then described considerations that are necessary for correct temporal credit assignment in each training scenario. We discussed that the literature largely overlooks this concept and, as such, fails to commit to either case, which in turn leads to suboptimal policies and learning instabilities. Second, we began with the observation that, in practice, the application of approximate solution methods is confined to a very small range of high discount factors. We then argued that unlocking the capacity to use the full range of discount factors could be critical in obtaining more desirable behaviours, those which better maximise the true objective. To exemplify this, we considered training scenarios where, due to the presence of time limits during evaluation, the magnitude of the discount factor plays a significant role on the learned behaviours. Such a connection between time limits and discount factors is due to the fact that the magnitude of the discount factor establishes an effective time-horizon for the agent, which in turn alters the agent's sense of immediacy with respect to the remaining time. Motivated by this we searched for the root cause of the issue. We identified that the issue is due to large differences in action gaps across the state space, a scenario which is especially prevalent in tasks with sparse rewards. To remedy this, we introduced a method that achieves more homogeneous action gaps by mapping the action-value estimates to a logarithmic space and performing updates in that space instead, thus enabling a much larger range of discount factors to be used with function approximation. While these ideas are important on their own, they cast light on broader lines of inquiry which we discuss below.

We showed that the correct handling of time limits in reinforcement learning is a simple, but important, consideration. In fact, mishandling of time limits has been a major source of confusion in evaluating progress in reinforcement learning research (Tucker et al., 2018). Consequently, we think that studying such simple pathologies contributes to the reinforcement learning community by: (1) enabling a better assessment of new ideas; (2) improving reproducibility; and of course (3) achieving better performances by avoiding instabilities and suboptimal policies.

The success of using a logarithmic mapping to address a shortcoming of the conventional squared Bellman error indicates the importance of studying general nonlinear mappings in reinforcement learning. In other words, considering general nonlinear mappings opens up a large design space of methods that may feature interesting new properties. This line of work aligns with the concurrent general proposition by van Hasselt et al. (2019) for which our logarithmic mapping provides a concrete example.

While using a logarithmic mapping has the desirable property of magnifying near-zero returns, it does the opposite to large returns. In other words, a logarithmic mapping has a too-low slope when encountering large returns, which in turn leads to an over-compression of such returns. This may explain why our LogDQN has performed unfavourably in some dense reward scenarios, such as in the Atari 2600 game of Skiing in which the agent encounters a reward at every step. This requires developing altered solutions in order to maintain the desirable magnification property of the logarithmic mapping for small returns while avoiding the over-compression property for large returns. (See Anonymous (2022) for a recent development in this direction.)

Finally, the ideas discussed in each part of the thesis are orthogonal to one another and, in principle, can be combined. In fact, the results throughout the first part of the thesis were obtained using partial-episode bootstrapping which was introduced in the second part. This combination improved the performances of the proposed methods and our baselines. Therefore, the approach of this thesis serves as a testament to the general usefulness of studying each of the two subproblems in credit assignment (i.e. structural and temporal credit assignment) in isolation by: (1) bringing clarity to the contributions; (2) allowing easier analysis; and (3) yielding a distinct toolbox for each subproblem.

Bibliography

Achiam, J. (2018). Spinning up in deep reinforcement learning. URL: `https://spinningup.openai.com`.

Amit, R., Meir, R., and Ciosek, K. (2020). Discount factor as a regularizer in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 269–278.

Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. (2020). Learning dexterous in-hand manipulation. *International Journal of Robotics Research* 39(1), pp. 3–20.

Anonymous (2022). Orchestrated value mapping for reinforcement learning. Submitted to the International Conference on Learning Representations. Under review. URL: `https://openreview.net/forum?id=c87d0TS4yX`.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 449–458.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47, pp. 253–279.

Bellemare, M. G., Ostrovski, G., Guez, A., Thomas, P., and Munos, R. (2016). Increasing the action gap: New operators for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1476–1483.

Bellman, R. E. (1957). *Dynamic Programming.* Princeton University Press.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35(8), pp. 1798–1828.

Berge, C. (1989). *Hypergraphs: Combinatorics of Finite Sets.* North-Holland.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming.* Athena Scientific.

Bloembergen, D., Tuyls, K., Hennes, D., and Kaisers, M. (2015). Evolutionary dynamics of multi-agent learning: A Survey. *Journal of Artificial Intelligence Research* 53, pp. 659–697.

Böhmer, W., Kurin, V., and Whiteson, S. (2020). Deep coordination graphs. In *Proceedings of the International Conference on Machine Learning*, pp. 2611–2622.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. *arXiv:1606.01540*.

Buşoniu, L., Babuška, R., and De Schutter, B. (2008). A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 38(2), pp. 156–172.

Buşoniu, L., De Schutter, B., and Babuška, R. (2006). Decentralized reinforcement learning control of a robotic manipulator. In *Proceedings of the IEEE International Conference on Control, Automation, Robotics and Vision*, pp. 1–6.

Castellini, J., Oliehoek, F. A., Savani, R., and Whiteson, S. (2019). The representational capacity of action-value networks for multi-agent reinforcement learning. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 1862–1864.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. (2018). Dopamine: A research framework for deep reinforcement learning. *arXiv:1812.06110*.

Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multi-agent systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 746–752.

Coumans, E. and Bai, Y. (2019). PyBullet: A Python module for physics simulation for games, robotics and machine learning. URL: `https://pybullet.org`.

Dabney, W., Ostrovski, G., and Barreto, A. (2021). Temporally-extended ε-greedy exploration. In *Proceedings of the International Conference on Learning Representations*.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018). Implicit quantile networks for distributional reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 1096–1105.

Dayan, P. (1992). The convergence of TD(λ) for general λ. *Machine Learning* 8(3), pp. 341–362.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. (2017). OpenAI Baselines. URL: `https://github.com/openai/baselines`.

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B. (2015). Deep reinforcement learning in large discrete action spaces. *arXiv:1512.07679*.

Durugkar, I. and Stone, P. (2017). TD learning with constrained gradients. In *NeurIPS Deep Reinforcement Learning Symposium*.

Farahmand, A.-m. (2011). Action-gap phenomenon in reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 172–180.

Farquhar, G., Gustafson, L., Lin, Z., Whiteson, S., Usunier, N., and Synnaeve, G. (2020). Growing action spaces. In *Proceedings of the International Conference on Machine Learning*, pp. 4335–4346.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. (2018). Noisy networks for exploration. In *Proceedings of the International Conference on Learning Representations*.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pp. 249–256.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Guestrin, C., Lagoudakis, M., and Parr, R. (2002). Coordinated reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 227–234.

Ha, D., Dai, A., and Le, Q. V. (2017). Hypernetworks. In *Proceedings of the International Conference on Learning Representations*.

Harada, D. (1997). Reinforcement learning with time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 577–582.

Harutyunyan, A. (2020). What is an agent? URL: `http://anna.harutyunyan.net/wp-content/uploads/2020/09/What_is_an_agent.pdf`.

Hassabis, D., Kumaran, D., Summerfield, C., and Botvinick, M. (2017). Neuroscience-inspired artificial intelligence. *Neuron* 95(2), pp. 245–258.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3215–3222.

Huang, W., Mordatch, I., and Pathak, D. (2020). One policy to control them all: Shared modular policies for agent-agnostic control. In *Proceedings of the International Conference on Machine Learning*, pp. 8398–8407.

Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation* 6(6), pp. 1185–1201.

Kriváchy, T., Cai, Y., Cavalcanti, D., **Tavakoli, A.**, Gisin, N., and Brunner, N. (2020). A neural network oracle for quantum nonlocality problems in networks. *npj Quantum Information* 6(1), pp. 1–7.

Kurin, V., Igl, M., Rocktäschel, T., Boehmer, W., and Whiteson, S. (2021). My body is a cage: The role of morphology in graph-based incompatible control. In *Proceedings of the International Conference on Learning Representations*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521(7553), pp. 436–444.

Leottau, D. L., Ruiz-del-Solar, J., and Babuška, R. (2018). Decentralized reinforcement learning of robot behaviors. *Artificial Intelligence* 256, pp. 130–159.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations*.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* 8(3), pp. 293–321.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M. J., and Bowling, M. (2018). Revisiting the Arcade Learning Environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research* 61, pp. 523–562.

Maei, H. R. (2011). *Gradient Temporal-Difference Learning Algorithms*. PhD thesis. University of Alberta.

Matignon, L., Laurent, G. J., and Le Fort-Piat, N. (2012). Independent reinforcement learners in cooperative Markov games: A survey regarding coordination problems. *Knowledge Engineering Review* 27(1), pp. 1–31.

McCarthy, J. (2007). What is artificial intelligence? URL: `http://jmc.stanford.edu/articles/whatisai/whatisai.pdf`.

Metz, L., Ibarz, J., Jaitly, N., and Davidson, J. (2017). Discrete sequential prediction of continuous actions for deep reinforcement learning. *arXiv:1705.05035*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature* 518(7540), pp. 529–533.

Naddaf, Y. (2010). *Game-Independent AI Agents for Playing Atari 2600 Console Games*. Master's thesis. University of Alberta.

Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems* 11(3), pp. 387–434.

Panait, L., Tuyls, K., and Luke, S. (2008). Theoretical advantages of lenient learners: An evolutionary game theoretic perspective. *Journal of Machine Learning Research* 9(13), pp. 423–457.

Pardo, F., **Tavakoli, A.**, Levdik, V., and Kormushev, P. (2018). Time limits in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 4045–4054.

Pathak, D., Lu, C., Darrell, T., Isola, P., and Efros, A. A. (2019). Learning to control self-assembling morphologies: A study of generalization via modularity. In *Advances in Neural Information Processing Systems*, pp. 2295–2305.

Pohlen, T., Piot, B., Hester, T., Azar, M. G., Horgan, D., Budden, D., Barth-Maron, G., van Hasselt, H., Quan, J., Večerík, M., Hessel, M., Munos, R., and Pietquin, O. (2018). Observe and look further: Achieving consistent performance on Atari. *arXiv:1805.11593*.

Prokhorov, D. V. and Wunsch, D. C. (1997). Adaptive critic designs. *IEEE Transactions on Neural Networks* 8(5), pp. 997–1007.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Rashid, T., Samvelyan, M., Schroeder de Witt, C., Farquhar, G., Foerster, J. N., and Whiteson, S. (2018). QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 4295–4304.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-Learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166. Department of Engineering, University of Cambridge.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach.* 3rd edition. Prentice-Hall.

Sakryukin, A., Raïssi, C., and Kankanhalli, M. S. (2020). Inferring DQN structure for high-dimensional continuous control. In *Proceedings of the International Conference on Machine Learning*, pp. 10469–10477.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks* 20(1), pp. 61–80.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. In *Proceedings of the International Conference on Learning Representations.*

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks* 61, pp. 85–117.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations.*

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

Seyde, T., Gilitschenski, I., Schwarting, W., Stellato, B., Riedmiller, M., Wulfmeier, M., and Rus, D. (2021). Is bang-bang control all you need? Solving continuous control with Bernoulli policies. In *Advances in Neural Information Processing Systems.*

Sharma, S., Suresh, A., Ramesh, R., and Ravindran, B. (2017). Learning to factor policies and action-value functions: Factored action space representations for deep reinforcement learning. *arXiv:1705.07269.*

Silver, D. (2018). Deep learning, reinforcement learning, and the credit assignment problem. URL: `https://drive.google.com/file/d/1a7K9XkG62mJk7RmzoU4AGG_wy3Ceybmo/view`.

Singh, S. P., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning* 38(3), pp. 287–308.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. (2017). Value-decomposition networks for cooperative multi-agent learning. *arXiv:1706.05296.*

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis. University of Massachusetts Amherst.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning* 3(1), pp. 9–44.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, pp. 1038–1044.

Sutton, R. S. (2020). John McCarthy's definition of intelligence. *Journal of Artificial General Intelligence* 11(2), pp. 66–67.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction.* 2nd edition. MIT Press.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pp. 1057–1063.

Szepesvári, C. (2010). *Algorithms for Reinforcement Learning.* Morgan & Claypool.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., and Vicente, R. (2017). Multi-agent cooperation and competition with deep reinforcement learning. *PLOS ONE* 12(4), pp. 1–15.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the International Conference on Machine Learning*, pp. 330–337.

Tang, Y. and Agrawal, S. (2020). Discretizing continuous action space for on-policy optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 5981–5988.

Tavakoli, A., Fatemi, M., and Kormushev, P. (2021). Learning to represent action values as a hypergraph on the action vertices. In *Proceedings of the International Conference on Learning Representations.*

Tavakoli, A., Levdik, V., Islam, R., Smith, C. M., and Kormushev, P. (2019). Exploring restart distributions. In *Multidisciplinary Conference on Reinforcement Learning and Decision Making.*

Tavakoli, A., Pardo, F., and Kormushev, P. (2018). Action branching architectures for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4131–4138.

Tessler, C., Tennenholtz, G., and Mannor, S. (2019). Distributional policy optimization: An alternative approach for continuous control. In *Advances in Neural Information Processing Systems*, pp. 1352–1362.

Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033.

Troost, S., Schuitema, E., and Jonker, P. (2008). Using cooperative multi-agent Q-Learning to achieve action space decomposition within single robots. In *Proceedings of the International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems*, pp. 23–32.

Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control* 42(5), pp. 674–690.

Tucker, G., Bhupatiraju, S., Gu, S., Turner, R. E., Ghahramani, Z., and Levine, S. (2018). The mirage of action-dependent baselines in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 5015–5024.

Uhlenbeck, G. E. and Ornstein, L. S. (1930). On the theory of the Brownian motion. *Physical Review* 36(5), pp. 823–841.

Van de Wiele, T., Warde-Farley, D., Mnih, A., and Mnih, V. (2020). Q-Learning in enormous action spaces via amortized approximate maximization. *arXiv:2001.08116.*

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with Double Q-Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 2094–2100.

van Hasselt, H., Quan, J., Hessel, M., Xu, Z., Borsa, D., and Barreto, A. (2019). General non-linear Bellman equations. *arXiv:1907.03687.*

van Hasselt, H. and Wiering, M. A. (2009). Using continuous action spaces to solve discrete problems. In *Proceedings of the International Joint Conference on Neural Networks*, pp. 1149–1156.

van Seijen, H., Fatemi, M., and **Tavakoli, A.** (2019). Using a logarithmic mapping to enable lower discount factors in reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 14134–14144.

Wang, J., Ren, Z., Han, B., Ye, J., and Zhang, C. (2021). Towards understanding cooperative multi-agent Q-Learning with value factorization. In *Advances in Neural Information Processing Systems.*

Wang, T., Liao, R., Ba, J., and Fidler, S. (2018). NerveNet: Learning structured policy with graph neural networks. In *Proceedings of the International Conference on Learning Representations.*

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 1995–2003.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards.* PhD thesis. University of Cambridge.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-Learning. *Machine Learning* 8(3), pp. 279–292.

West, D. B. (1996). *Introduction to Graph Theory.* Prentice-Hall.

White, M. (2017). Unifying task specification in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 3742–3750.

Whitehead, S. D. and Ballard, D. H. (1991). Learning to perceive and act by trial and error. *Machine Learning* 7(1), pp. 45–83.

Wierstra, D., Förster, A., Peters, J., and Schmidhuber, J. (2010). Recurrent policy gradients. *Logic Journal of the IGPL* 18(5), pp. 620–634.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8(3), pp. 229–256.

Xu, Z., van Hasselt, H., and Silver, D. (2018). Meta-gradient reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2402–2413.

Zhang, S. and Sutton, R. S. (2017). A deeper look at experience replay. In *NeurIPS Deep Reinforcement Learning Symposium.*

Appendix A

Standard Benchmarks

Designing generally competent agents raises the question of how to best evaluate them. Ideally, as remarked by Bellemare et al. (2013), the agent should be compared across domains that are (i) varied enough to claim generality, (ii) each interesting enough to be representative of settings that might be faced in practice, and (iii) each created by an independent party to be free from experimenter's bias. To this end, in each contributing chapter of this thesis, we evaluate our agents on either or both of two sets of benchmarks that more or less feature these properties and, as such, are widely adopted by the reinforcement learning community to empirically evaluate general competency. Below we briefly introduce these benchmarking sets.

A.1 Atari 2600 Games

The Atari 2600 is a classic video gaming console featuring a myriad of games, each one different, interesting, and designed to be a challenge for human players. The Arcade Learning Environment (ALE), originally introduced by Bellemare et al. (2013), makes available dozens of Atari 2600 games and presents an evaluation methodology for empirically assessing agents designed for general competency. The usefulness of ALE as a testbed is apparent from the enormous attention it has received from the scientific community and the number of success stories it has facilitated. In this section we outline the interface and some key properties of the ALE environments.

Deep Q-networks (DQN) (Mnih et al., 2015) is the first artificial agent to achieve human-level play in a large fraction of Atari 2600 games, one that we build on whenever evaluating our contributions in these games. Hence,

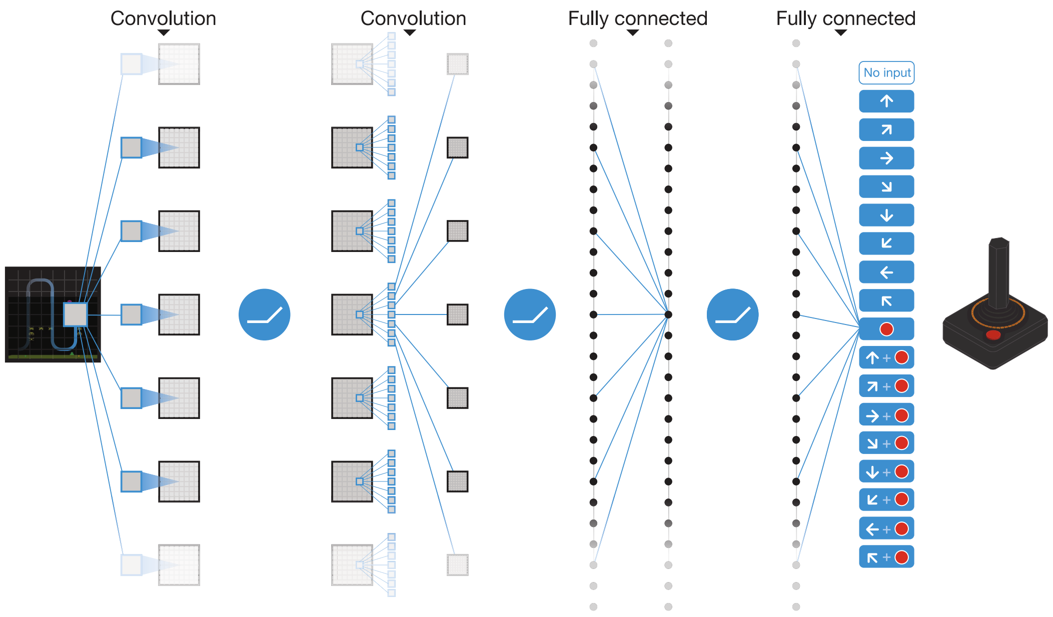


Figure A.1: Schematic illustration of the DQN's network architecture for Atari 2600 games. Image from "Mnih et al. (2015). Human-level control through deep reinforcement learning. *Nature* 518(7540), pp. 529–533."

in what follows, we further describe the preprocessing operations that partly contributed to the broad success of DQN in these games. Figure A.1 shows a schematic illustration of the DQN's network architecture for solving the Atari 2600 games, provided for reference.

Observations A single game screen (frame) in an Atari 2600 game is 160 pixels wide and 210 pixels high, with a 128-colour palette. As such, each observation in the ALE by default consists of a single frame: a two-dimensional array of 7-bit pixels, 160 pixels wide and 210 pixels high (Bellemare et al., 2013). Working directly with these raw frames can be demanding in terms of computation and memory requirements. In the context of DQN, Mnih et al. (2015) employ a basic preprocessing step to reduce the input dimensionality and to deal with some artefacts of the Atari 2600 emulator. First, to encode a single frame they take the maximum value for each pixel colour value over the frame being encoded and the previous frame (this is known as *frame pooling*). This step has proven necessary to remove flickering that is present in games where some objects appear only in even frames while other objects appear only in odd frames, an artefact caused by the limited number of sprites Atari 2600 can display at once. Second, they then extract the luminance channel from the RGB frame and rescale it to obtain a greyscale frame of size 84×84. They apply this preprocessing to the four most recent frames and stack them to produce the input to the network. Such preprocessed observations are used by our agents throughout this thesis to interact with the Atari 2600 games.

Actions The action space of the ALE is defined by the Atari 2600's joystick with three degrees of freedom: a composition of three positions for each of the two axes of the joystick and an optional button press. This implies that these games can have up to 18 unique discrete actions. The joystick controller for Atari 2600 is shown on the right in Figure A.1.

Rewards The reward signal in the ALE corresponds to the difference in the game score (based on the underlying Atari 2600 game) between the previous time step and the current time step. As the scale of scores varies greatly from game to game, DQN replaces each positive reward with 1, each negative reward with -1, and leaves 0 rewards unchanged. Clipping rewards in this manner affects the performance in some games as the agent cannot differentiate between rewards of different magnitude (i.e. agent only observes the sign of the reward). Nevertheless, this simple heuristic limits the scale of the error derivatives and makes it easier to use the same learning rate across numerous games (Mnih et al., 2015). Moreover, the reward function in the Atari 2600, and thus the ALE, is deterministic: given a particular emulator state s and a joystick input a there is a unique resulting next reward r.

Transition Dynamics The transition dynamics in the Atari 2600, and thus the ALE, is deterministic: given a particular emulator state s and a joystick input a there is a unique resulting next state s'. To better represent challenges that could be faced in practice, these games should feature stochastic transition dynamics. To this end, Machado et al. (2018) introduce an addition to the ALE that enables a form of stochasticity called *sticky actions*, whereby with 25% probability the environment executes the action from the previous step instead of the agent's new action. In our Atari 2600 experiments in this thesis, we use this stochastic variant of the ALE to evaluate each agent.

Interactions In the ALE, an agent interacts with the environment in an episodic manner. A standard practice is to restrict the agent's decision points by repeating a selected action for k consecutive frames, where $k = 4$ is commonly used in the literature (Mnih et al., 2015). This is known as *frame skipping* (Naddaf, 2010). We also use this simple technique with $k = 4$ to allow the agent to play more games without significantly increasing the runtime.

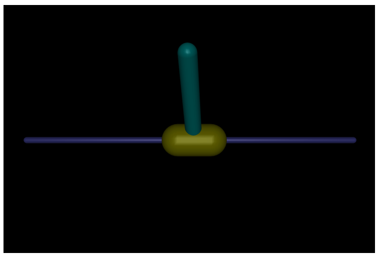
An episode begins by resetting the ALE to its initial configuration and terminates when the game is over. Furthermore, a time limit is commonly used to restrict the maximum length of an episode to $27,000$ agent steps. In some games the player has a number of "lives" which are lost one at a time. There are different ways to deal with this special case, where losing a life should be avoided but is distinct from a "game over." In Dopamine (Castro et al., 2018), the framework which we use throughout this thesis for our Atari 2600 evaluations, an episode does not terminate when the agent loses a life but the game is not over yet. Instead, an artificial termination signal is passed to the agent to help it learn about avoiding "death."

A.2 Physical Control Benchmarks

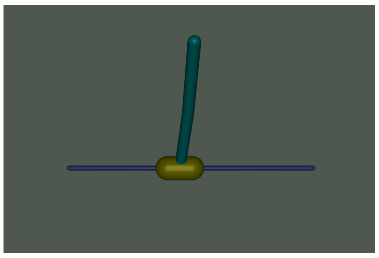
The ALE benchmark offers a set of complex environments with diverse wonky dynamics and an enormous observation space. Nonetheless, the action space of the ALE is limited to a small set of 18 actions. To enable evaluating our agents in arbitrarily larger action spaces with an underlying combinatorial structure, we resort to a benchmarking set of physical control environments. These environments feature continuous action spaces of diverse dimensionality, with the continuous sub-actions along each dimension spanning the range $[-1, 1]$. This provides us with a wide range of combinatorial action spaces for evaluating our agents in Chapters 3 and 4.

By default, these environments return low-dimensional continuous feature observations from which the continuous state can be recovered. Due to this low-dimensional non-pixel nature of observations, training and evaluating agents in these environments is substantially less demanding in terms of computational requirements than the ALE. This is partly why in Chapters 3 and 5 we only evaluate our agents in these tasks, and do not additionally include evaluations in the ALE.

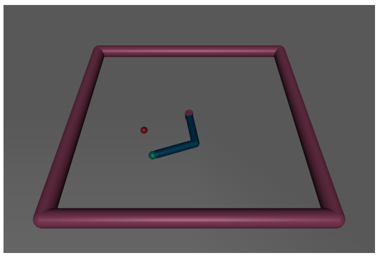
Each environment in this testbed consists of an articulated structure (which is controlled by the agent) in a physical world where all interactions are simulated by a (real-world) physics engine. In our experiments involving this testbed we rely mostly on the same set of environments, albeit based on two different physics engines: PyBullet (Coumans and Bai, 2019) and MuJoCo (Todorov et al., 2012). In what follows, we provide a high-level overview of these environments and their goals. In the descriptions, names are followed by a tuple indicating whether the environment has an existing and stable implementation based on PyBullet or MuJoCo.



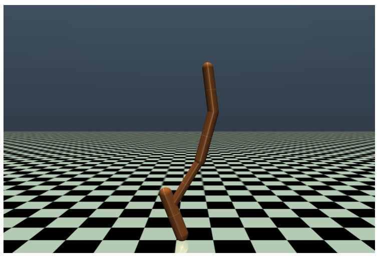
InvertedPendulum `(PyBullet, MuJoCo)` consists of a pole that is mounted on a pivot point on an actuated cart. The goal is to balance the pole by applying horizontal forces to the cart at its base.



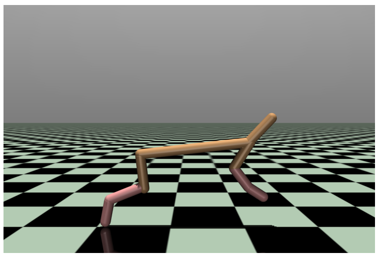
InvertedDoublePendulum `(PyBullet, MuJoCo)` consists of a two-link pole (i.e. a pole on a pole) that is mounted on a pivot point on an actuated cart. The goal is to balance the two-link pole near the upright position by applying horizontal forces to the cart at the base.



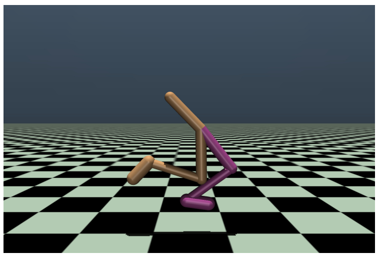
Reacher `(PyBullet, MuJoCo)` is a planar manipulator with two rigid links and two actuated joints. The goal is to control the joints such that the end-effector (green sphere) reaches the target position (red sphere) and then remains there.



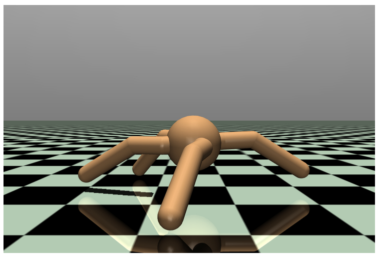
Hopper `(PyBullet, MuJoCo)` is a planar monopod robot with four links (corresponding to the torso, upper leg, lower leg, and foot) along with three actuated joints. The goal is to hop forward as fast as possible without falling.



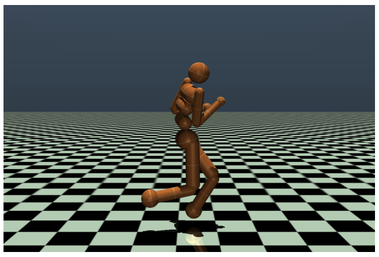
HalfCheetah `(PyBullet, MuJoCo)` is a planar biped robot with eight rigid links, including two legs and a torso, along with six actuated joints. The goal is to run forward as quickly as possible.



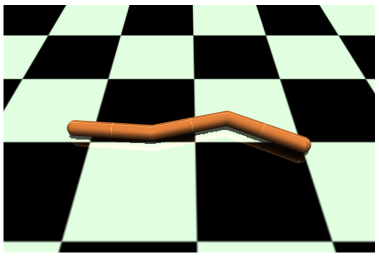
Walker2D `(PyBullet, MuJoCo)` is a planar biped robot consisting of seven rigid links, corresponding to two legs and a torso, along with six actuated joints. The goal is to walk forward as fast as possible without falling.



Ant `(PyBullet, MuJoCo)` is a three-dimensional quadruped robot with 13 rigid links, including a torso and four legs, along with eight actuated joints, two per each leg. The goal is to move forward as quickly as possible while maintaining a normal standing height.



Humanoid `(MuJoCo)` is a three-dimensional humanoid robot consisting of 13 rigid links along with 17 actuated joints. The goal is to move forward as fast as possible without falling.



Swimmer `(MuJoCo)` is a planar robot with 3 rigid links and 2 actuated joints, sliding on a two-dimensional surface with viscous fluid dynamics. The goal is to swim forward as fast as possible.

Appendix B

Learning Curves for Atari 2600 Games

Here we provide complete learning curves for all the Atari 2600 experiments of this thesis:

- Figure B.1 shows complete learning curves for HGQN and DQN across 29 games with 18 valid actions.
- Figure B.2 shows complete learning curves for HGQN, HG-Rainbow†, DQN, and Rainbow† across six select games with 18 valid actions, featuring the three best and worst-performing games from Figure 4.4a.
- Figure B.3 shows complete learning curves for LogDQN and DQN across 55 games.

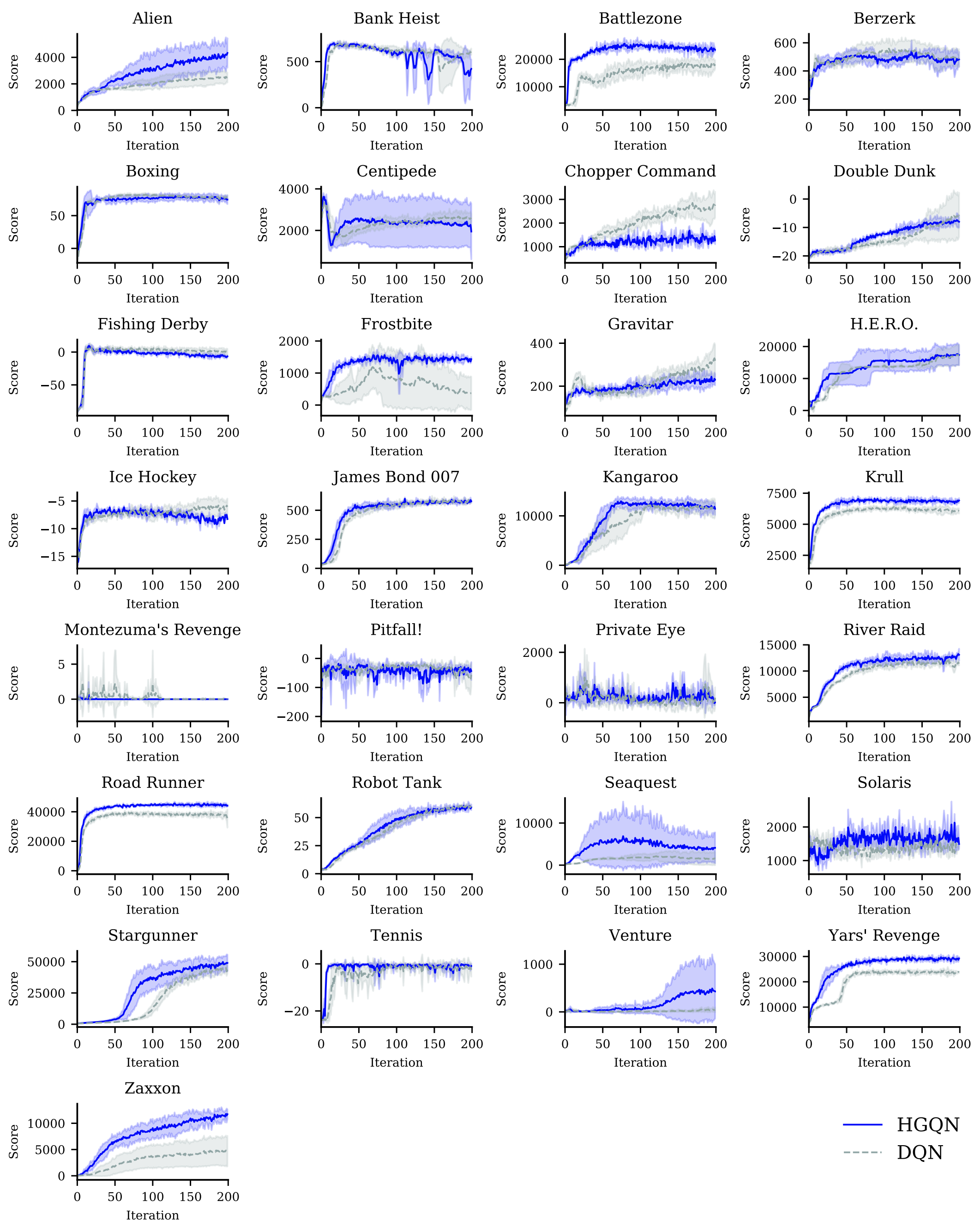


Figure B.1: Learning curves in 29 Atari 2600 games with 18 valid actions for HGQN and DQN. Shaded regions show standard deviation.

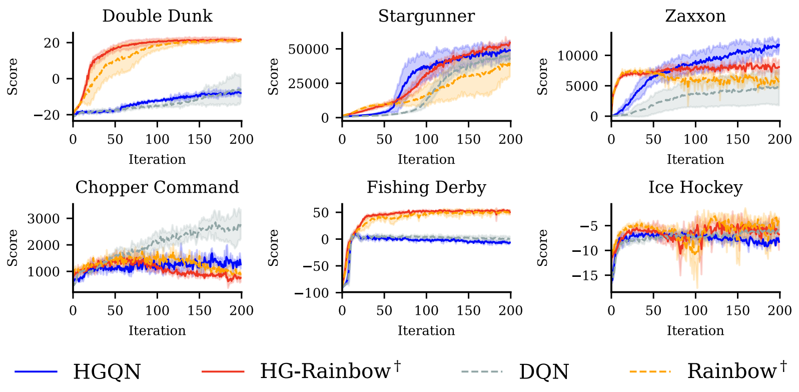


Figure B.2: Learning curves for HGQN, HG-Rainbow†, DQN, and Rainbow† in six Atari 2600 games with 18 valid actions, featuring the three best and worst-performing games from Figure 4.4a. Shaded regions indicate standard deviation.

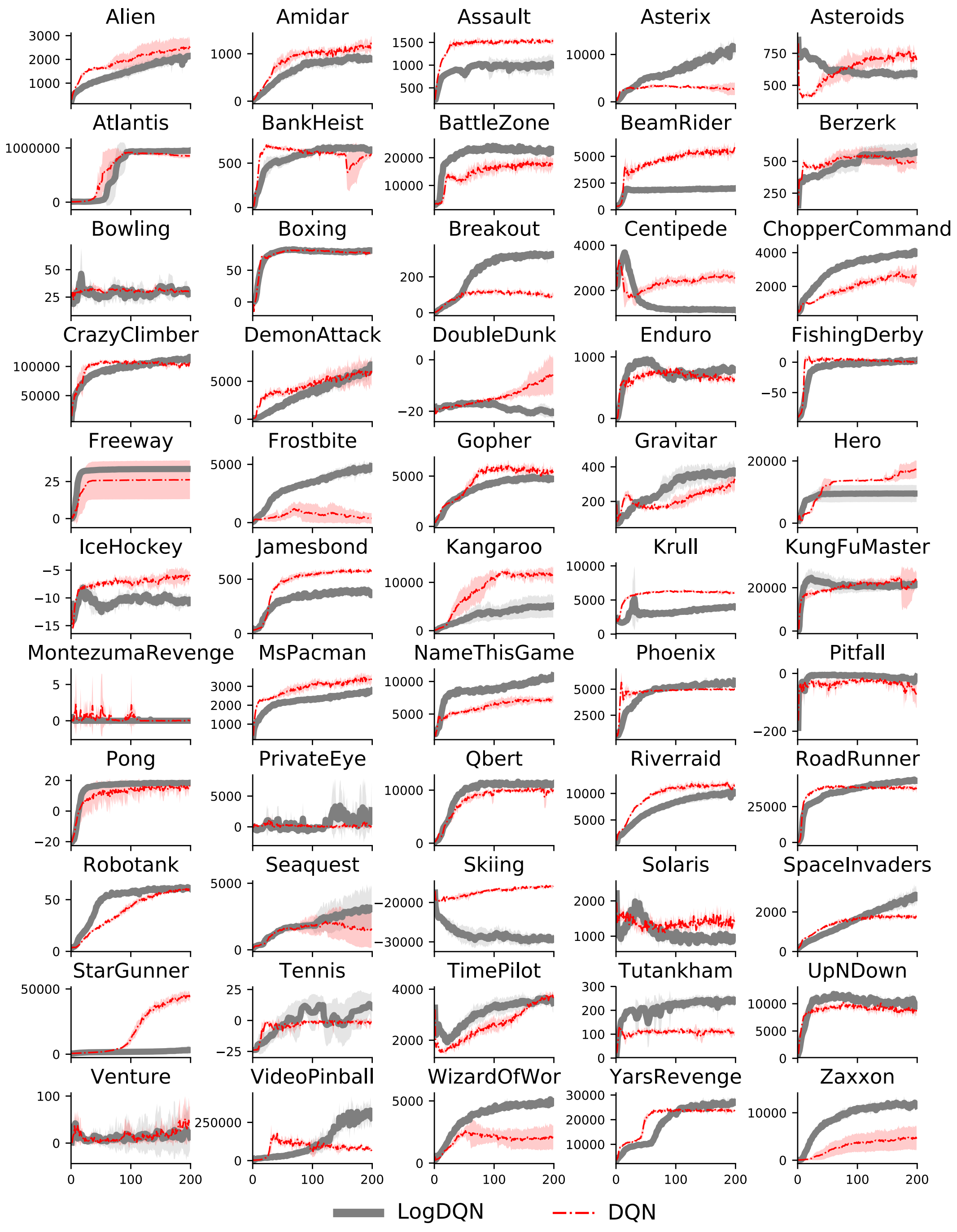


Figure B.3: Learning curves in 55 Atari 2600 games for LogDQN and DQN. Shaded regions indicate standard deviation.

Appendix C

Convergence of Logarithmic Q-Learning

In this chapter we present the proof of convergence for Logarithmic Q-Learning in its most general form.[1]

C.1 Definitions and Theorem

Our Logarithmic Q-Learning method is defined by the following equations:

$$f(x) \doteq c \ln(x + \gamma^k) + d \quad ; \quad f^{-1}(x) = e^{(x-d)/c} - \gamma^k \tag{C.1}$$

$$\begin{aligned} R_t^+ &\doteq \begin{cases} R_t & \text{if } R_t > 0 \\ 0 & \text{otherwise} \end{cases} \\ R_t^- &\doteq \begin{cases} |R_t| & \text{if } R_t < 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{C.2}$$

$$Q_t(s,a) \doteq f^{-1}\left(\widetilde{Q}_t^+(s,a)\right) - f^{-1}\left(\widetilde{Q}_t^-(s,a)\right) \tag{C.3}$$

$$\tilde{A}_{t+1} \doteq \arg\max_{a'} \left(Q_t(S_{t+1}, a')\right) \tag{C.4}$$

$$U_t^+ \doteq R_{t+1}^+ + \gamma f^{-1}\left(\widetilde{Q}_t^+(S_{t+1}, \tilde{A}_{t+1})\right) \tag{C.5}$$

$$\hat{U}_t^+ \doteq f^{-1}\left(\widetilde{Q}_t^+(S_t, A_t)\right) + \beta_{reg,t}\left(U_t^+ - f^{-1}\big(\widetilde{Q}_t^+(S_t, A_t)\big)\right) \tag{C.6}$$

[1]The proof is due to Harm van Seijen which we adopt from van Seijen et al. (2019) for a complete exposition of the Logarithmic Q-Learning algorithm in this thesis.

$$\widetilde{Q}^+_{t+1}(S_t, A_t) \doteq \widetilde{Q}^+_t(S_t, A_t) + \beta_{log,t}\left(f\big(\hat{U}^+_t\big) - \widetilde{Q}^+_t(S_t, A_t)\right) \tag{C.7}$$

$$U^-_t \doteq R^-_{t+1} + \gamma f^{-1}\left(\widetilde{Q}^-_t(S_{t+1}, \tilde{A}_{t+1})\right) \tag{C.8}$$

$$\hat{U}^-_t \doteq f^{-1}\left(\widetilde{Q}^-_t(S_t, A_t)\right) + \beta_{reg,t}\left(U^-_t - f^{-1}\big(\widetilde{Q}^-_t(S_t, A_t)\big)\right) \tag{C.9}$$

$$\widetilde{Q}^-_{t+1}(S_t, A_t) \doteq \widetilde{Q}^-_t(S_t, A_t) + \beta_{log,t}\left(f\big(\hat{U}^-_t\big) - \widetilde{Q}^-_t(S_t, A_t)\right) \tag{C.10}$$

For these equations, the following theorem holds:

Theorem 1 *Under the definitions above, Q_t converges to Q^* w.p. (with probability) 1 if the following conditions hold:*

1. $0 \leq \beta_{log,t}\,\beta_{reg,t} \leq 1$
2. $\sum_{t=0}^{\infty} \beta_{log,t}\,\beta_{reg,t} = \infty$
3. $\sum_{t=0}^{\infty} (\beta_{log,t}\,\beta_{reg,t})^2 < \infty$
4. $\lim_{t\to\infty} \beta_{reg,t} = 0$

C.2 Proof of Convergence

C.2.1 Part 1

We define $Q^+_t(s,a) \doteq f^{-1}\left(\widetilde{Q}^+_t(s,a)\right)$ and prove in Section C.2.2 that from (C.5), (C.6), and (C.7) it follows that:

$$Q^+_{t+1}(S_t, A_t) = Q^+_t(S_t, A_t) + \beta_{reg,t}\,\beta_{log,t}\left(U^+_t - Q^+_t(S_t, A_t) + c^+_t\right), \tag{C.11}$$

with c^+_t converging to zero w.p. 1 under condition 4 of our theorem (Theorem 1), and U^+_t defined as:

$$U^+_t \doteq R^+_{t+1} + \gamma\, Q^+_t(S_{t+1}, \tilde{A}_{t+1}).$$

Similarly, using definition $Q^-_t(s,a) \doteq f^{-1}\big(\widetilde{Q}^-_t(s,a)\big)$ and (C.8), (C.9), and (C.10) it follows that:

$$Q^-_{t+1}(S_t, A_t) = Q^-_t(S_t, A_t) + \beta_{reg,t}\,\beta_{log,t}\left(U^-_t - Q^-_t(S_t, A_t) + c^-_t\right), \tag{C.12}$$

with c_t^- converging to zero w.p. 1 under condition 4 of our theorem, and U_t^- defined as:

$$U_t^- \doteq R_{t+1}^- + \gamma\, Q_t^-(S_{t+1}, \tilde{A}_{t+1}).$$

It follows directly from the definitions of Q_t^+ and Q_t^- and (C.3) that:

$$Q_t(s,a) = Q_t^+(s,a) - Q_t^-(s,a). \tag{C.13}$$

Subtracting (C.12) from (C.11) and substituting this equivalence yields:

$$\begin{aligned} &Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \\ &\beta_{reg,t}\, \beta_{log,t} \left(R_{t+1}^+ - R_{t+1}^- + \gamma\, Q_t(S_{t+1}, \tilde{A}_{t+1}) - Q_t(S_t, A_t) + c_t^+ - c_t^- \right). \end{aligned} \tag{C.14}$$

From (C.2) it follows that $R_t = R_t^+ - R_t^-$. Furthermore, the following holds:

$$\begin{aligned} Q_t(S_{t+1}, \tilde{A}_{t+1}) &= Q_t\left(S_{t+1}, \arg\max_{a'} \left(Q_t(S_{t+1}, a') \right) \right) \\ &= \max_{a'} Q_t\big(S_{t+1}, a'\big) \end{aligned}$$

Using these equivalences and defining $\alpha_t \doteq \beta_{reg,t}\, \beta_{log,t}$ and $c_t \doteq c_t^+ - c_t^-$, it follows that:

$$\begin{aligned} Q_{t+1}(S_t, A_t) &= Q_t(S_t, A_t) + \\ &\alpha_t \left(R_{t+1} + \gamma \max_{a'} Q_t(S_{t+1}, a') - Q_t(S_t, A_t) + c_t \right), \end{aligned} \tag{C.15}$$

with c_t converging to zero w.p. 1 under condition 4 of our theorem. This is a noisy Q-Learning algorithm with the noise term decaying to zero. As we show in Section C.2.2, c_t is fully specified (in the positive case, and likewise in the negative case) by Q_t^+, U_t^+, and $\beta_{reg,t}$, which implies that c_t is measurable given information at time t, as required by Lemma 1 in Singh et al. (2000). Invoking that lemma, it can therefore be shown that the iterative process defined by (C.15) converges to Q_t^* if $0 \leq \alpha_t \leq 1$, $\sum_{t=0}^{\infty} \alpha_t = \infty$, and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$, as is guaranteed by the first three conditions of our theorem. The steps are similar to the proof for Theorem 1 of Singh et al. (2000), which we do not repeat here.

C.2.2 Part 2

In this section we prove that (C.11) holds under the definitions from Section C.1, $Q_t^+(s,a) \doteq f^{-1}\left(\widetilde{Q}_t^+(s,a)\right)$, and condition 4 of our theorem. The proof of (C.12) follows the same steps but instead with the "−" variants of the variables. For readability, we use β_1 for $\beta_{log,t}$ and β_2 for $\beta_{reg,t}$.

The definition of Q_t^+ implies $\widetilde{Q}_t^+(s,a) = f\left(Q_t^+(s,a)\right)$. Using these equivalences, we can rewrite (C.5), (C.6), and (C.7) in terms of Q_t:

$$f\big(Q_{t+1}^+(S_t,A_t)\big) = f\big(Q_t^+(S_t,A_t)\big) + \beta_1\Big(f\big(\hat{U}_t^+\big) - f\big(Q_t^+(S_t,A_t)\big)\Big), \quad \text{(C.16)}$$

with

$$\hat{U}_t^+ = Q_t^+(S_t,A_t) + \beta_2\Big(R_{t+1}^+ + \gamma\, Q_t^+(S_{t+1},\tilde{A}_{t+1}) - Q_t^+(S_t,A_t)\Big). \quad \text{(C.17)}$$

By applying f^{-1} to both sides of (C.16), we get:

$$Q_{t+1}^+(S_t,A_t) = f^{-1}\Big(f\big(Q_t^+(S_t,A_t)\big) + \beta_1\Big(f\big(\hat{U}_t^+\big) - f\big(Q_t^+(S_t,A_t)\big)\Big)\Big), \quad \text{(C.18)}$$

which can be rewritten as:

$$Q_{t+1}^+(S_t,A_t) = Q_t^+(S_t,A_t) + \beta_1\Big(\hat{U}_t^+ - Q_t^+(S_t,A_t)\Big) + e_t^+, \quad \text{(C.19)}$$

where e_t^+ is the error due to averaging in the logarithmic space instead of the regular space:

$$\begin{aligned} e_t^+ \doteq f^{-1}\Big(f\left(Q_t^+(S_t,A_t)\right) + \beta_1\Big(f\big(\hat{U}_t^+\big) - f\big(Q_t^+(S_t,A_t)\big)\Big)\Big) \\ - Q_t^+(S_t,A_t) - \beta_1\Big(\hat{U}_t^+ - Q_t^+(S_t,A_t)\Big). \quad \text{(C.20)} \end{aligned}$$

The key to proving (C.11), and by extension our theorem, is proving that e_t^+ goes sufficiently fast to 0. We prove this by defining a bound on $|e_t^+|$ and showing that this bound goes to zero. Figure C.1 illustrates the bound. The variables in the figure refer to the following quantities:

$$\begin{aligned} a &\rightarrow Q_t^+(S_t,A_t) \\ b &\rightarrow \hat{U}_t^+ \\ v &\rightarrow (1-\beta_1)\,a + \beta_1\, b \\ \tilde{w} &\rightarrow (1-\beta_1)f(a) + \beta_1 f(b) \\ w &\rightarrow f^{-1}(\tilde{w}) \end{aligned}$$

The error e_t^+ corresponds to:

$$e_t^+ = f^{-1}\big((1-\beta_1)\,f(a) + \beta_1\,f(b)\big) - \big((1-\beta_1)\,a + \beta_1\,b\big) = f^{-1}(\tilde{w}) - v = w - v.$$

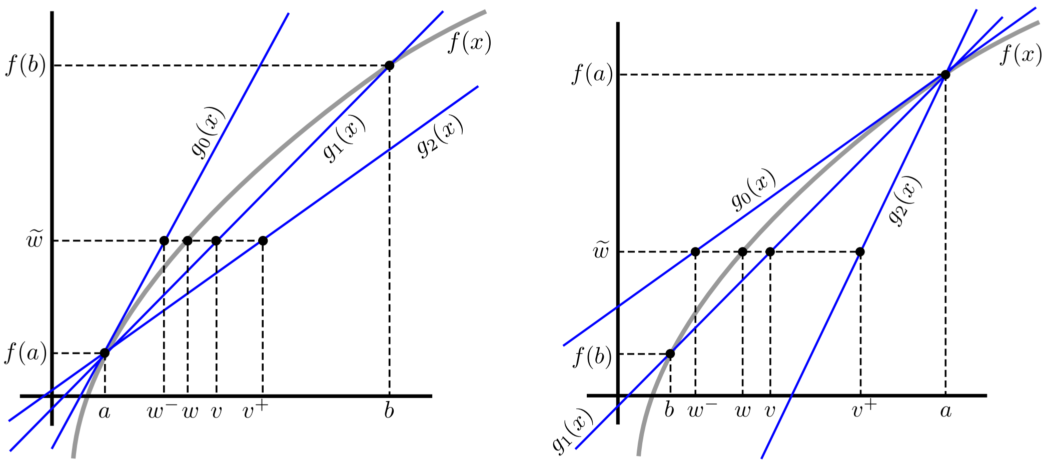


Figure C.1: Bounding the error for two cases: (left) $a < b$ and (right) $a > b$.

Note here that since f is a strictly concave function, the definitions of $\tilde{w}$ and v directly imply $\tilde{w} < f(v)$. Because f^{-1} is monotonically increasing, it follows that $w < v$, which yields $|e_t^+| = v - w$.

In both graphs of Figure C.1, besides the mapping function $f(x)$, three more functions are plotted: $g_0(x)$, $g_1(x)$, and $g_2(x)$. These three functions are all linear functions passing through the point $\big(a, f(a)\big)$. The function $g_0(x)$ has derivative $f'(a)$, while $g_2(x)$ has derivative $f'(b)$. The function $g_1(x)$ additionally passes through the point $\big(b, f(b)\big)$, giving it the derivative $\big(f(a) - f(b)\big)/(a-b)$.

As illustrated by the figure, $g_1(v) = \tilde{w}$ and $g_1^{-1}(\tilde{w}) = v$. Furthermore, for x between a and b the following holds (in both cases):

$$g_0(x) \geq f(x) \geq g_1(x) \geq g_2(x).$$

And, equivalently:

$$g_0^{-1}(x) \leq f^{-1}(x) \leq g_1^{-1}(x) \leq g_2^{-1}(x).$$

We bound $|e_t^+| = v - w$ by using a lower bound w^- for w and an upper bound v^+ for v. Specifically, we define $w^- \doteq g_0^{-1}(\tilde{w})$ and $v^+ \doteq g_2^{-1}(\tilde{w})$, and can now bound the error as follows: $|e_t^+| \leq v^+ - w^-$. Next, we compute an expression for the bound in terms of a, b, and f.

First, note that for the derivatives of g_0 and g_2 the following holds:

$$g_0'(x) = f'(a) = \frac{f(a) - \tilde{w}}{a - w^-} \quad ; \quad g_2'(x) = f'(b) = \frac{f(a) - \tilde{w}}{a - v^+}\,.$$

From this it follows that:

$$w^- = \frac{\tilde{w} - f(a)}{f'(a)} + a \quad ; \quad v^+ = \frac{\tilde{w} - f(a)}{f'(b)} + a\,.$$

Using this we rewrite our bound as:

$$\begin{aligned}
v^+ - w^- &= \frac{\tilde{w} - f(a)}{f'(b)} - \frac{\tilde{w} - f(a)}{f'(a)} \\
&= \left(\frac{1}{f'(b)} - \frac{1}{f'(a)}\right)\big(\tilde{w} - f(a)\big) \\
&= \left(\frac{1}{f'(b)} - \frac{1}{f'(a)}\right)\Big((1-\beta_1)f(a) + \beta_1 f(b) - f(a)\Big) \\
&= \beta_1\left(\frac{1}{f'(b)} - \frac{1}{f'(a)}\right)\Big(f(b) - f(a)\Big).
\end{aligned}$$

Recall that $f(x) \doteq c\,\ln(x + \gamma^k) + d$. The derivative of $f(x)$ is:

$$f'(x) = \frac{c}{x + \gamma^k}\,.$$

Substituting $f(x)$ and $f'(x)$ in the expression for the bound gives:

$$\begin{aligned}
v^+ - w^- &= \beta_1\left(\frac{b+\gamma^k}{c} - \frac{a+\gamma^k}{c}\right)\Big(c\ln(b+\gamma^k) + d - \big(c\ln(a+\gamma^k) + d\big)\Big) \\
&= \beta_1(b-a)\big(\ln(b+\gamma^k) - \ln(a+\gamma^k)\big) \\
&= \beta_1(a-b)\big(\ln(a+\gamma^k) - \ln(b+\gamma^k)\big) \\
&= \beta_1(a-b)\ln\left(\frac{a+\gamma^k}{b+\gamma^k}\right) \\
&= \beta_1(a-b)\ln\left(\frac{a-b}{b+\gamma^k} + 1\right).
\end{aligned}$$

Using the definitions of a and b, the results for the bound for e_t^+:

$$|e_t^+| \leq v^+ - w^- \leq \beta_1\Big(Q_t^+(S_t,A_t) - \hat{U}_t^+\Big)\ln\left(\frac{Q_t^+(S_t,A_t) - \hat{U}_t^+}{\hat{U}_t^+ + \gamma^k} + 1\right). \tag{C.21}$$

Definition (C.6) can be written as:

$$\hat{U}_t^+ \doteq Q_t^+(S_t,A_t) + \beta_2\Big(U_t^+ - Q_t^+(S_t,A_t)\Big) \tag{C.22}$$

yielding:

$$\begin{aligned}
Q_t^+(S_t,A_t) - \hat{U}_t^+ &= Q_t^+(S_t,A_t) - \Big(Q_t^+(S_t,A_t) + \beta_2\big(U_t^+ - Q_t^+(S_t,A_t)\big)\Big) \\
&= \beta_2\Big(Q_t^+(S_t,A_t) - U_t^+\Big).
\end{aligned}$$

Substituting this in (C.21) gives:

$$|e_t^+| \leq \beta_1 \beta_2 \left(Q_t^+(S_t, A_t) - U_t^+\right) \ln\left(\frac{\beta_2 \left(Q_t^+(S_t, A_t) - U_t^+\right)}{\hat{U}_t^+ + \gamma^k} + 1\right).$$

Let us define c_t^+ as:

$$c_t^+ \doteq \left(Q_t^+(S_t, A_t) - U_t^+\right) \ln\left(\frac{\beta_2 \left(Q_t^+(S_t, A_t) - U_t^+\right)}{\hat{U}_t^+ + \gamma^k} + 1\right).$$

Hence, $|e_t^+| \leq \beta_1 \beta_2 c_t^+$. Substituting maximum bound of $|e_t^+|$ and (C.22) in (C.19), we get:

$$Q_{t+1}^+(S_t, A_t) = Q_t^+(S_t, A_t) + \beta_1 \beta_2 \left(U_t^+ - Q_t^+(S_t, A_t) + c_t^+\right) \tag{C.23}$$

with c_t^+ going to zero if β_2 goes to zero, which concludes this part of the proof.