Policy-Value Concordance for Deep Actor-Critic Reinforcement Learning Algorithms

by

Jonas Buro
Bachelor of Science (Honours), University of Victoria, 2021

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Policy-Value Concordance for Deep Actor-Critic Reinforcement Learning Algorithms

by

Jonas Buro
Bachelor of Science (Honours), University of Victoria, 2021

Supervisory Committee

Dr. Brandon Haworth, Supervisor
(Department of Computer Science)

Dr. Teseo Schneider, Departmental Member
(Department of Computer Science)

ABSTRACT

Designing general agents to optimize sequential decision-making underneath uncertainty has long been central to artificial intelligence research. Recent advances in deep reinforcement learning (RL) have made progress in this pursuit, achieving superhuman performance in a collection of challenging and visually complex domains, in a *tabula rasa* fashion without embedding human domain knowledge. Although making progress towards designing general problem-solving agents, these methods require significant amounts of data to learn effective decision-making policies relative to humans, preventing their application to most real-world problems for which no simulator exists. It is clear that the question of how to best learn models intended for downstream purposes such as planning in this context remains unresolved. Motivated by this gap in the literature, we propose a novel learning objective for RL algorithms with deep actor-critic architectures, with the goal of further investigating the efficacy of such methods as autonomous general problem solvers. These algorithms employ artificial neural networks as parameterized policy and value functions, which guide their decision-making processes. Our approach introduces a learning signal that explicitly captures desirable properties of the policy function in terms of the value function from the perspective of a downstream reward-maximizing agent. Specifically, the signal encourages the policy to favour actions in a manner that is concordant with the relative ordering of value function estimates during training. We hypothesize that when correctly balanced with other learning objectives, RL algorithms incorporating our method will converge to comparable strength policies using less real-world data relative to their original instantiations. To empirically investigate this hypothesis, we incorporate our technique with state-of-the-art RL algorithms, ranging from simple policy gradient actor-critic methods to more complex model-based architectures, and deploy them on standard deep RL benchmark tasks, and then perform statistical analysis on their performance data.

Contents

List of Algorithms

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Dr. Brandon Haworth, for his patience and guidance during my time in graduate school.

NSERC and the Victoria Faculty of Graduate Studies, for the financial support.

My fellow academic colleagues Theodore, Nils, Leo, and Andrea, for the many hours of entertaining and stimulating company.

DEDICATION

For my family,
without whom this would have not been possible.

Science is a way of trying not to fool yourself. The first principle is that you must not fool yourself, and you are the easiest person to fool..

Richard Feynman

We have not succeeded in answering all our problems. The answers we have found only serve to raise a whole set of new questions. In some ways we feel we are as confused as ever, but we believe we are confused on a higher level and about more important things.

Earl C. Kelley

Chapter 1

Introduction

A major challenge facing artificial intelligence (AI) researchers interested in developing general problem-solving agents is equipping them with sophisticated planning capabilities. The ability to plan, or to generate a functional sequence of actions to achieve a particular goal, is a fundamental characteristic of intelligent behaviour. The act of planning implies an underlying reasoning process in which the effectiveness of potential courses of action are evaluated for their likelihood of attaining a desired result. The ability to plan was a significant development in human evolution, and is thought to have been a driving force behind the emergence of higher-order brain functions, such as language [Corballis, 2009]. Without the ability to plan, an agent is restricted to making decisions based solely on trial-and-error reasoning, severely reducing the problem domains where it can achieve satisfactory performance. Accordingly, considerable attention has been devoted towards researching computational planning methods. This thesis focuses on making progress in this domain, specifically in the subfield of AI known as reinforcement learning (RL), in which agents learn to act autonomously by distilling experiences of environments into decision-making policies. Our work concerns itself with designing an appropriate learning signal for training such agents, such that the representations they learn are both useful for downstream applications such as planning, and learned efficiently in terms of the amount of environmental interactions required to do so.

Historically, computer systems which solved challenging problems that require planning relied upon a top-down approach involving human domain knowledge and heuristics. However, as evidenced by the growing complexity of such systems, and argued convincingly in "The Bitter Lesson" [Sutton, 2019], the most promising AI methods over the past century have not been those that incorporate human knowledge, but rather those that leverage general-purpose techniques scalable alongside Moore's predicted exponential increase in available computing power [Moore, 1998]. These include forward search methods, which inform decision-making by exploring and evaluating possible future scenarios, and learning meth-

ods, which improve their performance on tasks through experience. Examples from fields like computer chess, Go, speech recognition, and computer vision consistently demonstrate that while human-centric approaches can yield short-term gains, long-term progress favors techniques that exploit computational power without relying on specific domain insights.

As such, in the pursuit of developing general problem-solving agents, modern AI researchers have increasingly adopted a data-driven and automated method called machine learning (ML). Rather than relying on hard-coded rules, ML agents learn patterns and relationships by analyzing data. RL, a subfield of ML, is a widely researched and promising method for general agent design, in which search and learning are core components. RL is inspired by the concept of reinforcement [Thorndike, 1911] in physiology, which observes that the consequences of behaviours are highly correlated with the probability of their repetition in the animal kingdom. At a very high level, RL algorithms shape the conduct of agents by transmitting reward signals conditioned on their actions within environments. RL has contributed to remarkable technological achievements in recent years, particularly after its integration with artificial neural networks. However, the research area faces significant challenges that impede its practical application to a wider set of real-world settings, particularly those for which a simulator is difficult, expensive, or impossible to source. In particular, state-of-the-art algorithms are sample inefficient, requiring significant amounts of data to learn effective action selection policies and corresponding state evaluation functions. We address this limitation by proposing a novel optimization objective for training deep RL agents that incorporate parameterized policy and value predictors. This objective quantifies a desirable relationship between these two entities, namely the degree to which they exhibit *concordance*, or how well policy predictions align with the relative ordering of the value function estimates. We conjecture that this characteristic is a robust signal for gradient-based RL optimization methods, such that its incorporation promotes convergence to useful policies in a manner that is more sample efficient than previous approaches.

This work begins with a literature review of computational planning milestones, which contextualizes the state-of-the-art computer science methods upon which we build. We then describe the technical details underlying our work, beginning with an analysis of Markov decision processes, the mathematical framework which supports many sequential decision-making algorithms. Afterwards, we introduce fundamental RL concepts, and gradually narrow the focus to the mechanisms of those algorithms which allow us to investigate our proposed objective. Following this, we describe our experimental methodology and results. We conclude with a discussion of the implications and limitations of our findings and potential future research directions.

Chapter 2

Historical Background

This chapter provides a literature review of important milestones within the field of artificial intelligence (AI) research. The following selection focuses on computational approaches to games, which were among the set of problems pioneering computing science researchers began to investigate in the middle of the 20th century. These showcase the complexity of planning in decision-making processes, provide clear problem inputs and well-defined performance measures, and provide abundant data through simulations. As such, they serve as suitable tools for computer scientists to refine their computational and machine learning techniques, which might then be applied to other decision-making problems. This literature review sets out to clarify the motivations for and the milestones of the study of game-playing AI, and by doing so, motivates and contextualizes the research conducted in this thesis.

Claude Shannon, a major contributor to the field of computer science, argued for the utility of analyzing chess in his paper *Programming a Computer for Playing Chess* [Shannon, 1950], noting that:

> ...the chess machine is an ideal one to start with, since: (1) the problem is sharply defined both in allowed operations (the moves) and in the ultimate goal (checkmate); (2) it is neither so simple as to be trivial nor too difficult for satisfactory solution; (3) chess is generally considered to require 'thinking' for skillful play; a solution of this problem will force us either to admit the possibility of a mechanized thinking or to further restrict our concept of 'thinking'; (4) the discrete structure of chess fits well into the digital nature of modern computers."

Shannon believed that the theoretical progress made through such an endeavour might enable the pursuit of more practical goals, such as designing machines capable of language translation or making strategic decisions in military operations. In the same paper, Shannon speculated about strategies which are critical for computers to play chess effectively.

He suggested that programs could “learn" heuristic evaluation functions to estimate the value of certain board states, based on previous experience. He also considered lookahead search, wherein the computer looks many moves ahead when deciding what move to play. These methods are crucial components for many game-playing AI agents, even to this day. Although never succeeding in implementing his ideas due to the hardware and software limitations of his times, his speculations were accurate: these methods would later form the basis for computer chess grandmasters.

Alan Turing also took an interest in games. In his work *Digital Computers Applied to Games* [Turing, 1953], he writes:

> Research into the techniques of programming a machine to tackle complicated problems of this type [games] may in fact lead to quite important advances, and help in serious work in business and economics—perhaps, regrettably, even in the theory of war.

He goes on to argue for the possibility of creating a program that is capable of playing chess, by demonstrating a crude procedure for doing so. By admitting its faults, and suggesting that some might take this as evidence for the statement that one cannot program a machine to play a better game than one plays oneself, he constructs a counter-argument which draws upon computers' extraordinary calculation speeds and the possibility to leverage this in learning from experience. In his words: “[i]n connexion with the question of the ability of a chess-machine to profit from experience, one can see that it would be quite possible to programme the machine to try out variations in its method of play ...and adopt the one giving the most satisfactory results. This could certainly be described as “learning,” though it is not quite representative of learning as we know it." His chess bot may have been crude, but his ideas were not, as evidenced by later grandmaster programs which drew upon the ideas he discusses.

2.1 Von Neumann's Minimax Theorem

The theoretical breakthrough underlying many combinatorial game computer champions was achieved by John von Neumann. In his 1928 work, *Zur Theorie der Gesellschaftsspiele* (translated as ‘On the Theory of Social Games’) [v. Neumann, 1928], he introduces a formalization of finite, perfect information, two-player, zero-sum games, wherein the gain of one player is the loss of the other, like in chess. His Minimax theorem from the same paper asserts that in these games, there exists a strategy for player A that maximizes his gain conditioned on player B employing a strategy to minimize that same gain. This observation

Algorithm 1 Minimax Algorithm for Game Tree with Alternating Player To Move

```
1: function minimax(node, depth, maximizingPlayer)
2: if depth = 0 or node is a terminal node then
3:    return evaluate(node)
4: end if
5: if maximizingPlayer then
6:    maxEval ← −∞
7:    for each childNode in node.children do
8:       value ← minimax(childNode, depth - 1, false)
9:       maxEval ← max(maxEval, value)
10:   end for
11:   return maxEval
12: else
13:   minEval ← +∞
14:   for each childNode in node.children do
15:      value ← minimax(childNode, depth - 1, true)
16:      minEval ← min(minEval, value)
17:   end for
18:   return minEval
19: end if
```

provides a simple strategy for such games: if player A considers the worst-case scenario in terms of his opponent's actions when selecting his own, he maximizes his chance of winning. The Minimax theorem is typically applied to so called game trees in the form of an algorithm called Minimax search (Algorithm 1). A game tree's root corresponds to the current position of the game, and its child nodes correspond to those positions which can be reached by playing a move from that position, which in turn have children which correspond to reachable positions from that position, and so on. The leaves of the tree are the final states of the game, which are evaluated typically as 1 for a player one win, 0 for a tie, and -1 for a player two win. To evaluate the quality of a move at the root, the algorithm applies a recursive approach wherein it propagates up the values from the leaves, alternating between maximizing and minimizing the potential outcomes at each level, simulating perfect adversarial play from the two players. The move selected at the root corresponds to the arc leading to either the maximum value child or the minimum value child, depending on if it is the maximizing or minimizing player's turn to act.

Although straightforward and effective, Minimax exhaustively explores the entire tree. This becomes problematic when faced with the combinatorial explosion of nodes in games with a large branching factor or a large number of moves before a terminal position, like checkers, chess, or Go. Creative enhancements to this algorithm which focus on reducing

either the height or width of the game tree during search would go on to form the basis for many computer grandmasters in classical games.

2.2 Samuel's Checkers Program

In the 1950's, a researcher at IBM by the name of Arthur Samuels, who shared Shannon's and Turing's belief that teaching computers to play games was a fruitful endeavour, set out to develop a checkers playing program. Samuels wrote a program in assembly that was a stronger player than he was on an IBM 701, whose CPU clocked in the Megahertz range and had less than 20KB of memory. His project evolved with time, and his first approach was described in his 1959 paper *Some Studies in Machine Learning Using the Game of Checkers* [Samuel, 2000], perhaps the seminal work in machine learning. Within it, he argues for - in a very similar fashion to Shannon arguing for the benefits of investigating chess - the utility of studying checkers. He notes that "[c]heckers ... was chosen because the simplicity of its rules permits greater emphasis to be placed on [machine] learning techniques." The total number of unique checkers positions is approximately 10^{20}, and each position has an average branching factor of 10. Chess, comparatively, has a much larger state space. Shannon estimated the total number of chess positions as roughly 10^{43} [Shannon, 1950]. Additionally, chess and has an average of around 35 legal moves per position. As a result of the hardware limitations of the times, Samuels decided to investigate checkers.

Samuel's method somewhat aligns with Shannon's and Turing's prior chess program designs, in that the computer plays by looking ahead and heuristically evaluating future positions, however he additionally and more importantly introduced two novel machine learning strategies. The first is a rote learning approach which is grafted upon a variation of von Neumann's Minimax search. During gameplay, encountered board positions are evaluated based on a heuristic look-ahead search. Unlike pure Minimax search, in which the search horizon is the terminal states, this heuristic search looks ahead only to a specified depth or ply. At this ply, a hard-coded heuristic evaluates the position according to a set of features, which take into consideration material imbalances and piece configurations which are deemed by Samuel to be an indication of how advantageous the position is for the program. Afterwards, the result is propagated back up the search tree, just as in Minimax. The position and the value are stored together in a table. This table is filled using data accumulated through gameplay against human opponents, self-play, and master games. Subsequent searches benefit from this, as the occurrence of cached positions at the end of the look-ahead effectively amplify the search depth, leveraging the earlier conducted search stored in the table. This produced continuous improvement in the program, which eventually played at an amateur level.

The second strategy is a general procedure for evolving or *learning* the heuristic evaluation function through experience, and is much more impactful. In his method, the features $f_1, f_2, \ldots, f_n$ of the evaluation function are still hard-coded, however, they are weighted by learnable coefficient parameters such that they can be expressed as a linear polynomial $\alpha_1 f_1 + \alpha_2 f_2 + \cdots + \alpha_n f_n$. These parameters are updated according to how well the heuristic matched the actual terminal position evaluation which was most likely to be reached from that position. In Samuel's words: "we are attempting to make the [predicted] score, calculated for the current board position, look like that calculated for the terminal board position of the chain of moves which most probably will occur during actual play. Of course, if one could develop a perfect system of this sort it would be the equivalent of always looking ahead to the end of the game. The nearer this ideal is approached, the better would be the play." Samuel's checkers program, although never attaining grandmaster level, had a major impact on the field of AI research, and his state-value approximation method is the ancestor of the method used by algorithms studied in this thesis.

Russel and Norvig, in *Artificial Intelligence: A Modern Approach* [Russell and Norvig, 2010] declare that Samuel's checkers program " ...was the first successful use of machine learning of any kind ... Samuels suggested most of the modern ideas in reinforcement learning, including temporal-difference learning and function approximation." John McCarthy, host of the 1956 Dartmouth Summer Research Project on Artificial Intelligence which arguably established AI research as a formal discipline, noted in *In Memoriam: Arthur Samuel: Pioneer in Machine Learning* [McCarthy and Feigenbaum, 1990] "[f]rom 1949 through the late 1960s, [Samuel] did the best work in making computers learn from experience." Rich Sutton and Andrew Barto, in their book *Reinforcement Learning An Introduction* [Sutton and Barto, 2018] give Samuel's project credit, writing: "Samuel's checkers-playing program was widely recognized as a significant achievement in artificial intelligence and machine learning."

2.3 McCarthy and Alpha-Beta Search

Later versions of Samuel's program, such as the one he describes in *Some Studies in Machine Learning Using the Game of Checkers. II - Recent Progress* [Samuel, 1967], employ a more sophisticated search method called alpha-beta search, proposed by John McCarthy during the Dartmouth conference. This method optimizes traditional Minimax search by considerably reducing the amount of nodes which need be evaluated in the game tree by pruning away those branches which are guaranteed not to influence the result. Intuitively, during the search process, if a position is examined which is unacceptable to the player about to make a

move, relative to the results of the positions explored thus far, she would deny her opponent making the move leading to that position, opting to make a different move earlier. Thus, the further examination of positions reachable from the unacceptable position's predecessor is unnecessary, as a better alternative exists. This improvement allows the search to spend its bounded compute budget in subtrees which are more promising. Samuel, who was the first to implement alpha-beta search (still on the IBM 701), gathered empirical data and noted that "[relative to Minimax search] alpha-beta pruning can easily reduce the work by factors of a thousand or more in real game situations" and that "there are no hazards associated with this form of pruning." Knuth et al. would later formalize the method in *An Analysis of Alpha-Beta Pruning* [Knuth and Moore, 1975], proving its correctness and deriving bounds on the number of nodes it evaluates during its search. They showed that it achieves the same result as the Minimax algorithm while requiring significantly fewer node evaluations in the best case. Specifically, in scenarios where some conditions on the game tree are fulfilled, alpha-beta search explores nodes proportional to the square root of the number explored by Minimax. This improvement allows the search to go deeper into more promising subtrees, establishing alpha-beta search as the premiere look-ahead search algorithm.

2.4 Chinook: Checkers Champion

Chinook was a computer program built to play the game of checkers, or 8x8 draughts, and in 1994 became the first program in history to be crowned human-computer world champion. The project was led by Jonathan Schaeffer, a Canadian AI researcher, who set out to study heuristic search methods in an experimental domain simpler than chess, much like Samuels did. Chinook's challenger was Marion Tinsley, an American mathematician and checkers world champion, who lost a remarkable five games between 1950 and 1992. After drawing the first six matches of the series, Tinsley forfeit due to health complications, and would unfortunately pass away months later. Chinook would go on to defend its title against Don Lafferty a year later, in a full 32 game series, winning once and drawing 31 times.

Chinook's success was due to a combination of a massive endgame database (on the order of billions of positions), a strong hand-crafted heuristic evaluation function and a deep tree search based on alpha-beta. The project as a whole was a massive software engineering project spanning multiple decades, and was at the time one of the longest running computations completed to date [Schaeffer et al., 2007]. Much of the computational resources were used for generating and verifying the integrity of the endgame database, which by 2007 had grown to store all checkers positions with ten pieces or less, which amounts to approximately 40 trillion positions. The project culminated in the "solving" of checkers, in which Schaeffer

and his team proved that the final result of a checkers game with no mistakes by either player results in a draw, a considerable achievement considering that checkers has on the order of 10^{20} possible positions. The researchers foreshadowed the shift away from human encoded knowledge in game-playing AI, noting that the incredible amount of resources and knowledge engineering required to achieve dominance in the domain of checkers may not scale to harder AI problems [Schaeffer et al., 1996]. This sentiment is echoed in the "Bitter Lesson [of AI]" [Sutton, 2019], in which it is argued that general methods that scale arbitrarily with computational power, such as search and learning, are significantly more effective than human-centric approaches within AI.

2.5 Tesauro and TD-gammon

A major milestone in machine learning was reached in 1992 when a backgammon program by the name of TD-gammon [Tesauro et al., 1995] learned to play at a strong master level without the help of any human domain knowledge and entirely through games of self-play. The game of backgammon presents an interesting challenge, as it involves a state space larger than chess and a branching factor of around 400, prohibitively restricting the use of conventional heuristic search methods, such as alpha-beta. Gerald Tesauro found success in backgammon using a technique from the field of reinforcement learning called temporal difference (TD) learning [Sutton, 1988] with a generalized nonlinear function approximator in the form of a multilayer perceptron [Rumelhart et al., 1985]. More specifically, Tesauro trained an artificial neural network to predict the win probability of encoded backgammon states using games generated by TD-gammon playing against itself. During training, the network observed a sequence of game states starting from the opening position and ending in a terminal position. At every time step, the network weights were updated to reduce the difference between its evaluation of previous turns' positions and its evaluation of the current turn's position. Given a particular state at runtime, the maximum network evaluation over all possible actions from that state was then chosen as the move to play. TD-gammon had a noticable impact on the professional backgammon scene by introducing novel opening strategies which were adopted by strong human players. Additionally, the use of neural networks in TD-gammon would become central to later RL methods in the field of game-playing AI, and beyond.

2.6 Deep Blue: Chess Champion

IBM's Deep Blue [Campbell et al., 2002] became the first computer Chess grandmaster after defeating reigning human champion Garry Kasparov in a much-publicized match in 1997. Deep Blue was a massively parallel supercomputer system which used custom hardware VLSI chips to search 50-100 million chess positions per second. The alpha-beta inspired search originated in software at a controller workstation and was then distributed to responder workstation nodes after a certain ply was reached. These workstation nodes would then continue the search, which was implemented in their chips' hardware, to a certain depth, up to 40 moves ahead in certain positions. A complex handcrafted evaluation function featuring thousands of system weighted features guided the search, which was improved automatically by analyzing grandmaster games. In addition to its enormous search capabilities, the system also had access to an endgame database containing positions with five or less on board pieces and their corresponding optimal moves, which it could use to play perfectly from. An as extensive opening book prepared by grandmasters was also compiled, effectively "squeezing" the region to which search methods need be applied to the middle game. To this day, Deep Blue's influence can be felt, with many top chess engines, such as Stockfish, still incorporating variations of alpha-beta.

2.7 AlphaGo, AlphaZero, and MuZero

Google DeepMind introduced AlphaGo [Silver et al., 2016] in 2016, which is a program designed to play the Chinese game of Go. Go, like backgammon, is considered to be an incredibly challenging task for AI owing to its enormous search space and the difficulty of devising a strong state evaluation function, thus restricting approaches based on classic heuristic search algorithms such as alpha-beta. Until AlphaGo's release, the strongest computer Go players only managed to achieve low dan-level ranks and were based on a selective lookahead search method called Monte Carlo tree search [Browne et al., 2012]. In it, the game tree is built as usual by exploring potential futures, however, unlike Minimax search variations, the maximal depth nodes of the search are evaluated by multiple simulated random-play rollouts from that position until the end of the game. The results of these many rollouts are then averaged to approximate the position's value, similar to sample-average approximation techniques. AlphaGo's team introduced a new approach towards computer Go that uses neural networks to evaluate the value of states and select moves. These networks were trained using supervised learning on human expert games as well as reinforcement learning on games of self-play. They were then combined with a Monte Carlo tree search algorithm

to select moves at runtime. AlphaGo became the first program to beat a human world Go champion, defeating Lee Sedol in 2016.

AlphaGo's successor AlphaZero [Silver et al., 2017a] learned how to play Go at a level exceeding that of AlphaGo in a completely tabula rasa fashion, without the use of human data, guidance, or domain knowledge. It was based solely on a self-play deep reinforcement learning algorithm that integrated a Monte Carlo tree search within its network's training regiment. The search process makes use of two separate components: a simulator implements the rules of the game, which are used to update the game state while traversing the search tree; and a neural network jointly predicts the corresponding policy and value of a board position. The only information which is necessary as input to the algorithm are the state transitions within the search tree, the actions available in each state, and a method to evaluate terminal nodes for win, draw, or loss. The system has demonstrated remarkable results in not only Go, but in other two-player games as well, such as chess and Shogi where it became computer champion. Many real world problems can be modeled as games, and extensions to this algorithm have been applied successfully in multiple domains, such as discovering new algorithms.

AlphaZero's succesor MuZero [Schrittwieser et al., 2020] was a historical achievement in AI, taking a significant step towards building a standalone system which can learn to solve tasks based solely on first principles. It achieved superhuman performance in a range of challenging and visually complex domains, such as Atari games, without any given knowledge of their underlying dynamics. Instead of searching within the environment's state space like in AlphaZero, MuZero builds and searches within a generated latent model. This latent model is an abstract and compact representation of the environment and its dynamics. MuZero's ability to both learn a model of its environment and use it to successfully plan demonstrates a significant advance in reinforcement learning and the pursuit of general purpose algorithms, which this research aims to contribute to.

Chapter 3

Technical Background

This chapter introduces the relevant technical background to contextualize the research done in this thesis. We first present a summary of Markov decision processes (MDPs), which provide a mathematical framework for modeling decision-making problems underneath uncertainty. We then discuss Reinforcement Learning (RL), a machine learning paradigm which takes an interactive approach to solving such problems, focusing on those methods which integrate learned policy and function approximators, which form the basis for our experiments. Next, we present an overview of Monte Carlo Tree Search (MCTS), a general planning algorithm for decision-making processes. Lastly, we describe MuZero and its derivatives, a family of state-of-the-art RL algorithms which combine these concepts to autonomously learn and plan in visually complex domains.

3.1 Markov Decision Processes

MDPs, first studied in the 1950s by Richard Bellman as extensions to Andrey Markov's stochastic processes [Markov, 2006], are to this day the *de facto* standard mathematical models for sequential decision-making problems. Such scenarios can be formulated as follows: a decision maker, or agent, observes an environment at a specified point in time. Based on this observation, and perhaps other previous observations, the agent takes an action that probabalisticly evolves the environment into a new configuration at a subsequent point in time, while also emitting to the agent a reward signal. The goal of the agent is to choose actions which maximize a predetermined performance criterion, expressed as a function of the reward. Such iterative processes might repeat indefinitely, or until some stopping condition is met, such as exceeding a time threshold or encountering a specific environment configuration. This framework was historically applied to a wide variety of problems ranging from industrial processes such as queueing and inventory control, to power system management [White,

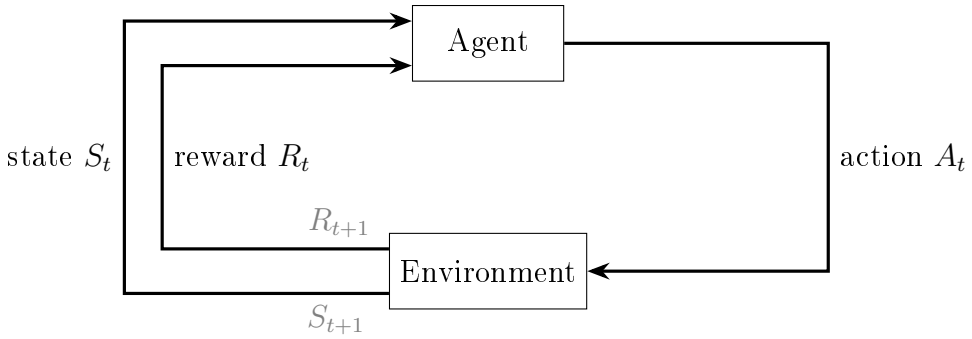

Figure 3.1: Relationship between agent and environment in a MDP

1993]. Today, MDPs and their extensions provide the mathematical foundation for sequential decision-making problems and RL. The following provides a brief introduction to MDPs. For more detailed information, see [Puterman, 2014].

3.1.1 Definition

A MDP is specified by a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma\}$:

- $\mathcal{S}$ is the set of possible environment states, which take on a very specific meaning in this context that is separate from the agent's immediate perceptions or direct sensory inputs. An environment-emitted state is a potentially heavily processed signal that carries system status information relevant to the agent's decision making. In order for a decision process to be considered a *Markov* decision process, this state signal must have a particular property called the Markov property, which is explained in the next section.

- $\mathcal{A}$ is the set containing all possible actions an agent can take in the environment.

- The state-action transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to P(\mathcal{S})$ models the probability distribution over potential successor states, given a specific state and action, allowing for the modeling of problems wherein taken actions result in non-deterministic (probabalistic) outcomes.

- The reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ describes the reward the agent receives upon transitioning to a specific state via a given state and action.

- $\gamma \in (0, 1]$ is the discount factor, which signals to the agent the importance of future rewards relative to immediate rewards.

Starting at a given initial state $S_0 \in \mathcal{S}$ at time $t = 0$ drawn from some distribution describing permissible starting states $\mathcal{D}$, the agent sequentially interacts with the environment by observing at every timestep t a state S_t, taking action A_t according to its action selection policy $\pi : \mathcal{S} \to P(\mathcal{A})$, thereby inducing state $S_{t+1} \sim \mathcal{T}(\cdot \mid S_t, A_t)$ in the environment and receiving reward $R_{t+1} = \mathcal{R}(S_t, A_t, S_{t+1})$. This iterative process gives rise to a potentially indefinite trajectory $S_0, A_0, R_1, S_1, A_1, R_2, S_2 \ldots$. The goal of the agent at any time t is to maximize a function of the future reward, called the return: $G_t : \{R_i \mid i > t\} \to \mathbb{R}$. In the simplest case, the return is the sum of future rewards: $G_t = R_{t+1} + R_{t+2} + \ldots$. A more general form of the return, which we will use in the subsequent analyis, considers future discounted rewards, $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$. This form introduces a discount factor $\gamma \in (0, 1]$ which controls the emphasis on future rewards relative to immediate reward. The goal of maximizing return can be expressed as an optimization problem, whereby a process searches over the space of all possible decision-making policies for an optimal policy π^* which maximizes the specified return.

3.1.2 Assumptions

The major assumption of the MDP framework is that the environment is Markovian, or equivalently, that every state in the environment has the Markov property. If it is true that at any time t, and for all states s, rewards r, and all possible values of past events $S_0, A_0, R_1, S_1, A_1, \ldots, R_t, S_t, A_t$

$$Pr(S_{t+1} = s, R_{t+1} = r \mid S_t, A_t) = Pr(S_{t+1} = s, R_{t+1} = r \mid S_0, A_0, \ldots, S_t, A_t),$$

then the environment is Markovian. The conditional independence of the transition dynamics with respect to the history implies that the future behavior of the environment is determined solely by the current state and action taken. Also known as the "independence of path" or "memoryless" property, the Markov property enables predictions about all future states and expected rewards by iterating the one-step dynamics of the environment, in a manner that is equally effective whether one has knowledge of only of the current state of the environment or the complete history up to the current state. Thus, this constraint on the state signal significantly simplifies reasoning about the effects of taking actions in a decision process, which is critical for designing tractable solution methods, as these then need not consider potentially unmanageable histories.

In certain environments, such a Markov state can be simply expressed as a direct sensory input gathered from the environment, such as a board position in a game of checkers, or 8x8 draughts. Within such a state is contained all the relevant information for making a

move; the previous history is inconsequential. However, in more complex environments, the state may require a more complex description, such as one that depends on previous sensory inputs. For example, in the game of Texas Hold'Em Poker, the designed state signal might include information about the betting action on previous streets, in addition to statistics of the opponents' betting proclivities during previous hands. Although in practice it is not always possible to design a perfectly Markov state, studying decision processes with this constraint in place has allowed researchers to create and understand algorithms which have been successfully applied to tasks where this constraint is not fulfilled [Sutton and Barto, 2018].

In the fully observable setting, which the subsequent analysis assumes, the agent has access to a full description of the MDP, including the state-action transition function $\mathcal{T}$, the state space $\mathcal{S}$, the action space $\mathcal{A}$, and the reward function $\mathcal{R}$. Further, these are assumed as stationary throughout. In the finite MDP setting, the time, state, and action spaces are discrete and finite.

Other frameworks which extend MDP concepts to a broader set of problems, such as partially observable MDPs (POMDPs) [Spaan, 2012] which model decision making under conditions of uncertain sensing, and stochastic games [Shapley, 1953], in which multiple agents act simultaneously, are also active research topics.

3.1.3 Analysis

In order to guide the search for a useful policy, the agent requires an indication of how useful it is to be in a given state, or take a certain action from a given state. Given a particular policy π, the state-value function $V_\pi(s)$, and the action-value function $Q_\pi(s, a)$, formalize these notions. Specifically, $V_\pi(s)$ captures the expectation of the return G_t, given that the agent is at state s at time t and acts according to policy π:

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

Similarly, $Q_\pi(s, a)$ captures the expectation of the return G_t given that the agent is at state s at time t and acts according to policy π, conditioned on action a:

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

Bellman Expectation Equations

These value functions can be expressed recursively as what are known as the Bellman expectation equations, which establish the relationship between the value of a state or state-action

pair in terms of the values of its possible successor states or state-actions, respectively. They form the basis for ways to compute, approximate, and learn value functions in MDPs, which give rise to action selection policies.

$$\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi\big[G_t \mid S_t = s\big] \\
&= \mathbb{E}_\pi\big[R_{t+1} + \gamma R_{t+2} + \cdots \mid S_t = s\big] \\
&= \mathbb{E}_\pi\big[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} \dots) \mid S_t = s\big] \\
&= \mathbb{E}_\pi\big[R_{t+1} + \gamma G_{t+1} \mid S_t = s\big] \\
&= \mathbb{E}_\pi\big[R_{t+1} \mid S_t = s\big] + \gamma \mathbb{E}_\pi\big[G_{t+1} \mid S_t = s\big] \\
&= \mathbb{E}_\pi\big[R_{t+1} \mid S_t = s\big] + \gamma \mathbb{E}_\pi\big[\mathbb{E}_\pi\big[G_{t+1} \mid S_{t+1} = s'\big] \mid S_t = s\big] \\
&= \mathbb{E}_\pi\big[R_{t+1} \mid S_t = s\big] + \gamma \mathbb{E}_\pi\big[V_\pi(s') \mid S_t = s\big] \\
&= \mathbb{E}_\pi\big[R_{t+1} + \gamma V_\pi(s') \mid S_t = s\big] \\
&= \sum_a \pi(a \mid s) \sum_{s'} \mathcal{T}(s' \mid s, a)\big[\mathcal{R}(s, a, s') + \gamma V_\pi(s')\big]
\end{aligned}$$

The analagous equation for the action-value function $Q_\pi(s, a)$ is:

$$Q_\pi(s, a) = \sum_{s'} \mathcal{T}(s' \mid s, a)\big[\mathcal{R}(s, a, s') + \gamma \sum_{a'} \pi(a' \mid s') Q_\pi(s', a')\big]$$

Belllman Optimality Equations

The above-mentioned value functions induce a partial ordering over policies: given two policies π and π', $\pi \geq \pi'$ if and only if $\forall s \in \mathcal{S}, V_\pi(s) \geq V_{\pi'}(s)$. The optimal state-value function $V_*(s)$ and optimal state-action function $Q_*(s, a)$ assign to each state or state-action pair the largest expected return achievable by any policy. An optimal policy π^* is a policy whose value functions are optimal. The Bellman optimality equations specify the necessary conditions for optimal value functions:

$$V_*(s) = \max_\pi V_\pi(s)$$

$$Q_*(s, a) = \max_\pi Q_\pi(s, a)$$

By looking one step ahead from a given state or state-action pair, these equations can be expressed in terms of one another. Given state s, to find $V_*(s)$ all possible actions $a \in \mathcal{A}$ are considered, and then the highest $Q_*(s, a)$ score is selected.

$$V_*(s) = \max_a Q_*(s, a)$$

For $Q_*(s,a)$, all possible successor states $s' \in \mathcal{S}$ that can be induced via (probabalistic) $\mathcal{T}$ given state s and action a are considered. The immediate reward upon entering s' and the discounted expected future reward of s' are weighted by the probability of $\mathcal{T}$ inducing s' given s and a.

$$Q_*(s,a) = \sum_{s'} \mathcal{T}(s' \mid s,a)\big[\mathcal{R}(s,a,s') + \gamma V_*(s')\big]$$

By combining these results, V_*, Q_* can be written into recursive forms.

$$\begin{aligned} V_*(s) &= \max_a Q_*(s,a) \\ &= \max_a \sum_{s'} \mathcal{T}(s' \mid s,a)\big[\mathcal{R}(s,a,s') + \gamma V_*(s')\big] \end{aligned}$$

$$\begin{aligned} Q_*(s,a) &= \sum_{s'} \mathcal{T}(s' \mid s,a)\big[\mathcal{R}(s,a,s') + \gamma V_*(s')\big] \\ &= \sum_{s'} \mathcal{T}(s' \mid s,a)\big[\mathcal{R}(s,a,s') + \gamma \max_{a'} Q_*(s',a')\big] \end{aligned}$$

These equations provide the insight that the optimal value functions can be computed by maximizing the expected cumulative reward over all possible future states or state-action pairs, and provide the fundamental solution approach for MDPs. Once the optimal value of each state-action is known, a policy which selects actions greedily with respect to these values is optimal:

$$\pi_*(s) = \operatorname*{argmax}_a Q_*(s,a)$$

3.1.4 Solution Methods and Computational Complexity Results

It has been shown that computing an optimal policy for large classes of MDPs is P-complete [Papadimitriou and Tsitsiklis, 1987] in the arithmetic model of computation, indicating the availability of polynomial time solutions, but suggesting that efficiently parallelizable solutions are unlikely (as it is generally thought that $P \neq NC$). This can be realized for example by linear programming (LP) [d'Epenoux, 1963] in time weakly polynomial in the number of states and actions [Littman et al., 1995]. Other solution methods, such as the dynamic programming algorithms policy and value iteration, are polynomial in the number of states, actions, and a term involving the discount factor [Littman et al., 1995]. These results seem to indicate that solving MDPs is tractable, however the number of possible states, and thus the number of state-action pairs, and thus the number of Bellman optimality equations, grows exponentially as the number of state features, or descriptors, grows linearly. This manifestation of the "curse of dimensionality" [Bellman, 1957] discourages the application of these methods to problems with higher dimensional state representations, which include

many modern problems of interest.

We conclude this section with the pseudocode for the classic MDP solution algorithm approximate value iteration (Algorithm 2), which given a finite MDP, outputs an ϵ-close optimal policy. This provides the reader with a full description of an example MDP solution method and context for a more robust approach towards sequential decision-making introduced in the next section. The main idea underlying this algorithm is repeated updating of (randomly initialized) value estimates of the states using the Bellman equations until a convergence criterion is met. Value iteration performs sweeps of the state space and updates the corresponding state-value estimates according to the Bellman operator $\mathcal{B}^*$, which is defined as:

$$\mathcal{B}^*\big(V(s)\big) \triangleq \max_a \sum_{s'} \mathcal{T}(s' \mid s, a)\big[\mathcal{R}(s, a, s') + \gamma V(s')\big]$$

It is known that $\mathcal{B}^*$ satisfies the Contraction Mapping Theorem, such that, for any V, $\lim_{i\to\infty} \mathcal{B}_i^*(V) = V_*$ [Szepesvári, 2022]. This implies that the value estimates will converge to the optimal value function V_* in the limit of sweeps. The optimal policy can be extracted by acting greedily with respect to the converged value function. In practice, the algorithm terminates when the maximum change in value estimates across all states falls below a specified threshold ϵ, in time $\tilde{O}\big(\frac{|S|^2|A|}{1-\gamma}\ln(\frac{1}{\epsilon})\big)$ [Szepesvári, 2020].

Both LP and dynamic programming methods require a full description of the environment's transition and reward dynamics, which become increasingly difficult or impossible to specify for sufficiently complex problems. The following section introduces an approach to sequential decision-making that is inspired by such iterative solution methods, but is not bound by these strong constraints.

Algorithm 2 Approximate Value Iteration

Initialize $V(s)$ arbitrarily for all states s
Set a small positive threshold ϵ for the convergence criterion
while $\Delta > \epsilon$ **do**
  $\Delta \leftarrow 0$
  for each state s **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{T}(s' \mid s, a)\big[\mathcal{R}(s, a, s') + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
end while
return policy π such that
$\pi(s) = \operatorname{argmax}_a \sum_{s'} \mathcal{T}(s' \mid s, a)\big[\mathcal{R}(s, a, s') + \gamma V(s')\big]$

3.2 Reinforcement Learning

Reinforcement learning (RL) [Sutton and Barto, 2018, Bertsekas and Tsitsiklis, 1996, Plaat, 2022] is a machine learning framework within the field of artificial intelligence (AI) that formalizes a computational approach to learning from interaction. RL investigates how an agent might distill the information contained within previous experience of a decision process into an action selection strategy for maximizing future return within it. This framework is adjacent to but conceptually distinct from another machine learning framework *supervised learning*, wherein learners receive instructive or supervisory feedback from labeled data to learn relationships between inputs and outputs. Although some modern RL algorithms incorporate supervised learning as subroutines, RL is fundamentally based on evaluative feedback in the form of delayed reward signals. This enables learning in interactive settings where labeled data is impractical to source.

This type of experience-driven learning is closely related to the mathematical discipline of optimal control theory [Todorov, 2006], shares similarities with human learning [Nasser et al., 2017, Lee et al., 2012, Glimcher, 2011, Kahneman, 2011], and has been theorized to be an integral component of artificial general and super intelligence [Turing, 1950, Silver et al., 2021, Schmidhuber, 2007, Hutter, 2005, Hutter et al., 2024]. RL is predicated on the reward hypothesis, posited by RL pioneer Rich Sutton, which states that "all what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward)" [Sutton, 2004]. Together with McCarthy's claim that "intelligence is the computational part of the ability to achieve goals in the world," [McCarthy, 1998] a sufficiency for the study of RL is offered, as taken together, these two claims imply that solving RL is sufficient for building generally intelligent agents [Bowling et al., 2023].

In mathematical analyses of this framework, the agent-environment interfaces are canonically represented as MDPs, which reduce the problem of RL to the relationship between three signals: one which represents the choices the learner or agent makes (actions), one which represents the context within which these are made (states), and one which provides evaluative feedback to the learner (rewards) [Sutton and Barto, 2018]. In this context, RL can be viewed as an attempt to overcome the limitations of traditional MDP planning methods, namely their inability to address the "curse of dimensionality" which arises from growing state and action spaces, and the requirement of a fully-specified environment model. State-of-the-art RL incorporates function approximation techniques in order to address the dimensionality challenge, and in lieu of an a priori model instead relies on state-action-next state-reward samples gathered through interaction with the environment for policy optimization.

The RL framework does not presuppose an a priori specified description of the environment's dynamics, and as a result, at least in principle, offers flexibility in terms of the types of problems that it can address. However, there exist a number of conceptual and algorithmic challenges which prevent its widespread application to real-world scenarios [Dulac-Arnold et al., 2021]. Since the underlying environment's dyanmics are unknown, while the RL agent is learning how to act it is faced with balancing between exploring unfamiliar regions which might potentially yield high reward, and exploiting those regions which have done so during previous interactions. This *exploration-exploitation dilemma* arises from the fact that pursuing either exploration or exploitation exclusively will result in suboptimal performance, thus a suitable learning strategy must specify a trade-off between the two. Finding the right balance becomes increasingly difficult in situations where all experience must be gathered in the real world as opposed to within a simulation, as unconstrained exploration may lead to data acquisition failures, e.g. a *tabula rasa* agent totals the vehicle it is learning to drive. Another challenge is that of *temporal credit assignment*. Deciding whether an action is preferential may not be possible immediately after taking it, as the effects of the action with respect to the expected return may be delayed. This problem is exacerbated in environments which emit rewards sparsely, as the information gained by the agent per experience sample is low. Additionally, the state and action spaces in many real-world problems can be very large, resulting in a combinatorial explosion when attempting to explore all possible actions from every possible state. To combat this, current RL algorithms incorporate function approximators, typically articial neural networks (ANNs). Although enabling learning in problems with high dimensional input, such as Atari, ANNs have limited theoretical convergence guarantees, particularly in non-linear optimization landscapes, and require significant amounts of data. This results in state-of-the-art RL algorithms showcasing poor sample efficiency when learning tasks relative to humans, although steady progress is being made in this regard. In addition, many RL algorithms lack general applicability and require extensive problem-specific hyperparameter tuning, discouraging their use by non-specialists.

Despite these hurdles, RL has been successfully deployed in a variety of complex real-world scenarios, e.g. learning robotic control tasks [Gu et al., 2017], fine-tuning large language models [Ouyang et al., 2022], devising novel sorting algorithms [Mankowitz et al., 2023], managing nuclear reactors [Degrave et al., 2022], designing TPU chip layouts [Mirhoseini et al., 2021] and creating powerful agents for games like Chess, Shogi, Go [Silver et al., 2016, Silver et al., 2017a, Schrittwieser et al., 2020], Texas Hold 'Em [Brown and Sandholm, 2019], Atari [Mnih et al., 2013, Mnih et al., 2015, Hafner et al., 2023, Ye et al., 2021], Dota 2 [OpenAI et al., 2019], StarCraft 2 [Vinyals et al., 2019], and Stratego [Pérolat et al., 2022]. With the advent of deep learning methods [LeCun et al., 2015, Goodfellow et al., 2016] and

open source software libraries [Paszke et al., 2019, Frostig et al., 2018, Abadi et al., 2016], interest in RL has experienced rapid growth in recent years, as evidenced by the sharp rise in the number of annually published papers indexed by Google Scholar [Henderson et al., 2018]. In order to facilitate this growth, and to more efficiently prototype and test RL algorithms, RL practitioners have devised a variety of software libraries [Hill et al., 2018, Liang et al., 2018, Niu et al., 2024] and benchmarks that are designed to test a wide array of reasoning capabilities. These include digitized versions of classic control theory problems [Osband et al., 2020, Towers et al., 2023], classic board and card games [Lanctot et al., 2019], Atari [Bellemare et al., 2013, Machado et al., 2018], sophisticated continuous control problems in physics-based simulations [Tassa et al., 2018, Todorov et al., 2012], and carefully crafted experiments that test agents against desired competencies [Osband et al., 2020]. In the literature, progress is measured and driven in part by comparing algorithms on subsets of these benchmarks, with the title "state-of-the-art" given to those which outperform competitors.

The remainder of this section gives an introduction to fundamental RL algorithms which we incorporate into the work done in this thesis. The RL landscape is diverse and features many different design choices about what to learn and how to learn it. A definitive taxonomy is a non-trivial undertaking and falls outside of the scope of this thesis. However, a brief discussion will help to locate our work within the literature. More detailed information can be found in [Sutton and Barto, 2018, Szepesvári, 2022, Silver, 2015, Abbeel, 2020].

3.2.1 Model-free RL Overview

The main distinction within RL algorithms is whether or not an algorithm has access to a learned or provided model of the environment that enables planning. Here, by model we mean any representation that explicitly encodes knowledge about the structure of the environment or task. The major advantage such a model provides an agent is the ability for counterfactual reasoning or "imagination," whereby potential future trajectories are evaluated for their efficacy to achieve a desired goal. Those algorithms which use a model for policy optimization are model-based RL (MBRL) algorithms and those who do not are model-free RL (MFRL) algorithms. While there do exist hybrid methods which incorporate both approaches, this distinction gives rise to a broad categorization.

MFRL [Otto, 2021] finds a basis in psychologist Edward Thorndike's "Law of Effect" [Thorndike, 1911], which captures the observation that animals exposed to situations that produce a satisfying effect become more likely to seek out that situation again, and vice versa. In MFRL, the agent takes initially random actions in an environment, observes the reward it gets, and uses that information directly to guide its future actions by distilling it into value

function estimates (value-based or critic MFRL), policy estimates (policy-based or actor MFRL), or a combination of both (actor-critic MFRL) . This type of "primitive" trial-and-error learning corresponds to what psychologists term *habitual* control of learned behaviour patterns [Dickinson, 1985], as it does not employ a counterfactual reasoning process and is entirely reactive to environmental stimuli. This approach towards learning from interaction is observed in intelligent beings such as humans [Haith and Krakauer, 2013, Kahneman, 2011], however as a standalone learning mechanism it is limited both from a functional and computational standpoint. In MFRL, the gathered experience is consumed solely for reward-driven policy optimization and is not otherwise incorporated into any persistent description of the environment. In addition to preventing reasoning about potential futures, this framework trades off simplicity for inflexibility in the sense that any change to the environment's incentive structure requires a complete retooling of the agent's response. Nevertheless, MFRL methods were the first RL methods to perform at or exceeding human ability in complex sequential decision-making tasks [Mnih et al., 2015, OpenAI et al., 2019, Vinyals et al., 2019] during the ascent of deep learning [LeCun et al., 2015].

Although a complete investigation of MFRL methods falls outside the scope of this thesis, we introduce some of the relevant ideas and algorithms for understanding the experiments conducted in this work. The reader interested in MFRL may find John Schulman's work [Schulman et al., 2017] useful, as it underlies many modern MFRL approaches.

3.2.2 Value-based MFRL

Value-based methods derive agent policies indirectly by learning to approximate either the state-value function $V(s)$ or the action-value function $Q(s, a)$. With these estimates in hand, a policy is then derived from them by selecting those actions which lead to maximal value states. The simplest of these methods, Q-Learning, which we describe next, is the foundation for many modern deep RL algorithms which use neural networks to approximate the action-value function, such as Deep Q-Networks (DQN) [Mnih et al., 2013]. The following discussion highlights important concepts within these methods, which are found throughout the literature and are relevant for understanding the experiments conducted in this thesis.

Q-Learning

The classic value-based MFRL algorithm tabular one-step Q-learning [Watkins, 1989] (Algorithm 3) derives the agent's policy from value estimates of state conditioned actions, which are stored in a two dimensional array in memory. Q-learning implements a common RL algorithm design pattern called *generalized policy iteration* [Sutton and Barto, 2018]. Within

it, two processes, one which evaluates the agent's current policy, and one which improves it, alternate or interleave in order to improve the agent's performance. The *policy evaluation* process considers experience gathered by the agent's current policy and estimates its corresponding state-value or action-value function. In the *policy improvement* process, the new value function estimates are used to improve the agent's policy. In theory, this alternation of processes continues until these processes stabilize; that is, once the value functions and policy converge to their respective optima, which is the point at which the Bellman optimality equations hold for all states. In practice, due to resource constraints, this process is terminated early, for example when the performance of the agent no longer improves significantly.

In Q-learning, the evaluation step uses gathered experience to update a state-action value estimate by applying an update rule based on the Bellman equations: given a sample of experience consisting of a state s, an action a, a reward r, a next state s', a step size parameter α and discount factor γ, the state-action value estimate $Q(s,a)$ is updated according to

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Bigg(\overbrace{\underbrace{r + \gamma \max_{a'} Q(s',a')}_{\text{TD-target}} - \underbrace{Q(s,a)}_{\text{current approximation}}}^{\text{TD-error}}\Bigg).$$

This update rule is an application of a central concept within RL called *temporal difference* (TD) *learning* [Sutton, 1988]. TD learning measures the discrepancy between the predicted value of a state or state-action pair and a target value consisting of the immediate reward obtained and the expected value of future states or state-actions. This so called TD-error quantifies how surprised the agent is when real outcomes differ from its estimates. If the the outcome of taking an action from a particular state provides the agent with more reward than anticiptated, it will increase its estimate of the corresponding state-action pair, and vice versa. This concept of using estimates to improve other estimates is known as *bootstrapping*. In one step Q-Learning, state-action value estimates are bootstrapped using estimates of state-action values that are one step into the future. This method can be extended to consider estimates at larger temporal distances, trading off an increase in bias and the potential for compounding errors in exchange for the inclusion of longer-term consequences. In Q-Learning's policy improvement step, the algorithm uses the new Q-values from the evaluation step to update its policy. It then samples an action from the policy via an exploratory strategy, and then applies it in the real environment. One such exploratory strategy is ϵ-greedy. Given a state s, ϵ-greedy selects the action a with the highest $Q(s,a)$ score with probability $1-\epsilon$ and a random action with probability ϵ. This ensures continual exploration of the environment during the learning process, guaranteeing that in the limit

Algorithm 3 Online One-Step Q-Learning for N Finite MDP Trajectories

Initialize $Q(s, a)$ arbitrarilly for all s, a.
for N times **do**
  Initialize starting state s
  while s is not terminal **do**
    Choose an action a from policy derived from Q-values using an exploration strategy (e.g., ϵ-greedy).
    Take action a, observe the reward r and the next state s'
    Update the Q-value for the current state-action pair:
      $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    Update the current state: $s \leftarrow s'$
  end while
end for

each action in every state is sampled an infinite amount of times. As a result, Q-learning is proven to converge to the optimal action-values in situations where these are represented discretely [Watkins and Dayan, 1992].

3.2.3 Policy Gradient MFRL

Policy gradient methods are a type of policy search (PS) method for finding (near) optimal parameters for a parameterized action-selection policy with regards to some performance measure, such as the expected discounted future reward. Direct PS methods are a type of "black box" zero-order optimization method, where only the results of policy application influence the search for an optimal set of parameters, such as in random search or population based methods. In contrast, first-order policy gradient methods optimize the policy by using the gradient of the expected reward with respect to the policy parameters. Compared to the value-based methods explained above, policy gradient methods take a more direct approach to agent learning by directly optimizing the policy. Instead of learning an intermittent value function and deriving from it a policy, policy gradient methods explicitly store a θ-parameterized policy representation $\pi_\theta(\cdot|s)$ as a neural network, and then optimize θ directly through gradient ascent informed by trajectory data from the environment. More specifically, given some performance measure $J(\theta)$ on data gathered through π_θ, the goal is to update the parameters θ approximately in proportion to its gradient. The following section introduces the policy gradient theorem, which is the foundation for these algorithms, some of which we use in our experiments.

Policy Gradient Theorem

The policy gradient theorem [Sutton et al., 1999, Abbeel, 2020], provides a way to compute the gradient of the expected return with respect to the policy parameters while avoiding differentiating through the unknown dynamics of the environment. Given a policy $\pi_\theta(\cdot|s)$, the expected return $J(\theta)$ is the expected value of the cumulative reward $R(\tau)$ over trajectories $\tau = (s_0, a_0, r_1, s_1, \ldots, r_T, s_T)$ generated through actions selected by π_θ. Since these trajectories are dependent on the environment's dynamics, which are assumed to be unknown to the agent, the gradient $\nabla_\theta J(\theta)$ cannot be calculated analytically. The following mathematical derivation shows that a sampling based approach can be used to estimate $\nabla_\theta J(\theta)$:

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\big[R(\tau)\big] \\
&= \nabla_\theta \sum_\tau Pr(\tau \mid \theta) R(\tau) \\
&= \sum_\tau \nabla_\theta Pr(\tau \mid \theta) R(\tau) \\
&= \sum_\tau \frac{Pr(\tau \mid \theta)}{Pr(\tau \mid \theta)} \nabla_\theta Pr(\tau \mid \theta) R(\tau) \\
&= \sum_\tau Pr(\tau \mid \theta) \frac{\nabla_\theta Pr(\tau \mid \theta)}{Pr(\tau \mid \theta)} R(\tau) \\
&= \sum_\tau Pr(\tau \mid \theta) \nabla_\theta \log Pr(\tau \mid \theta) R(\tau) \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\big[\nabla_\theta \log Pr(\tau \mid \theta) R(\tau)\big]
\end{aligned}
$$

This expression can be simplified further by temporally decomposing the path into states and actions:

$$
\begin{aligned}
\nabla_\theta \log Pr(\tau \mid \theta) &= \nabla_\theta \log \left[\mu(s_0) \prod_{t=0}^{T} \pi_\theta(a_t \mid s_t) \mathcal{T}(s_{t+1} \mid s_t, a_t)\right] \\
&= \nabla_\theta \left[\log \mu(s_0) + \sum_{t=0}^{T} \log \pi_\theta(a_t \mid s_t) + \sum_{t=0}^{T} \log \mathcal{T}(s_{t+1} \mid s_t, a_t)\right] \\
&= \nabla_\theta \sum_{t=0}^{T} \log \pi_\theta(a_t \mid s_t) \\
&= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)
\end{aligned}
$$

where $\mu(s_0)$ is the distribution over initial states. What is important to note here is that since the environment dynamics and intital state distribution do not depend on the param-

eterization of the policy, their gradient with respect to θ is 0 and can be ignored in the calculation of the gradient. The following result shows how to arrive at an unbiased estimate to the gradient $\nabla_\theta J(\theta)$ by accumulating the gradients of the log probabilities of the actions taken in each state, weighted by the return of the trajectory:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log Pr(\tau \mid \theta) R(\tau)\right] \\ &\approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log Pr(\tau^{(i)} \mid \theta) R(\tau^{(i)}) \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) R(\tau^{(i)})\end{aligned}$$

This leads directly to the simplest policy gradient algorithm REINFORCE.

REINFORCE

The REINFORCE algorithm [Williams, 1992] (Algorithm 4), also known as Monte-Carlo policy gradient, applies the policy gradient theorem to estimate the gradient of the expected return with respect to the policy parameters. In REINFORCE, for each episode, the agent collects a trajectory of states, actions, and rewards. After the episode ends, the agent updates the policy parameters based on the total rewards received during that trajectory. Specifically, the policy parameters are updated by applying gradient ascent using the return from each episode to scale the gradient of the log-probability of the action taken. This update is based on the idea that actions leading to higher rewards should be more likely, while actions leading to lower rewards should be less likely. REINFORCE is the ancestor of more refined policy gradient methods, such as those involving an additional value function network that helps deal with the variance in episodic returns, which we will consider in the next section.

3.2.4 Actor-Critic MFRL

Actor-critic methods combine aspects of both value-based and policy-based(gradient) approaches. Variants of these methods simultaneously estimate the parameters of two structures, called the actor and the critic. The actor represents the standard RL policy, and the critic represents either the state or action-value function, and both are normally represented by neural networks. The actions selected by the actor are evaluated by the critic, typically according to their temporal difference error. These evaluations are then incorporated into the actor's parameter update rule. This additional signal helps reduce the variance in

Algorithm 4 Monte-Carlo REINFORCE

Initialize policy parameters θ arbitrarily
for ever **do**
  Initialize the starting state s_0
  Sample a trajectory $\tau = (s_0, a_0, r_1, s_1, \ldots, s_T)$ by interacting with the environment using policy $\pi_\theta(\cdot|s)$
  Calculate return $R(\tau) \leftarrow \sum_{t=0}^{T} r_t$
  for each step t in the episode **do**
    Update the policy parameters:
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)$
  end for
end for

the policy gradient estimates, by providing a more stable estimate of an action's 'goodness' relative to the sum of rewards of noisy trajectory rollouts, such as in Monte-Carlo policy gradient methods. The following section explains the actor-critic algorithms which we use in our experiments.

Advantage Actor-Critic

A major drawback of Monte-Carlo policy gradient is that the gradient estimates suffer from high variance, as the rollouts may vary substantially due to randomness in the policy used to generate them in combination with the stochasticity of the environment dynamics. In REINFORCE, the policy's parameters are updated based on the total return from each episode at every timestep t. The problem with this, besides the high variance in the return, is that the reward gathered previous to an action a_t is not due to that action, but to previously made decisions. REINFORCE with *reward-to-go* policy gradient scaling, rectifies this credit assignment and reduces the return variance by considering only the rewards gathered *after* a_t, $R_t = \sum_{i=t}^{T} r_t$, when updating the policy parameters. Advantage Actor-Critic (A2C) [Mnih et al., 2016] (Algorithm 5) further reduces the variance of the gradient estimates by subtracting a baseline function of the state from the return. When the subtracted state-dependent baseline is the approximate value function V, the scaling term becomes $R_t - V(s_t)$, which is an estimate of the *advantage* of taking a_t at s_t compared to all other actions: $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$. The advantage function helps reduce the variance in gradient estimates by comparing the value of an action to the average value of all actions in the same state, as approximated through previous rollouts. A variation to using the reward-to-go to compute the advantage is to estimate it using a one step bootstrap estimate using the value function: $A(s_t, a_t) \approx r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$. This allows for learning from partial trajectories and further trades off variance for bias. Simultaneously to policy learning, the

Algorithm 5 Online Advantage Actor-Critic (A2C)

Initialize policy parameters θ and value function parameters ϕ arbitrarily
for number of iterations **do**
  Initialize the starting state s_0
  for each step t in the episode **do**
    Choose action a_t from policy $\pi_\theta(\cdot|s_t)$.
    Take action a_t, observe reward r_t and next state s_{t+1}
    Compute the advantage $A(s_t, a_t) = r_t + \gamma V(s_{t+1}) - V(s_t)$
    Update the policy parameters (actor):
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$
    Update the value function parameters (critic):
      $\phi \leftarrow \phi + \beta(r_t + \gamma V(s_{t+1}) - V(s_t)) \nabla_\phi V(s_t)$.
  end for
end for

critic is updated using a TD learning rule to minimize the difference between the predicted value $V(s_t)$ and the observed return with value function bootstrap, similar to Q-learning.

Proximal Policy Optimization

An issue with approaches like REINFORCE and A2C is that they sometimes make disastrously large policy updates, driving future generated trajectories into lower reward areas of the state space, which in turn leads to worsening of the policy in future update steps. This is unrecoverable, as there is no mechanism by which to 'correct' this error, such as in supervised learning where future examples can rectify such bold adjustments. Building upon previous approaches which include a hard constraint to limit the difference between the old and new policy during policy updates [Schulman et al., 2015], proximal policy optimization (PPO) [Schulman et al., 2017, Bick, 2021] (Algorithm 6) encourages the updated policy to stay within a proximity of the old policy after every parameter update step. This is achieved by introducing a surrogate policy objective function that is aligned with the original policy gradient objective, but additionally includes a soft constraint based on the ratio between the policy used to collect the data, and the most recent policy during the policy optimization phase of the algorithm. Specifically, PPO optimizes a clipped surrogate policy objective function:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A(s_t, a_t), \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right) A(s_t, a_t)\right)\right]$$

Here, the objective is expressed as an expectation over a subset or minibatch of gathered samples. $\pi_\theta(a_t|s_t)$ is the probability of taking action a_t under the current policy parameters

θ, and $\pi_{\theta_{\text{old}}}(a_t|s_t)$ is the probability under the policy used to collect the data during the rollout phase. The ratio between the two is computationally cheap approximation of how divergent the old and new policy are. The term $A(s_t, a_t)$ is the advantage estimate, which measures how much better or worse the action a_t is compared to the expected value of all actions in state s_t. The clipping function $\text{clip}(\cdot, 1-\epsilon, 1+\epsilon)$ constrains the output into into certain range controlled by ϵ. The key idea to understanding why this is supposed to prevent large policy updates is that when the probability ratio falls outside the range $[1-\epsilon, 1+\epsilon]$ and is clipped, the gradient with respect to those samples becomes zero. Consequently, those samples which pass through the clipping function and selected by the *min* function do not contribute to the policy gradient estimate. As the optimizer aims to optimize the overall objective, the process will tend to reduce the number of samples that fall outside the clipping range, encouraging smaller, more controlled policy updates.

In addition to this clipped surrogate policy gradient objective, PPO incorporates a value function loss term to update the critic, which can be architectured as a head of a common network shared with the policy or a different network altogether. For simplicity, we express the value function as being parameterized also by θ. The loss term consists of the squared-error between the value function estimate and an n-step bootstrapped discounted sum of future rewards:

$$L^{VF}(\theta) = \mathbb{E}_t\left[\left(V_\theta(s_t) - \sum_{i=t}^{n} \gamma^{i-t} r_t - V_\theta(s_{t+n})\right)^2\right]$$

The final loss term in PPO's optimization objective is a term which is based on the Shannon entropy [Shannon, 1948] of the policy, which is a measure of the policy's stochasticity. The entropy term is added to the objective to encourage the policy to be more exploratory. In the discrete action setting with A actions, with a slight abuse of notation, the entropy is given by:

$$L^{SE}(\theta) = \mathbb{E}_t\left[-\sum_{a \in A|s_t} \pi_\theta(a \mid s_t) \log \pi_\theta(a \mid s_t)\right]$$

The final objective function for PPO is a weighted sum of the clipped surrogate policy objective, the value function loss, and the entropy loss, with weighting coefficients set typically at $c_1 = 1, c_2 = 0.01$:

$$L^{PPO}(\theta) = L^{CLIP}(\theta) + c_1 L^{VF}(\theta) + c_2 L^{SE}(\theta) \tag{3.1}$$

Before ending our brief description of PPO, it should also be mentioned that the training regiment in PPO also allows for multiple epochs of optimization using minibatches of data from the same trajectories, and can be done in a distributed fashion with multiple work-

Algorithm 6 Single Worker Proximal Policy Optimization (PPO)

```
Initialize network parameters θ
for number of iterations do
    Collect trajectories using policy π_θ
    Compute advantage estimates for all states
    for number of update epochs do
        Sample mini-batches of data from the trajectories
        Optimize L^PPO with respect to θ (Equation (3.1)) via mini-batches
    end for
end for
```

ers gathering data from identical environments. For a more detailed explanation, see the conference paper [Schulman et al., 2017].

Soft Actor-Critic

Soft Actor-Critic (SAC) [Haarnoja et al., 2018a] (Algorithm 7) is an actor-critic algorithm which learns a stochastic policy like PPO and A2C, but with a different objective function that results from a reframing of the standard RL goal of solely maximizing future return, to a new goal which accounts for the entropy of the agent's policy. This is known as the maximum-entropy RL framework, wherein the goal is to maximize a trade-off between the expected return and the entropy in order to avoid locally optimal solutions and ensure continual exploration during training [Williams and Peng, 1991]. In this context, the maximum-entropy optimal policy which SAC aims to learn is defined as:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t \Big(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right) \Big)\right]$$

where H corresponds to the Shannon entropy and α is a temperature parameter that controls the trade-off between the return and the entropy terms. In essence, this directly addresses the exploration-exploitation issue which is central to RL algorithm design by attemping to learn a reward-maximizing policy while maintaining as high stochasticity as possible.

SAC implements a 'soft' (entropy considerate) version of the generalized policy iteration design pattern, where alternating policy evaluation and policy improvement processes work together to find a maximum-entropy optimal policy. In the policy evaluation step, the value function of the current policy is calculated. The soft value function V in the maximum-entropy RL framework is:

$$V(s_t) \triangleq \mathbb{E}_{a_t\sim\pi}\Big[Q(s_t, a_t) - \alpha \log(\pi(a_t|s_t))\Big]$$

This formulation gives rise to the soft Bellman backup operator $\mathcal{B}$:

$$\mathcal{B}^{\pi}Q(s_t, a_t) \triangleq \mathbb{E}_{s_{t+1}\sim\mathcal{T}(\cdot|s_t,a_t)}\left[R(s_t, a_t, s_{t+1}) + \gamma V(s_{t+1})\right]$$

The soft Q function for a policy π can be determined in the limit process of soft Bellman backup operator applications for any starting Q in the tabular case [Haarnoja et al., 2018b]:

$$Q^{\pi}(s_t, a_t) = \lim_{k\to\infty} \mathcal{B}_k^{\pi} Q(s_t, a_t)$$

With the Q function in hand, a policy improvement step can be taken. The goal is to select a new policy that is greedy with respect to an exponentiated 'probability distribution-like' vector of the newly calculated Q function, such that the resultant policy is closer to the optimal maximum-entropy policy. For a single policy improvement step, in the soft policy iteration framework the policy at every timestep t is updated according to:

$$\pi(\cdot \mid s_t) = \operatorname*{argmin}_{\pi'\in\Pi} D_{\mathrm{KL}}\left(\pi'(\cdot \mid s_t) \,\middle\|\, \frac{\exp\left(\frac{1}{\alpha}Q^{\pi}(s_t, \cdot)\right)}{Z^{\pi}(s_t)}\right)$$

where $D_{\mathrm{KL}}(p, q) = \sum_a p(a) \log \frac{p(a)}{q(a)}$ is the Kullback-Leibler (KL) divergence [Kullback and Leibler, 1951], a measure of the difference between probability distributions p, q, which is zero if the two are identical, and Z is the partition function which normalizes the exponentiated Q-values. In order for this to be computationally tractable, the set of potential new policies needs to be restricted to a subset Π of all possible policies, for example those which can be parameterized as Gaussians. As such, after the KL divergence has been calculated, the result is projected into the realizable set Π.

This soft policy iteration design pattern converges to the optimal maximum-entropy policy in the limit of policy evaluation and policy improvement steps, but only in the tabular case. In more complex scenarios requiring function approximation to generalize over states and actions, such as when these are continuous, it is computationally expensive to run either of the improvement or evaluation steps to convergence. In this case, these processes are approximated by stochastic gradient descent using sample approximations, with samples gathered from the environment and stored in a replay buffer during a data gathering phase. To perform these optimizations, the most recent version of SAC uses two neural network types: a policy network π_θ (actor), and a action-value network Q_ϕ (critic).

The critic is trained to minimize the soft Bellman residual, which is the expectation of the TD-error. This makes sense, because when the Q-function is completely accurate for a policy π, the TD-error is zero. The objective is formed over the expectation over the data

samples contained in the replay buffer $\mathcal{D}$, or a minibatch thereof.

$$J_Q(\phi) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\phi(s_t,a_t) - \left(\mathbb{E}_{s_{t+1}\sim\mathcal{T}(\cdot|s_t,a_t)}\Big[R(s_t,a_t,s_{t+1}) + \gamma V(s_{t+1})\Big]\right)\right)^2\right]$$

where V can be rewritten in terms of Q_ϕ as $\mathbb{E}_{a_t\sim\pi}\Big[Q_\phi(s_t,a_t) - \alpha\log(\pi(a_t|s_t))\Big]$. In practice, a second neural network $Q_{\phi'}$ is used in the training target. This idea has been introduced in previous work [Lillicrap et al., 2016, Mnih et al., 2015], and has been shown to stabilize the learning process. Roughly, this provides a more stationary training target than using the current Q-function, and stabilizes training.

The policy is trained by minimizing the expected KL divergence between the policy and the exponentiated Q-values at a given state:

$$J_\pi(\theta) = \mathbb{E}_{s_t\sim\mathcal{D}}\left[D_{\mathrm{KL}}\left(\pi_\theta(\cdot \mid s_t)\middle\|\frac{\exp\left(\frac{1}{\alpha}Q_\phi(s_t,\cdot)\right)}{Z(s_t)}\right)\right].$$

Multiplying through by α and ignoring Z as it does not depend on θ gives:

$$J_\pi(\theta) = \mathbb{E}_{s_t\sim\mathcal{D}}\big[\mathbb{E}_{a_t\sim\pi_\theta}[\alpha\log(\pi_\theta(a_t \mid s_t)) - Q_\phi(s_t,a_t)]\big]$$

To minimize this objective using a sampling-based approach, and to allow for gradient information to flow through the sampling process, the policy is reparameterized [Kingma and Welling, 2014] as follows:

$$a_t = f_\theta(\epsilon_t; s_t)$$

where ϵ is sampled from a Gaussian distribution. This essentially decomposes the action sampling into a stochastic and deterministic part, allowing for normal backpropagation on the computation graph. The objective function is then:

$$J_\pi(\theta) = \mathbb{E}_{s_t\sim\mathcal{D},\epsilon_t\sim\mathcal{N}}\big[\alpha\log\pi_\theta\left(f_\theta\left(\epsilon_t; s_t\right) \mid s_t\right) - Q_\phi\left(s_t, f_\theta\left(\epsilon_t; s_t\right)\right)\big]$$

For more detailed information, see the conference paper. The algorithm pseudocode is given in (Algorithm 7).

Algorithm 7 Soft Actor-Critic (SAC)

Initialize policy parameters θ,
Initialize Q-function parameters ϕ_1, ϕ_2
Initialize temperature parameter α
Initialize target Q-function parameters $\phi'_1 \leftarrow \phi_1$, $\phi'_2 \leftarrow \phi_2$
Initialize the replay buffer $\mathcal{D}$
for each iteration **do**
  Collect experience by interacting with the environment using policy π_θ
  Store transitions (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
  for each gradient step **do**
    Sample a mini-batch of transitions from $\mathcal{D}$
    Update Q-functions by minimizing the critic loss:
      $\mathcal{L}^{critic} = \mathbb{E}[(Q\phi_i(s_t, a_t) - (r_t + \gamma \min_{j=1,2} Q_{\phi' j}(s_{t+1}, a_{t+1}) - \alpha \log \pi_\theta(a_{t+1}|s_{t+1})))^2]$
    Update the policy parameters by minimizing the actor loss:
      $\mathcal{L}^{actor} = \mathbb{E}[\alpha \log \pi_\theta(a_t|s_t) - Q_{\phi_1}(s_t, a_t)]$
    Adjust the temperature α to control the entropy term
    Update the target Q-function parameters:
      $\phi'_i \leftarrow \tau\phi_i + (1-\tau)\phi'_i$
  end for
end for

3.2.5 Model-based RL

MBRL methods [Plaat et al., 2023, Moerland et al., 2023, Luo et al., 2022] are given or learn from experience a model of the environment and then use it to improve their policy. The psychological basis for this family of algorithms was developed by Edward Tolman, who observed that animals who had a priori knowledge of an environment were better at solving various tasks within it compared to those that did not [Tolman, 1948]. Tolman argued that the experimental group had built a "latent model" or cognitive map of the environment which they then used when solving problems. This type of learning corresponds to *goal-directed* control of learned behavioural patterns [Dickinson, 1985], which implies knowledge of the relationship between actions and their consequences. This behaviour is clearly observable in humans [Lake et al., 2017, Corballis, 2009]; just as we construct abstract mental models to reason and plan, MBRL optimizes agent policies by simulating and evaluating potential future trajectories. This incorporation of temporally distant possibilities, or "imagination," is appealing as it enables agents to avoid pitfalls of unconstrained trial-and-error learning. MBRL's integration of learning a model and planning with it additionally promises the ability of transferring knowledge acquisition to different tasks within the same or different environments.

From a conceptual standpoint, MBRL offers those interested in designing general problem-

solving agents a framework to work within, and recently, in part due to advancements within deep learning [Bengio et al., 2013, LeCun et al., 2015], MBRL research activity has surged. State-of-the-art MBRL algorithms have shown strong results on RL benchmarks such as Atari and MuJoCo, achieving higher sample efficiency and similar asymptotic performance relative to MFRL methods [Hafner et al., 2023]. Although promising, MBRL algorithms remain data hungry relative to humans and are oftentimes extremely complicated from an algorithmic standpoint. Additionally, many fundamental questions in MBRL remain unanswered, such as how to identify the most crucial environment aspects the model should capture for effective downstream decision-making, or how model learning should proceed, particularly in high-dimensional state spaces. As such, MBRL methods vary widely. However, these can be roughly categorized into two classes according to the information their learned models capture.

Explicit MBRL

The classical approach is to learn the environment's dynamics model in order to make exact predictions of future states and rewards given particular states and actions. Besides enabling planning via repeatedly querying or "rolling out" the model, another benefit of and motivation for these kind of algorithms is that the model can be used to simulate new data at a fraction of the cost of gathering data directly from the environment, which can then be used to learn a policy. However, producing high quality data requires learning a high quality model, which is often a challenging task when the state space is large, high dimensional, or stochastic. Many different methods have been devised for dealing with the inherent uncertainties in learning from or planning with an imperfect model.

For example, DYNA-style [Sutton, 1991] algorithms incorporate model learning and planning with MFRL, wherein policy updates use real data gathered from the environment in addition to generated or imagined experience from the model. As more experience accumulates and the model becomes more accurate, the ratio between the two can be adjusted to incorporate more of the latter. This mixture reduces the impact of low quality data on the policy. Non-parametric Bayesian probabalistic models, such as Gaussian processes [Deisenroth and Rasmussen, 2011], take a principled approach to learning models with limited data by including uncertainty estimates for model predictions. These find success in low-dimensional robotic tasks such as the classic physically realized cart pole balancing problem or unicycle control, however become prohibitively expensive computationally when applied to problems with more complex state spaces. Other uncertainty-aware methods, like ensemble methods incorporating neural networks [Chua et al., 2018, Janner et al., 2019], average over multiple models to reduce the effect of model error and have demonstrated

comparable asymptotic performance to both Bayesian approaches and MFRL algorithms on low-dimensional continuous-control tasks in MuJoCo. Other hybrid approaches which combine MBRL and MFRL, such as [Nagabandi et al., 2018], which combines MBRL for sample-efficient initial model learning and MFRL for fine-tuning, or [Racanière et al., 2017], which combines a subpolicy distilled from data generated by a pretrained environment model with another trained in a standard MFRL manner, succesfully performed low-dimensional locomotive and visual puzzle tasks, respectively.

Latent MBRL

The convergence of advancements in distributed computing frameworks and deep learning algorithms has paved the way for another class of MBRL algorithms; namely the *latent planning* MBRL algorithms. Members of this family construct a compact, lower-dimensional representation of the decision process that retains data relevant to policy optimization, and then compute over quantities related to it instead of over raw experiential data. These methods draw on a key observation from representation learning [Bengio et al., 2013]; namely that certain input features have greater relevance to the optimization objective than others. By choosing, or even *learning*, a suitable mapping from raw observations to a descriptive lower dimensional latent space, these algorithms' internal models are desgined to abstract away inconsequential details and by doing so allow downstream tasks, such as value, reward, and latent state prediction during planning, to require less computational resources when compared with MBRL methods which plan at the observation level. This abstract model is typically architectured as a suite of ANNs that predict important quanties such as latent state transitions, rewards, and latent state values, and are either trained in a separate or end-to-end manner.

For example, in one strand of latent planning research [Ha and Schmidhuber, 2018], a random action selection policy gathers observations of a decision process, such as an OpenAI Gym environment, and then a variational autoencoder, a decoder-encoder neural network, compresses them into a lower dimensional space. This is achieved by minimizing the $L2$-loss between encoder-side observations gathered from the environment, and their corresponding reconstructions on the output side of the decoder. Afterwards and separately, a recurrent neural network is trained to predict future latent states over time, given an action signal and previous latent state, via maximum likelihood estimation. Taken together, these two networks form a "world model", which is rolled out to gather imagined training data for a linear control policy model, which selects actions in the real environment at test time. This approach outperformed MFRL algorithms such as DQN and A3C on visually high dimensional motor-control tasks.

Another strand of latent planning research [Hafner et al., 2019, Hafner et al., 2020, Hafner et al., 2021] culminating in Dreamer v3 [Hafner et al., 2023], outperforms specialized methods across 150 different RL benchmarks, all with a single hyperparameter configuration, also by learning a latent model of the environment and considering future scenarios within it. The algorithm features three neural networks: a world model that predicts the outcome of a given action in latent space, a critic network that judges a given action's value, and an actor network that chooses actions in the environment to reach the most valuable outcomes. The world model learns to map sensory inputs into discrete latent representations that are predicted by a sequence model, a type of recurrent ANN. The critic uses data generated by the world model underneath the actor's current action selection policy to predict distributions over latent state return values. The critic is trained by maximizing the likelihood of bootstrapped λ-returns [Sutton, 1988] of given latent states. The actor attempts to maximize future return while maintaining a degree of randomness to encourage exploration. It is trained alongside the critic on latent trajectories via the MFRL policy-gradient algorithm REINFORCE [Williams, 1992], while concurrently acting in the environment to generate new data to improve the world model.

As opposed to [Ha and Schmidhuber, 2018] wherein the model and policy learning occurs in distinct phases, learning in Dreamer proceeds in an interleaved fashion, with continual improvement of world model, critic, and actor ANNs during training. In both approaches, the learning objectives of the latent model and policy are separate, and the gradients do not flow through all components (ANN computation graphs) during backpropagation. [Lambert et al., 2020, Lambert, 2022] observes an *objective mismatch* between the model's learning paradigm and its future use in these and similar MBRL architectures, by showing that the historically accepted (log)-likelihood training objective of the (latent) dynamics model in MBRL does not necessarily correlate with downstream controller performance, even in low complexity environments. This suggests that the model should be trained in a way that is consistent with the task it will be used for. [Wei et al., 2023] provides a survey of recent solution categories addressing the objective mismatch problem in MBRL, however we will restrict the remainder of our discussion to the *value equivalent* approach.

Value Equivalent Latent Planning

Value equivalent latent planning algorithms take a principled approach to answering the open question of what environmental aspects MBRL algorithms' learned model should capture. The main idea is that the model's construction should take into account the future use of the model; in this context value-based planning is considered, wherein value estimates of states or state-actions guide policy optimization. The *value equivalence principle* [Grimm

et al., 2020] formalizes this notion by defining a model as being *value equivalent* to the true environment model underneath given sets of value functions and policies if the effect of the Bellman operator induced by any policy on any value function is the same for both. Given policies $\pi \in \Pi$ and value functions $v \in \mathcal{V}$, true model and approximation $m, \tilde{m} \in \mathcal{M}$, and Bellman operator $\mathcal{T}_{\pi,m}$ induced by policy π and model m, the approximation $\tilde{m}$ is value equivalent with respect to Π and $\mathcal{V}$ if and only if

$$\forall \pi \in \Pi, \forall v \in \mathcal{V}, \mathcal{T}_{\pi,m} v = \mathcal{T}_{\pi,\tilde{m}} v.$$

This establishes an equivalence relation which partitions the set of all possible models conditioned on given value and policy functions. As the considered sets of value and policy functions grow, the set of value equivalent models $\mathcal{M}(\Pi, \mathcal{V})$ shrinks as more constraints are introduced, eventually collapsing to the true environment model. From the perspective of an agent, any value equivalent model is indistinguishable to the true model with respect to value-based planning given the specified policies and value functions, which can be thought of as a "language" specifying those aspects of the environment that the agent should consider. Clearly, if a model is value equivalent underneath *all* possible policies and value functions, then using it to plan would result in optimal behaviour within the environment. The question then becomes whether selecting a subset of policies and value functions suffices to guarantee optimal planning. In this introductory work the authors show that under the same capacity constraints on value functions and models, a model trained to minimize a value equivalence loss based on the difference between the trained model's Bellman updates and sampled approximation of the true model's Bellman updates outperformed a model trained to minimize the traditional (log)-likelihood loss on low dimensional tasks. The authors additionally suggest that recent work in latent planning algorithms, including a member of the family of algorithms considered in this thesis, can be understood as approximations to the value equivalence principle.

The *order-k value equivalence principle* [Grimm et al., 2021] generalizes value equivalency by considering models that predict the same k-step Bellman updates as the true model for any policy and value function within specified sets, giving rise to the class of k-step value equivalent models $\mathcal{M}_k(\Pi, \mathcal{V})$. Since the Bellman operator is a contraction mapping, the result of applying the operator $k \to \infty$ times to any value function will converge to the same fixed point V_π (the given policy's true value function). This limit process induces the *proper value equivalence class*

$$\mathcal{M}_\infty(\Pi) = \lim_{k \to \infty} \mathcal{M}_k(\Pi, \mathcal{V}) = \{\tilde{m} \in \mathcal{M} : \forall \pi \in \Pi, \tilde{v}_\pi = V_\pi\}$$

which contains those models that share the same value functions with the true model across considered policies in the limit of Bellman operator applications. In this context, the set of considered policies uniquely determines the set of value functions, namely those that converge to the same value function according to the true model. Hence, the problem is reduced to specifying those policies that give rise to a set of models guaranteed to be optimal for planning. The authors use this principle to expand on their previous work, showing how minimizing a simplified version of the loss function of the latent model in MuZero simultaneously minimizes a proper value equivalence loss, providing a theoretical explanation for the algorithm's strong empirical performance.

We now introduce the *latent end-to-end Monte Carlo tree searchers*. Inspired by previous work that integrates learning and planning into an end-to-end process, such as [Silver et al., 2017b], and algorithms which incorporate Monte Carlo tree search (MCTS) as a policy improvement operator [Silver et al., 2017a], these algorithms integrate unified deep RL model, value function, and policy learning with MCTS to achieve state-of-the-art performance in complex discrete and continuous decision-making processes. These algorithms include MuZero [Schrittwieser et al., 2020] and its extensions, which improve its capabilities and decrease its computational requirements. We present a brief overview here, and later we describe MuZero's design, training, and acting processes in more detail.

Similar to previously mentioned latent MBRL algorithms, these algorithms learn a latent world model consisting of a representation and dynamics function, which encodes an observation into a latent state, and predicts the next latent state and reward, respectively. They also feature policy and value functions, which predict policies and values for latent states. These predictions are used to guide a search algorithm, which produces an enhanced target policy which forms a term of the optimization objective during training. Data is gathered from the environment by selecting actions according to the current policy, and stored in a replay buffer. These data are then used alongside the enhanced policy to shape the latent world model and prediction function estimates in a unified or *end-to-end* fashion, by backpropagating errors through all ANN's computation graphs.

3.3 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [Coulom, 2006] is a family of randomized search algorithms which approach planning in sequential decision-making problems through selective lookahead search. These algorithms collect statistics within the decision space in order to strategically guide the search into regions which are more promising for the decision maker. These algorithms have demonstrated strong performance in many challenging settings [Browne et al.,

2012], most notably in the game of Go, in which it is often difficult to devise a strong heuristic for use with traditional minimax-based search methods. In the context of the family of MBRL algorithms considered in this thesis, MCTS plays a crucial role in the model learning and acting processes.

3.3.1 Monte Carlo Methods

Monte Carlo methods approximate random variables by repeatedly taking random samples from the variable's underlying probability distribution and averaging them. The Law of Large Numbers from probability theory supports these methods, guaranteeing that the sample mean converges to the population mean in the sample limit:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n} X_i = \mathbb{E}[X]$$

where all X_i's are sampled from random variable X's underlying distribution.

These methods can be used in the context of MDPs to estimate the value of actions from a particular state. As a simple example of such a Monte Carlo method, consider an initial state s_0 in an MDP at time $t = 0$. The value of taking action a at s_0 is estimated in the following way. Run n simulations from s_0 until a terminal configuration is reached, choosing at each timestep t a random action a_t to advance the MDP to s_{t+1}. Let $N(a)$ be the number of times a was selected as the action from s_0, and let $R(a)$ be the running reward total collected within these simulations when choosing a at s_0. The value of a is then approximated as $\frac{R(a)}{N(a)}$ at the end of the n simulations.

3.3.2 Rollout-based MCTS

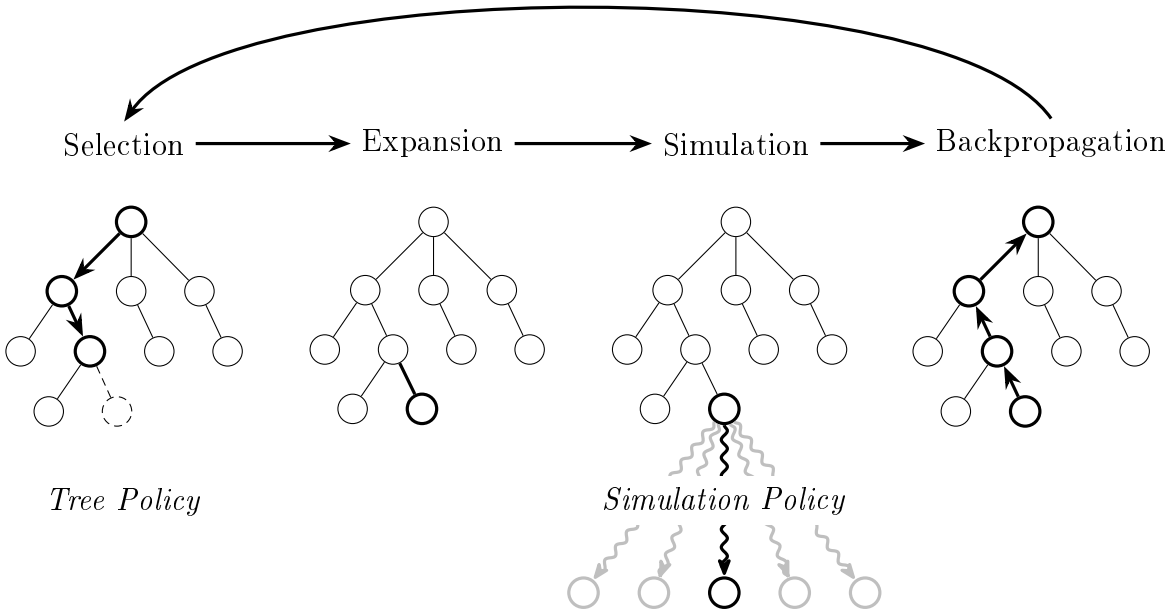

Figure 3.2: A single iteration of the standard MCTS algorithm

Rollout-based MCTS algorithms (Algorithm 8) use Monte Carlo action-value approximation to incrementally build an asymmetric lookahead tree. Each node in the tree corresponds to a state of the decision process, and each edge corresponds to an action from a state to a successor state. The tree is built over multiple iterations until a stopping criterion is met, such as a compute threshold or an iteration count, at which point the root actions are evaluated according to a performance measure and the best action (or distribution over actions) at the root of the tree is returned. The below sequence describes the scheme for a single search iteration of a vanilla MCTS, which is also shown in Figure 3.2.

1. Selection: Starting from the root node, the search tree is traversed using a *tree policy* to select actions, until it arrives at a non-terminal node with unvisited children.

2. Expansion: One or more of the selected node's children are added to the tree.

3. Simulation: A number of simulations, or rollouts, are run from one of the expanded children using a *simulation policy* to select actions. The rollouts are sequences of state-action-reward tuples generated from the decision process' forward model. The rollouts end at terminal states, and are then assigned a score, such as the total reward gathered during the simulation.

4. Backpropagation: The results of the rollouts are used to update statistics of nodes on the path from the node at which the simulations were conducted to the root.

Algorithm 8 MCTS

```
function MCTS(root state s_0, number of simulations I)
    Create a root node v_0 corresponding to MDP state s_0
    for I times do
        v_s ← Select(v_0)
        v_s ← Expand(v_s)
        value ← Simulate(v_s)
        Backpropagate(v_s, value)
    end for
    return action leading to highest value child of v_0
```

3.3.3 Upper Confidence Bounds

MCTS' performance critically depends upon the selection phase of the algorithm, which selects actions according to the *tree policy*, as it determines into which part of the decision space to grow the tree. In the simplest case, this policy samples actions uniformly at random. More sophisticated approaches use statistics gathered from previous iterations to guide the tree's growth into promising regions. To do so, the policy must balance between exploration of yet unexplored regions with exploiting those regions which have demonstrated high value in previous iterations. There are multiple approaches towards specifying this trade-off, such as the beforementioned ϵ-greedy strategy, wherein the currently highest estimated value action is chosen with probability $1 - \epsilon$, and a random action is taken with probability ϵ.

An exploration method from the closely related finite-armed stochastic bandit [1] (FASB) setting, a popular framework for studying sequential decision making problems in which a learner sequentially samples from a finite set of distributions with the goal of maximizing reward, takes a different approach to solving this dilemma. In the FASB setting, an agent is tasked with selecting an option a from a finite set of A options at every timestep t over a finite time horizon T. These options are random variables with underlying unknown but fixed probability distributions, corresponding to the reward the agent will receive when selecting an option. The goal of the agent is to maximize the cumulative reward over T. Policies that

[1] The term bandit refers to a slot machine, colloquially referred to as a "one-armed bandit". The FASB problem is akin to sequentially selecting which slot machine lever to pull given a set of initially unknown machines, in order to maximize profit over a given number of pulls.

specify how options are to be selected are evaluated based on the loss caused by not always selecting the best option, also known as the regret. Let $X_{a,t}$ denote the reward payoff of selecting option a at time t, and let A_t be the option selected by the agent at time t, then the regret can be written as $R(T) = \max_a \mathbb{E}[\sum_{t=1}^{T}(X_{a,t} - X_{A_t,t})]$. A policy is said to resolve the exploration-exploitation trade-off if it achieves expected regret that is within a constant factor of the proven lower bound $\Omega(\log t)$ [Lai and Robbins, 1985]. The Upper Confidence Bound (UCB) method [Auer, 2003] achieves this by selecting an option a_t at time t according to

$$a_t = \operatorname*{argmax}_a \left[\hat{\mu}_{a,t} + C\sqrt{\frac{\ln t}{N(a,t)}} \right]$$

where $\hat{\mu}_{a,t}$ is the empirical reward estimate of option a at time t, $N(a,t)$ is the number of times the option has been selected thus far and C is a tuneable bias parameter. UCB selects the option which has the highest empirical reward estimate plus a scaled term that is inversely proportional to the number of times the option has been selected in the past. This additional term biases the selection towards options which have been explored less previously, and serves as a high confidence upper bound on the empirical error of $\hat{\mu}_{a,t}$.

3.3.4 UCB for Trees

Algorithm 9 shows how the above UCB exploration strategy is incorporated into an MCTS tree policy, in a process known as Upper Confidence Bounds for Trees (UCT) [Kocsis and Szepesvári, 2006]. Essentially, every node within the tree is treated like its own FASB problem instance. Here $C(v)$ corresponds to the set of v's children, $\frac{Q(v_i)}{N(v_i)}$ corresponds to the empirical value estimate of v_i, and $N(v)$ corresponds to the number of times v has been visited thus far. MCTS with this UCB enhancement converges in the iteration limit to the optimal policy in single agent domains and to the minimax value function in two player zero-sum games. [Kocsis and Szepesvári, 2006]

PUCT (Predictor + UCT) [Rosin, 2011] enhances the UCT selection strategy by incorporating prior probability estimates that provide contextual information on the likelihood of each action being optimal at different decision points within the tree. The predictors output

Algorithm 9 UCT Select

```
1: function Select(v)
2:     while v is not a leaf do
3:         v ← max_{v_i ∈ C(v)} [ Q(v_i)/N(v_i) + C sqrt( ln N(v) / N(v_i) ) ]
4:     end while
5:     return v
```

a distribution over actions given a state within the tree, assigning a weight to each action based on previous experience within or of the decision process. This predictor can take many forms, including statistical models, neural networks, or policies distilled from analysis of human expert trajectories. PUCT MCTS forms the foundation of the search algorithm in MuZero, as shown in the next section.

3.4 MuZero

In this section we describe in detail the original latent end-to-end MBRL Monte Carlo tree searcher, MuZero [Schrittwieser et al., 2020]. The algorithm and its derivatives are among the most successful approaches towards autonomous general agent design to date, matching or exceeding the superhuman performance of task specific planning algorithms in traditional games like chess and Go. They additionally were the first to achieve superhuman ability in a suite of Atari 2600 games [Bellemare et al., 2013], without any prior knowledge of their underlying dynamics or human supervision, and within a data constraint of 100k in-game transitions, which is roughly two hours of real-time game experience. These algorithms exhibit a first principles approach towards automatically building and understanding a model of a given decision process, demonstrating the potential of the MBRL framework to scale to more complex real-world problems. We proceed by describing the algorithm and its various components, including its latent model architecture and its acting and learning mechanisms. We provide pseudocode for both the high level algorithm (Algorithm 10) and its internal MCTS (Algorithm 11). This section concludes with a summary of MuZero's limitations and extensions. More details regarding the algorithm can be found in its conference paper, and in the paper of its immediate predecessor [Silver et al., 2017a].

3.4.1 Algorithm

MuZero is an offline, off-policy, deep MBRL algorithm that learns a parameterized value-equivalent model of a given decision process, and plans within it using MCTS in order to select actions and generate policy learning targets. This algorithm can be understood as falling within the *generalized policy iteration* RL framework, in which two interleaving processes, policy evaluation and policy improvement, are used to iteratively improve the agent's policy. In this context, the enhanced policy produced by the MCTS subroutine can be seen as the improvement operator, and the model's value estimates can be seen as the evaluation operator.

MuZero is divided into two main subroutines, the Actor and the Learner. The possibly

Algorithm 10 MuZero

Initialize model μ_θ, replay buffer D, and number of MCTS simulations I
function Actor
  loop
    $o \leftarrow$ initialize starting state
    while o is non-terminal **do**
      $s \leftarrow h_\theta(o_{\{n\}})$
      $\pi, v \leftarrow \text{MCTS}(s, \mu_\theta, I)$
      $a \sim \pi$
      $r, o' \leftarrow$ execute a in environment
      Add (o, a, r, π, v, o') to D
      $o \leftarrow o'$
    end while
  end loop

function Learner
  loop
    Sample batch of trajectories $B \sim D$
    $\ell \leftarrow$ compute loss on B (Equation (3.7))
    Update θ via gradient descent on ℓ
  end loop

numerous instantiations of the Actor are responsible for interacting with the environment, generating trajectories via a MCTS subroutine, and storing them in a replay buffer. The Learner samples trajectories from the replay buffer and uses them to update the model's parameters. The algorithm concurrently executes these two processes until a stopping criterion is met. During test time, the Actor is used to select actions. The high-level algorithm is shown in Algorithm 10.

3.4.2 Model

MuZero's model is a suite of several deep neural networks. A representation and dynamics network induce a compact, lower-dimensional MDP of the given decision process, which is then planned in using its prediction network to select actions. Let O, A, S denote the sets of environmental observations and actions, and latent hidden states, respectively.

- Representation network $h : O^n \to S$ encodes the previous n consecutive environment observations into a latent, lower dimensional space. This is architectured as a CNN for image-based input, and a MLP for vector-based input.

- Dynamics network $g : S \times A \to S \times \mathbb{R}$ produces a latent state and reward estimate,

given a previous latent state and real action. This is a MLP.

- Prediction network $f : S \to P(A) \times \mathbb{R}$ returns policy and value estimates of latent states. This is a MLP.

These networks are chained together to form a large computation graph μ_θ parameterized by network weights θ. Figure 3.3 shows how MuZero's networks are composed to generate and advance a latent MDP.

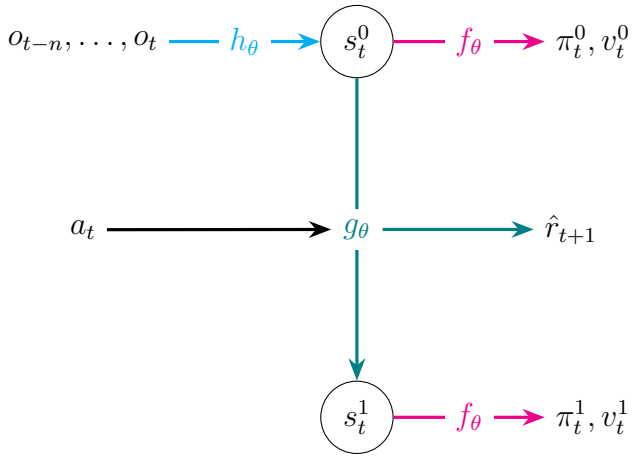

Figure 3.3: MuZero's function composition

3.4.3 MCTS

To select actions in the environment and to create policy learning targets for the model, MuZero executes a pUCT MCTS in the latent space. The major differences between this search algorithm and traditional MCTS are the inclusion of policy priors for action selection during the *selection* phase, and the proxying of *simulation policy* rollouts by bootstrapped value estimates. This has the benefit of incorporating an evolving representation to help guide the search into promising areas, as well as avoiding the computational bottleneck of simulating many rollouts in environments with long time horizons. These two components mirror the heuristic alpha-beta enhancement of minimax search, by reducing the width and the height of the search tree. MuZero's MCTS outputs a probability distribution over actions and a value estimate for the root node. Every node of the MCTS tree is associated with a hidden latent state s. The edges of the tree correspond to actions a from s. Each edge keeps track of a set of statistics, including $N(s, a)$, representing the number of times action a was taken in s, approximate action-value $Q(s, a)$, and policy prior $P(s, a)$. At the beginning of the search, the root latent state node is created by applying the representation function h_θ to the n previous observations. Subsequently, actions are selected by the pUCT selection

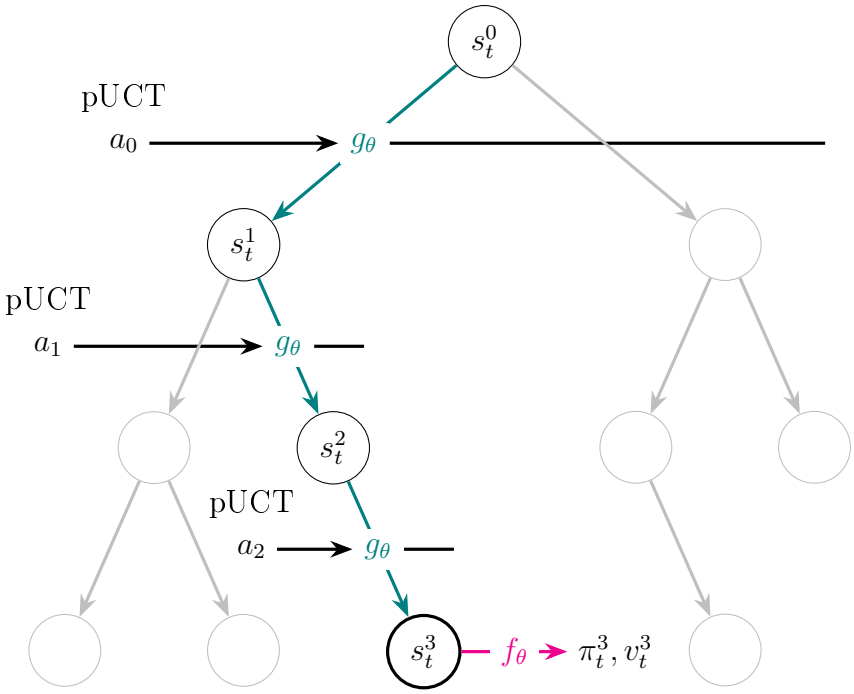

Figure 3.4: Snapshot of MuZero's MCTS adding a new node (s_3^t) to the search tree. At each node encountered during the tree traversal, all possible actions are evaluated according to their empirical value estimates and policy priors via the pUCT selection rule (Equation (3.2)).

rule, which balances between exploitation of high value actions and exploration of less visited actions, to traverse the tree:

$$a_k = \underset{a}{\operatorname{argmax}} \left[\underbrace{Q(s,a)}_{\text{value estimate}} + \underbrace{P(s,a)}_{\text{policy prior}} \cdot \underbrace{\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)} \cdot \left(c_1 + \log \left(\frac{\sum_b N(s,b) + c_2 + 1}{c_2} \right) \right)}_{\text{weighting of policy prior conditioned on visit counts}} \right], \tag{3.2}$$

Here, c_1, c_2 are constants controlling the relative influence of P compared with Q. Once an action is selected, it is passed alongside the previous latent state to the dynamics function g_θ to advance to next state s'. The model is recurrently rolled forward until reaching a leaf node s_l. A final action a_l is selected, and a new state and reward $s', r' = g_\theta(s_l, a_l)$ are computed. The prediction network f_θ computes a policy prior and value estimate for s', and a new node corresponding to it is added to the tree, as shown in Figure 3.4 and Algorithm 11. The statistics of all edges from the root to the leaf are then updated via backpropagation. Given a path length l from root to leaf, for each edge e on the path with distance k from the root, an $l-1-k$-step cumulative discounted reward estimate is calculated by bootstrapping from

Algorithm 11 MCTS in MuZero

function MCTS(root state s_0, model μ_θ, number of simulations I)
  initialize edge statistics for all a from s_0
  for $k = 1$ **to** I **do**
    $(r, s, v) \leftarrow$ SelectAndExpand(s_0)
    Backpropagate(r, s, v)
  end for
  return π^{MCTS}, v^{MCTS} (Equation (3.5) and Equation (3.6))

function SelectAndExpand(s)
  loop
    $a \leftarrow$ choose action according to pUCT (Equation (3.2))
    $(r', s') \leftarrow g_\theta(s, a)$
    if $N(s, a) = 0$ **then**
      add node s' as child of s on edge a
      initialize edge statistics for all a' from s'
      $\pi_\theta, V_\theta \leftarrow f_\theta(s')$
      return r', s', V_θ
    end if
    $s \leftarrow s'$
  end loop

function Backpropagate(r, s, v)
  for each edge e on the path from s to the root s_0 **do**
    update statistics for e (Equation (3.3) and Equation (3.4))
  end for

the value function estimate at the leaf:

$$G_k = \sum_{n=0}^{l-1-k} \gamma^n r_{k+1+n} + \underbrace{\gamma^{l-k} V(s_l)}_{\text{bootstrap}} \tag{3.3}$$

The discounted bootstrap is essentially a proxy for the information gathered by the numerous simulations from the terminal node in the traditional MCTS simulation phase. For every edge along the path from the leaf node to the root, $Q(s, a)$ and $N(s, a)$ are updated according to:

$$\begin{aligned} Q(s_{k-1}, a_k) &\leftarrow \frac{N(s_{k-1}, a_k) \cdot Q(s_{k-1}, a_k) + G_k}{N(s_{k-1}, a_k) + 1} \\ N(s_{k-1}, a_k) &\leftarrow N(s_{k-1}, a_k) + 1. \end{aligned} \tag{3.4}$$

After I MCTS iterations, the enhanced policy estimate π^{MCTS} and value estimate v^{MCTS}

are returned. The policy estimate is the visit count distribution at the root node:

$$\pi^{\text{MCTS}}(a) = \frac{N(s_0, a)^{1/T}}{\sum_b N(s_0, b)^{1/T}} \tag{3.5}$$

The temperature parameter T controls the degree of exploration in the search; its value is annealed from 1 to a small positive over the course of training such that the action selection policy becomes greedier as the model improves. The value estimate is the average discounted return at the root node:

$$v^{\text{MCTS}} = \sum_a \left(\frac{N(s_0, a)}{\sum_b N(s_0, b)} \right) Q(s_0, a) \tag{3.6}$$

3.4.4 Training

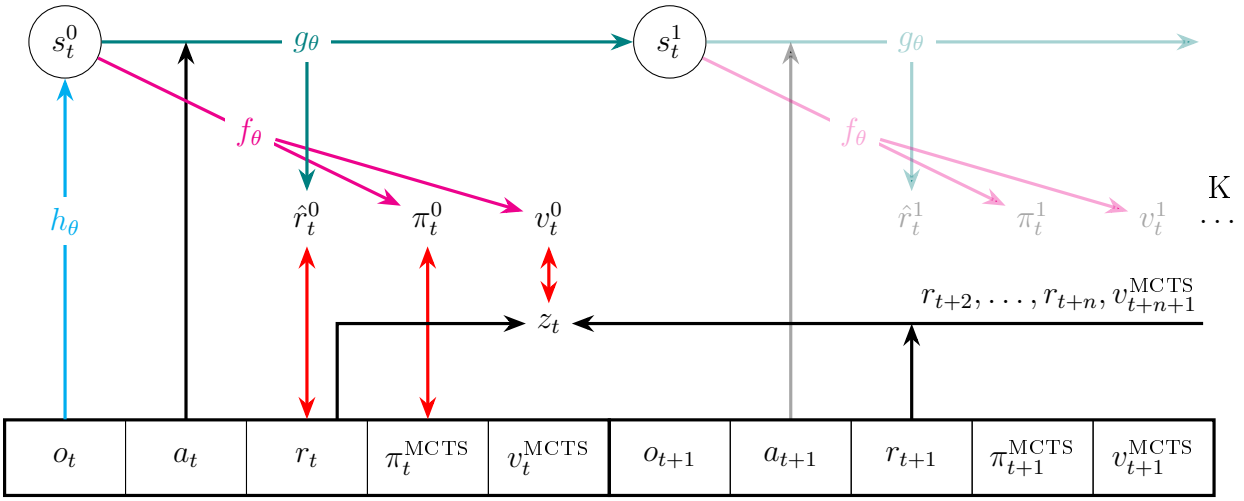

Figure 3.5: Relevant quantities for the first summation term in MuZero's sample loss at trajectory time t and rollout depth $k = 0$ (Equation (3.7)). The red lines indicate the values whose divergences are computed in the loss calculation. The low opacity components indicate the future network predictions that are included in the second loss summation term, when $k = 1$. The model is rolled forward a total of K times for every sample.

MuZero's fully differentiable recurrent model μ_θ is trained in an end-to-end manner via backpropation-through-time [Werbos, 1990]. The model is trained to minimize the error between its reward, value, and policy predictions and the grounded reward data alongside the value and policy targets generated by the Actors' MCTS searches. During training, trajectories of tuples of the form $\{o_t, a_t, r_t, \pi_t^{\text{MCTS}}, v_t^{\text{MCTS}}, o_{t+1}\}$ are sampled from the prioritized replay buffer. Given such a tuple with starting observation o_t and subsequent actions $a_t, \ldots, a_{t+K}$, μ_θ is unrolled for K steps, and the resulting predictions of reward, value, and

policy are aligned with the corresponding values along the equal length sequence from the trajectory, as seen in Figure 3.5. At each unrolled step $k = 0 \dots K$, the network has a loss to each target at that step, which are then summed to produce the loss for that tuple:

$$\ell_t(\theta) = \sum_{k=0}^{K} \underbrace{\ell_r(r_{t+k}, \hat{r}_t^k)}_{\text{reward loss}} + \underbrace{\ell_v(z_{t+k}, v_t^k)}_{\text{value loss}} + \underbrace{\ell_p(\pi_{t+k}^{\text{MCTS}}, \pi_t^k)}_{\text{policy loss}} + \underbrace{c\|\theta\|_2^2}_{\text{regularization}} \tag{3.7}$$

This size K window is then slid along the trajectory, and the losses are accumulated to form the total loss of the trajectory.

Depending on the context, the exact loss equations take particular forms. For games, where the reward only takes values in {-1,0,1} at the end of the trajectory, predicting intermediate rewards is nonsensical, and is excluded. In the general MDP case with potentially unbounded and intermediate rewards, the reward loss is either the mean squared error between the observed reward r_{t+k} and the model's prediction $\hat{r}_t^k$, or the cross entropy loss between a categorical representation of the observed reward and predicted reward. In games, the value loss computes the mean squared error between the value prediction and game outcome, and in general MDPs the cross entropy loss between the n-step bootstrapped discounted return $z_{t+k} = \sum_{i=0}^{n} \gamma^i r_{t+k+i} + \gamma^n v_{t+k+n+1}^{\text{MCTS}}$ and the model's value prediction v_t^k is calculated. The policy loss is always the cross entropy loss between the MCTS enhanced policy π_{t+k}^{MCTS} and the model's prediction π_t^k.

3.4.5 Limitations and Extensions

In its original form, although achieving significant results, MuZero is computationally expensive to train. It requires millions of experience samples to achieve strong performance in chess, Go, and Atari. This is in part due to the limited per transition feedback the model receives during training, and the off-policy learning issue caused by distilling stale data collected by previous weaker policies into the current policy. In MuZero, as training progresses, the current policy will increasingly differ from the policy originally used to gather trajectories for the Learner. This biases the state distribution and value targets along those trajectories, reducing the effectiveness of downstream policy improvement. To counteract this, a method called *Reanalyze* [Schrittwieser et al., 2021] refreshes the replay buffer by reevaluating old trajectories underneath the current model to produce better training targets. This reanalyzed data is then used by the Learner in proportion to or in lieu of data gathered from the environment, in an offline RL fashion, enhancing MuZero's data efficiency, and resulting in state-of-the-art in the 200k frame Atari benchmark.

Due to the nature of MuZero's MCTS, in which all actions are considered by pUCT

during the selection phase, MuZero is not applicable to environments with continuous action spaces, such as physical control tasks. *Sampled MuZero* [Hubert et al., 2021] changes the pUCT selection rule to only consider a subset of actions at each node, sampled from a distrbution determined by a proposal distribution β. Instead of considering only the policy prior π, the sampled-based equivalent $\frac{\hat{\beta}}{\beta}\pi$ is used when evaluating which action to select, with $\hat{\beta}$ being the empirical estimate of β, which in the simplest case is uniform. The MCTS produces an improved policy with respect to the subset of actions, which is then projected back into the space of policies that consider all actions. The model is then trained with regards to this policy, as is done in MuZero. *Sampled MuZero* retains MuZero's capabilities in discrete action spaces while extending it to continuous action spaces, outperforming state-of-the-art MFRL algorithms in the DeepMind Control Suite of benchmarks.

To address MuZero's sample inefficiencies, *EfficientZero* [Ye et al., 2021] incorporates an additional supervisiory signal during training via a latent state reconstruction loss. This loss is computed as the MSE between the hidden state vector produced by the dynamics function during rollouts and the result of applying the representation function to the corresponding temporally aligned observations from the replay buffer, in a self-supervised fashion. This provides more supervision for the dynamics functions, which is crucial to the prior predictions during MCTS. To address the off-policy issue arising from old data in the replay buffer, the value target z_{t+k} is dynamically adjusted based on the age of the sampled trajectory. For a number $l < n$ with l smaller for older data, the value target is adjusted to $z_{t+k} = \sum_{i=0}^{l} \gamma^i r_{t+k+i} + \gamma^l v^{\text{MCTS}}_{t+k+l+1}$, where the value estimate is recalculated via MCTS using the current policy. These enhancements, although deviating from MuZero's original intent of supervising model training solely via reward, policy, and value signals, allowed EfficientZero to achieve state-of-the-art performance in the 100k frame Atari benchmark.

MuZero's model is deterministic, and as such its performance is limited in environments that are stochastic (feature probabalistic transition dynamics) or partially observable. *Stochastic MuZero* [Antonoglou et al., 2022] generalizes MuZero to such environments by decomposing the original state transitions into two parts. The first action-conditioned transition moves from a given state to a so-called *afterstate*, which is a hypothetical state in the environment which occurs after an action has been chosen by the agent but before the environment has transitioned to a new state. This allows for separation of the effects of actions and the stochastic results induced by the environment. The second part is a probabalistic transition from an afterstate to the next state, where each result is a chance outcome conditioned on the states reachable through given state and action. To plan underneath this new formulation, the MCTS search is modified to consider alternating afterstate and state nodes in the search tree. The model features two new neural networks that learn the afterstate

dynamics and edge probability distributions. To train this model, a new term involving the afterstate predictions and their corresponding targets is added to the original MuZero's loss, and is then trained similarly to the original implementation. *Stochastic MuZero* exceeded the state-of-the-art in the stochastic games 2048 and backgammon, while maintaining the superhuman performance of MuZero in deterministic settings such as Go.

When the action space is large, and not every action is explored at the root of MuZero's MCTS, it can fail to provide an improved policy. As a result, when the MCTS 'enhanced' policy is distilled into the policy network's parameters during policy optimization, it worsens its performance. In *Gumbel MuZero* [Danihelka et al., 2022], MuZero's MCTS is retooled such that the search is guaranteed to result in an improved policy, even when not all actions at the root are investigated. This is relevant for environments with large action spaces, where exploring each action at the root is computationally unfeasible. To achieve this, the PUCB and PUCT action selection mechanisms are stripped out of MuZero's MCTS. Action selection at the root of the search tree is accomplished instead via a Gumbel-Top-k action sampling procedure, which iteratively samples m actions without replacement from the categorical action distribution according to $\text{argmax}_a(g(a) + logits(a))$, with $g \in \mathbb{R}^k$ is a vector of k Gumbel variables and logits provided by the policy network prior. Afterwards, a Sequential Halving algorithm determines the best action to take over these m actions, by dividing the simulation budget of n simulations equally to $\log_2(m)$ phases. In each phase, all considered actions are visited equally. After each phase, half of the actions are rejected according to their evaluation of $g(a)+logits(a)+\sigma(q(s,a))$, where σ is a function which scales the empirical value estimate proportionally to the visit count of a. Eventually, the procedure returns a single action, as shown in Figure 3.6. To construct an improved policy at the root, the calculated q values from the Sequential Halving phase are aggregated into a q-value vector $\mathbf{q}$, wherein non-sampled actions take on the default value $\sum_a \pi(a)q(s,a) \approx f_\theta(s)$. The new improved policy then takes the form $\pi' = softmax(logits(a) + \sigma(\mathbf{q}(s,a)))$, which is then distilled into the policy network π via a loss term $\mathcal{L}(\pi) \triangleq D_{\text{KL}}(\pi, \pi')$ during policy optimization. *Gumbel MuZero* also changes how actions are selected at non-root nodes during the simulation phase of MuZero's MCTS. Just as in the root-node action selection phase, a new policy π' is created via the completed q-values. Actions are then selected deterministically according to $\text{argmax}_a \left(\pi'(a) - \frac{N(a)}{1+\sum_b N(b)}\right)$, reducing the variance in rollouts. With these changes, *Gumbel MuZero* matched the state-of-the-art in chess, Shogi, and Go, and demonstrated continual learning in large action spaces even when sampling relatively an extremely small number of actions at the root of the search tree.

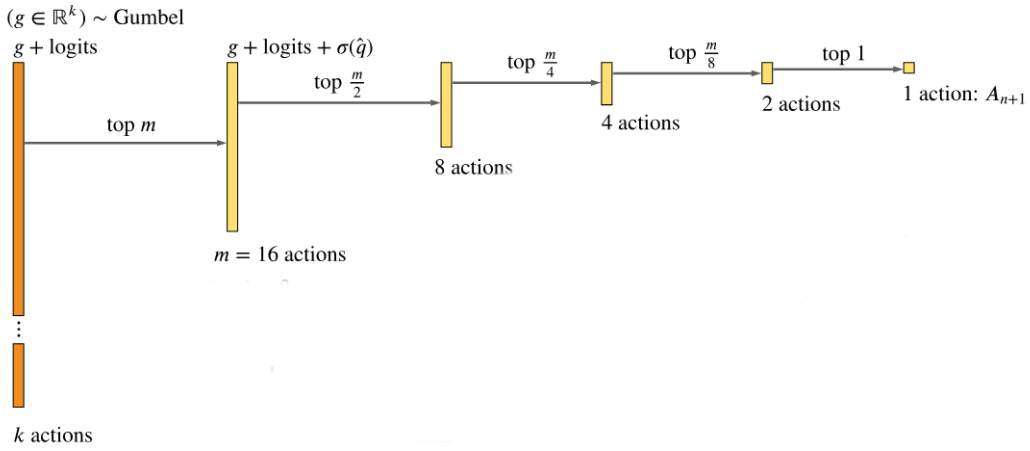

Figure 3.6: Sequential Halving algorithm for allocating action visit budget and selecting best action at the root of the search in Gumbel MuZero (from [Danihelka et al., 2022]).

Although MuZero and its derivatives demonstrate promising results in many RL benchmarks, these typically involve scenarios that require only short-term memory relative to many real-world tasks. It is known that RL algorithms generally struggle to perform in scenarios with long-term dependencies. Recently, UniZero [Pu et al., 2024] has addressed this by incorporating a transformer-based world model leveraging multi-head attention [Vaswani et al., 2017] with MuZero. This type of neural network architecture has shown success in dealing with larger context windows, especially in natural language processing applications, such as LLMs. In UniZero, the transformer architecture uses the entire trajectory during training, as opposed to previous MuZero-like algorithms wherein just the initial observation stack is encoded into the latent space. Just as in MuZero, UniZero's architecture includes a representation function, which maps observations to latent states, a dynamics function, which models transition and reward characteristics within the latent space, and a prediction function, which makes policy and value predictions. Additionally, UniZero incorporates a transformer 'backbone' network that is implicitly part of the latter two functions. This transformer maintains a memory of recent histories consisting of latent states and actions which is stored in its KV-cache. This cache is then queried during the MCTS to help guide the tree search, providing relevant context to predict future latent states, rewards, and policy/value evaluations of nodes. The entire architecture is trained in an end-to-end fashion using a prioritized replay buffer, just as in MuZero. The system was tested against other transformer-based deep RL architectures as well as MuZero derivatives, and showed the best performance on the long-term dependency benchmark VisualMatch, while remaining competitive with MuZero enhanced with an E0-like self supervised consistency loss on the Atari

benchmark.

MuZero and its derivatives are highly complex programs. The *LightZero* [Niu et al., 2024] open-source toolchain is a collection of benchmarks and interfaces for latent end-to-end MCTS algorithms, providing researchers with abstractions that simplify the complexities of these systems. The software suite contains implementations of many of the previously mentioned extensions of MuZero, taking a modular approach which decomposes these into data collector, data arranger, agent learner, and agent evaluator classes. It also provides various quality of life improvements for RL researchers, such as built out experiment logging and monitoring features, and distributed training optimizations. This reduces the entry barrier into this field of AI research, and allows for standardized comparison among these algorithms on a set of common benchmarks, opening the door for future research.

Chapter 4

Policy-Value Concordance

In this chapter we describe Policy-Value Concordance (PVC), a novel optimization algorithm for training deep RL agents incorporating parameterized policy and value predictors. In the context of decision-making, when the value function estimates encode useful information regarding which states of an environment are rewarding for the agent, then aligning the policy to the value function such that the probability of selecting actions is proportional to the estimated value received by taking them forms a coherent basis for downstream applications such as planning. PVC explicitly encourages this consistency through an auxiliary soft constraint on the policy and value function updates during training that measures the degree of alignment, or concordance, between the two. We hypothesize that if this additional objective is carefully balanced with existing learning objectives to avoid overwhelming the optimization landscape, its inclusion will reduce the volume of experiential data samples required by actor-critic RL algorithms to converge to an effective policy. Given that sample inefficiency is a significant barrier to deploying RL algorithms in real-world scenarios without perfect simulators, such a reduction would represent a step toward the development of general RL agents for these applications. We empirically investigate our hypothesis by incorporating our objective into several actor-critic RL algorithms, ranging from policy gradient methods such as advantage actor-critic with environment forward model access, to more complex algorithms like PPO across common deep RL benchmarks, with the goal of working towards integrating it with state-of-the-art MuZero derivatives. We begin by describing our methodology, followed by detailed explanations and results of the experiments we conducted to evaluate the effectiveness of our proposed objective. We conclude with a discussion of the implications of our findings, and avenues for future work.

4.1 Method

The intuition for PVC is that we would like to avoid situations in which the policy predicts action a_i with *higher* probability than a_j, when the value functions predict that taking a_i would lead to *lower* return than when taking a_j, and vice versa, particularly when the value function estimates become more accurate in later stages of training. Informally, this makes logical sense, as one's confidence in assessing the suitability of a particular circumstance is proportional to the amount of experience one has with it or similar circumstances. The following attempts to quantify this in the framework of sequential decision-making and MDPs. Given an experienced state s of a MDP, a learned stochastic policy π over actions conditioned on this state, and learned state-value function V and action-value function Q, PVC encourages $\pi(\cdot \mid s)$ to assign probabilities to actions according to the relative ordering of the value estimates of taking those actions. Given a state s with a discrete action set of size n with actions enumerated as $a_1, \ldots, a_n$ as well as non-negative Q, then for $i < j \leq n$, the policy and value function are *discordant* whenever

$$\begin{aligned} &\Big(\Big(\pi(a_i \mid s) \geq \pi(a_j \mid s)\Big) \wedge \Big(Q(s, a_i) < Q(s, a_j)\Big)\Big) \vee \\ &\Big(\Big(\pi(a_i \mid s) < \pi(a_j \mid s)\Big) \wedge \Big(Q(s, a_i) \geq Q(s, a_j)\Big)\Big). \end{aligned} \tag{4.1}$$

The alignment between policy predictions and one-step lookahead value estimates for a single state can thus be quantified by counting the number of discordances over all action pairs. This is achieved by evaluating the products of the pairwise differences in policy and value function predictions conditioned on a given state, and counting those for which the product is negative:

$$\textsc{discordant\_count}(\pi, Q, s) \triangleq \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{I}\Big[\Big(\pi(a_i \mid s) - \pi(a_j \mid s)\Big) \cdot \Big(Q(s, a_i) - Q(s, a_j)\Big) < 0\Big] \tag{4.2}$$

where $\mathbb{I}[\cdot]$ is the indicator function, returning 1 if the argument is true and 0 otherwise. In the case where access to the environmental transition dynamics or a model thereof is given, such that the resultant states $s'_1, \ldots, s'_n$ conditioned on the state and actions can be computed, this approach can be applied to V estimates, as well:

$$\textsc{discordant\_count}(\pi, V, s) \triangleq \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{I}\Big[\Big(\pi(a_i \mid s) - \pi(a_j \mid s)\Big) \cdot \Big(V(s'_i) - V(s'_j)\Big) < 0\Big] \tag{4.3}$$

Minimizing this discordance between policy and value functions is desirable, as, assuming that the value function estimate improves over the course of training, this ensures that the policy favors better actions over time, which directly increases performance during inference; either through direct application of the policy or through a search algorithm that incorporates it. Further, when the current policy is used to gather new data from the environment, as is the case in many RL algorithms, policy and value agreement will drive the gathered trajectories' state distribution into more rewarding areas of the state space. PVC shares a relationship with the concept of regret in RL, wherein an agent's actions are evaluated according to the difference between the reward obtained by applying them, and the highest possible reward attainable. Discordance arises when the actions chosen by the policy do not align with the highest-value estimates provided by the value function, meaning that the policy may select suboptimal actions according to the agent's understanding of the environment. These suboptimal choices lead to lower cumulative rewards, effectively acting as a source of regret. Although the PVC objective describes a desireable relationship between the policy and value functions from the perspective of a reward maximizing agent, particularly when the value function becomes more accurate, careful considerations must be made when integrating it with other learning objectives, as an overemphasis on concordance may impede their continual improvement during training. As such, the PVC loss is designed to be a soft constraint that guides the optimization process while still allowing the policy and value functions to learn useful representations.

4.1.1 Policy-Value Concordance Loss

To encourage concordance between parameterized policy and value functions during training of deep RL algorithms, PVC influences parameter updates through a loss term that is proportional to Equation (4.2), which the neural network optimizer will then try to minimize. Although Equation (4.2) captures the desired notion of policy and value function concordance discussed exactly, it incorporates the indicator step function, which is non-differentiable at $x = 0$ and has zero gradient elsewhere. As such, it is unsuitable as a deep neural network loss function as it does not provide useful information to the gradient descent optimizer about how to adjust the policy and value function parameters to decrease the loss during training. To address this, we propose a differentiable version of the PVC loss (Equation (4.4)). This version replaces the indicator function with the piecewise-linear leaky rectified linear unit

activation function, which is defined as:

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

This modifies the loss in two ways: first, the magnitude of the differences now contributes to the term, as opposed to just the sign, and second, the loss is now no longer strictly non-negative. When an i, j pair is discordant, the loss is increased in proportion to the magnitude of the product of differences of policy and value predictions. Conversely, when the pair is concordant, the loss is decreased in the same manner, at a rate of 0.01 times the magnitude of the product. This new formulation provides meaningful gradients to the optimizer, indicating how the policy and value network parameters need be adjusted in order to reduce the loss. The Q-version of the PVC sample loss $\mathcal{L}^Q_{PVC}$ is:

$$\mathcal{L}^Q_{PVC}(s) \triangleq \sum_{i=1}^{n} \sum_{j=i+1}^{n} \text{LeakyReLU}\Bigg(- \Big(\pi(a_i \mid s) - \pi(a_j \mid s)\Big) \cdot \Big(Q(s, a_i) - Q(s, a_j)\Big)\Bigg) \quad (4.4)$$

This loss formulation and the analagous V formulation can then be integrated in a general manner with other optimization objectives in various actor-critic RL algorithms. For example, in joint-optimization architectures featuring a value loss term $\mathcal{L}_\pi$, a policy loss term $\mathcal{L}_V$, the total loss $\mathcal{L}$ which the gradient descent optimizer attempts to minimize is then constructed as:

$$\mathcal{L} = \mathcal{L}_\pi + \alpha\mathcal{L}_V + \beta\mathcal{L}_{PVC}$$

Here, the weighting coefficients α, β set the relative influence of the different optimization objectives. In SOTA production RL algorithms, the policy and value coefficients are static, and the coefficients are determined experimentally, essentially acting as an additional hyperparameter. In PVC-augmented algorithms, increasing the value of β can be interpreted as focusing on exploiting more than exploring, and must thus be carefully tuned with respect to other learning objectives in order to prevent converging to local optima or failing to converge at all to a useful policy.

4.1.2 PVC Implementation Details

In deep learning frameworks such as Pytorch it is standard practice to write batched loss functions which leverage vectorized tensor operations to efficiently compute over multiple samples. Following this design principle we implement a batched PVC loss (Algorithm 12) that computes over input tensors of shape [*batch size*, *number of actions*] containing *batch*

Algorithm 12 Batched PVC Loss

function PVC_LOSS(
batched policy predictions tensor π_e,
batched value estimates tensor Q_e)
  $\pi_{\text{diff}} \leftarrow$ pairwise policy differences
  $Q_{\text{diff}} \leftarrow$ pairwise value differences
  $indices \leftarrow$ upper triangular indices
  $\pi_{\text{triu}} \leftarrow \pi_{\text{diff}}[indices]$
  $Q_{\text{triu}} \leftarrow Q_{\text{diff}}[indices]$
  $prod \leftarrow \pi_{\text{triu}} \times Q_{\text{triu}}$
  $score \leftarrow \text{LeakyReLU}(-prod)$
  return sum over mean $score$

size samples of state conditioned policy or one-step value estimates. The pairwise differences within the policy and value function evaluation tensors are computed by broadcasting their rows against their columns. The upper triangular indices are then gathered and the corresponding elements are sliced out of the tensors, to avoid double counting symmetrical pairs. The product of the tensor differences between these is then calculated and then a vectorized LeakyReLU is applied to its negation, as in (4.4). We then aggregate over the mean per-sample loss across the batch.

To equalize the relative impact of the policy and value function magnitudes on the PVC loss, and to provide a general approach that can be applied to unbounded value predictions, in practice before executing (Algorithm 12) we transform the one-step lookahead value estimates into a probability distribution-like representation similar to the policy predictions. This is achieved either via a softmax transform:

$$\text{Softmax}\big(Q(s,a_i)\big) = \frac{\exp\big(Q(s,a_i)\big)}{\sum_j \exp\big(Q(s,a_j)\big)}$$

which enforces all value estimates to sum to one and fall in the range $[0,1]$, or through a

standardization and normalization transform T over the vector of Q-values $Q(s_t, \cdot)$:

$$\text{Standardize}(t) = \frac{t - \mu}{\sigma}, \quad \mu = \frac{1}{n}\sum_{i=1}^{n} t_i, \quad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(t_i - \mu)^2}$$

$$\text{Normalize}(t) = \frac{t - t_{\min}}{t_{\max} - t_{\min} + \epsilon}, \quad t_{\min} = \min(t), \quad t_{\max} = \max(t)$$

$$T\big(Q(s_t, \cdot)\big) = \text{Normalize}\Big(\text{Standardize}\big(Q(s_t, \cdot)\big)\Big)$$

which first centers the value estimates around zero and enforces unit variance, and then rescales the values to fall into the range $[0, 1]$. This results in neither policy nor value function pair differences dominating the loss term.

As described in more detail in the following sections below, we investigate multiple approaches towards balancing the PVC learning objective with other optimization objectives, such as policy or value function loss terms in a variety of deep RL algorithms. The PVC loss can be weighted by a scalar β to control its relative importance during training, and can be hardened over time to account for the inaccuracies in the value function estimates in the early stages of training. The general PVC loss formulation is compatible with any actor-critic RL algorithm that utilizes a stochastic policy and has the ability to generate one-step lookahead value estimates. For algorithms without access to a learned or provided model of the environment, this requires a Q-value network or head to be present in the neural network architecture.

4.2 Experiments

This section describes the experimental methodology used to evaluate the effectiveness of the proposed PVC method within various deep RL architectures, with the goal of working towards integrating it with state-of-the-art MBRL methods such as MuZero derivatives. The experiments are designed to compare the performance of baseline actor-critic RL algorithms and their PVC-augmented counterparts. Our hypothesis is that incorporating the PVC loss will enhance sample efficiency, i.e., reduce the number of real-world interactions required to converge to an effective policy, relative to the baseline algorithms. To test this, we conduct experiments on several benchmark environments. Each experiment follows the general structure outlined below:

- We first train the baseline algorithm on the given environment using multiple random seeds, collecting performance metrics such as environmental interactions required to achieve a specified performance goal.
- We then train PVC-augmented versions of the same algorithm over the same set of random seeds.
- We then statistically compare the performance between the baseline and PVC-augmented algorithms, using methods such as the Wilcoxon rank-sum test to assess for significant differences in central tendencies within the performance measures. These are explained in more detail in the results section.

In addition to the primary performance comparisons, we explore several optimization strategies related to the PVC term. These include varying the relative weight of the PVC objective via the scalar β, as well as scheduling techniques that gradually harden the PVC loss during training to mitigate potential inaccuracies in early-stage value function estimates. We also investigate the effect of backpropagating gradients through just the policy, instead of through both the policy and the value function during parameter optimization. We aim to determine how these factors influence both sample efficiency and final performance.

The code is written in Python using the PyTorch deep learning framework [Paszke et al., 2019]. Baseline algorithm implementations are provided by CleanRL [Huang et al., 2022]. The less computationally demanding experiments were run on a quad CPU core i5 with a single NVIDIA GeForce RTX 3090 GPU, with 24GB of memory. The more demanding experiments were run on the Digital Alliance of Canada Cedar SLURM-enabled compute cluster, on nodes featuring Intel Silver 4216 Cascade Lake CPUs and NVIDIA V100 Volta GPUs with 32G of HBM2 memory.

4.2.1 Environments

The experiments were conducted using environments from or based upon those contained within the popular Gymnasium OpenAI Gym fork [Towers et al., 2024], an open-source toolkit used by RL researchers for the development and benchmarking of RL algorithms. Gymnasium offers a number of useful abstractions, such as vectorized environments for parallelized rollouts and policy evaluations, which allow researchers to focus on algorithm design instead of writing their own environment implementations from scratch. This toolkit allows for reproducibility in experiments and fair comparison across algorithms. The API is straightforward and allows for easy modification to pre-existing standardized environments, which include simple, low-dimensional problems with discrete state and actions spaces, as

well as more complex problems involving continuous actions and high-dimensional state representations. The following details the environments which we use for our experiments.

CartPole

First introduced in [Barto et al., 1983], this classic benchmark is designed to test the ability of RL algorithms to control a simple dynamical system consisting of an inverted pendulum or pole attached to a cart that moves along a frictionless track. The goal is to prevent the pole from falling over by applying forces to move the cart left or right. The state space is continuous, represented by four variables: the cart's position, the cart's velocity, the pole's angle, and the pole's angular velocity. The action space can either be selected to be discrete, with two possible actions: apply a force to the left or to the right to the cart, or continuous, wherein the applied force can vary along a continuum. The task is episodic, where an episode ends if the pole falls beyond a certain angle or the cart moves out of bounds. The reward structure is dense, providing a reward of $+1$ for every time step the pole remains upright, and the task is deemed solved if the agent manages to keep the pole upright for a threshold number of steps. The initial conditions of the system state variables are randomly sampled from a uniform distribution over $[-0.05, 0.05]$.

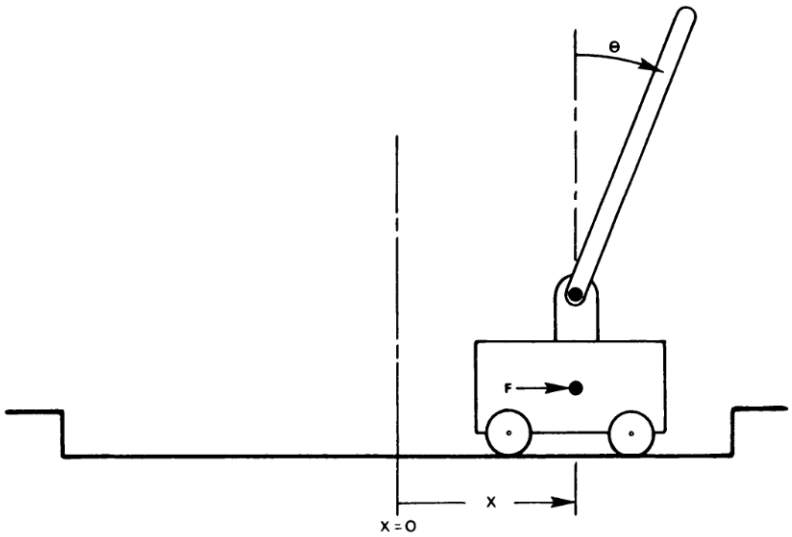

Figure 4.1: Visualization of the inverted pendulum problem, also known as CartPole (from [Barto et al., 1983])

LunarLander

The Lunar Lander environment simulates controlling a spacecraft as it descends toward the moon's surface. The agent must fire the craft's, or lander's, thrusters strategically to navi-

gate safely to a designated landing pad while taking into consideration gravity, inertia, and stochasticity in thruster application, simulated through the Box2D physics engine [Parberry, 2017]. The state space consists of eight continuous variables: the lander's x and y positions, its velocities in both directions, its angular orientation, its angular velocity, and two binary values indicating whether the left or right leg has made contact with the ground. The action space is discrete, with four possible actions: fire the main engine, fire the left or right side engines, or no op. The task is episodic, with an episode ending if the lander successfully touches down, crashes, or goes out of bounds. The reward structure is dense and incorporates several components such as a reward for moving closer to the landing pad, a reward proportional to the lander's speed, and large rewards or penalties for successful landings or crashes. Initial lander velocities are randomly initialized over a given range, and the moon's surface and landing zone location vary. The environment represents a complex control problem with multi-dimensional dynamics. For our experiments, we modify Gymnasium's original Lunar Lander implementation to account for a limited supply of fuel, as well as integrating a more sophisticated collision detection mechanism that prevents unrealistic lander behaviour, such as skipping along the moon's surface.

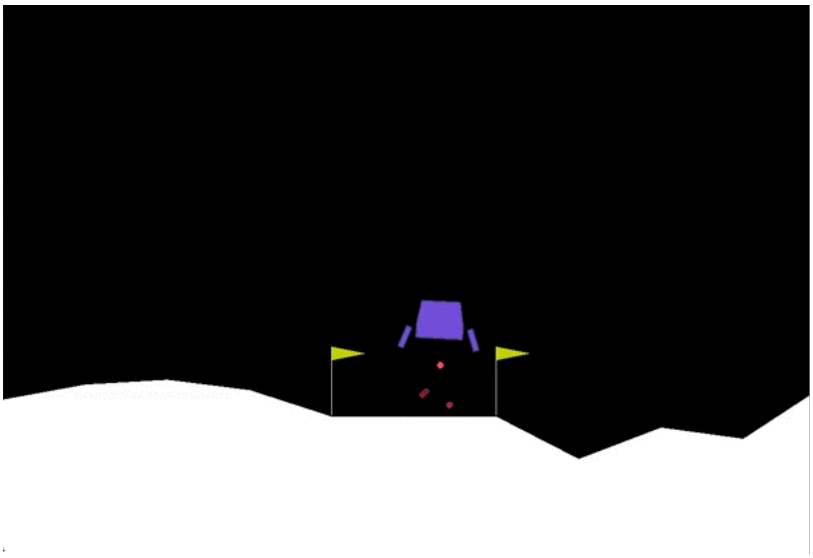

Figure 4.2: Visualization of the lunar lander environment

4.2.2 Advantage Actor-Critic with Forward Model Access

The first experiments we conducted were to integrate our method with a modified Advantage Actor-Critic (A2C) algorithm. Since the PVC loss is designed for algorithms which are able to generate one-step value function predictions, we modify A2C by providing access to the environment's forward model, allowing for collection of action-conditioned next state

samples during the rollout phase. In essence, this transforms the algorithm into a simple MBRL algorithm with a given model of the environment. The goal of this experiment is to serve as a proof of concept validating that a simple MBRL algorithm incorporating the PVC objective converges to an effective policy, which would provide justification for further investigation of the proposed PVC method. We train both the baseline and several versions of PVC-enhanced A2C agents over $n = 100$ seeds of the discrete CartPole environment. The enhanced versions take on the same hyperparameters as the baseline, and are differentiated only by how they set the static scalar PVC loss coefficient β in the range $[10, 1000]$. We set the success condition as balancing the pendulum for 500 timesteps. The experiments run until either the agent succeeds for ten contiguous episodes, or 600 training episodes elapse.

4.2.3 Proximal Policy Optimization with Forward Model Access

In these experiments, we integrate our method with a modified version of PPO and apply it to our custom Lunar Lander environment. The goal of these experiments is to test the generality of the PVC objective in both a more complex algorithmic setting as well as a more difficult control task. As PPO is a MFRL algorithm whose neural network architecture does not contain a Q-value head, we again provide forward model access through a custom Box2D state save and load mechanism, essentially transforming it into a MBRL policy gradient method. Preliminary results indicated that a more sophisticated approach for balancing the PVC objective with PPO's other actor and critic optimization objectives than the one implemented in the previous experiments was required, as a uniform search over β values yielded agents which failed to converge to similarly effective policies as the baseline. As a result, we implemented both a linear and geometric hardening schedule for β, in which the PVC loss gradually becomes more relevant to the optimization objective at each subsequent timestep during training, and investigated its effects. We conducted experiments in which the maximum β value took on values uniformly along a log-scale over $[1e^{-5}, 1e^{-1}]$. In addition, we also tested the effect of backpropagating PVC loss gradients through both the policy and the value function, as well as solely through the policy function. The experiments run until either the agent lands the craft successfully for ten contiguous episodes, or 500000 environmental interaction steps are taken.

4.3 Results

In this section we present results of two experiments we conducted to investigate the efficacy of our proposed PVC method. We begin by mentioning the statistical tools we use to analyze

the differences between the performance data of the baseline and PVC enhanced algorithms we tested. We then consider each set of experiments in turn and explain the results.

4.3.1 Statistical Analysis Methods

The RL algorithms which we use in our experiments are unstable from a performance standpoint due to numerous sources of variability, such as in initial network parameter initializations and episodic state distributions resulting from the stochasticity inherent in both the agents' policies and the environments' transition dynamics. As such, simply comparing the maximal performance achieved on a single training run or a set thereof for each algorithm variant with the baseline does not constitute a robust evaluation of their capabilities. To address this, multiple approaches for evaluating the significance of experimental results in deep RL have been suggested in the literature [Agarwal et al., 2021, Colas et al., 2019, Henderson et al., 2018]. After reviewing these, we decided on the following approach:

We compute 10000 resample bootstrap [Efron and Tibshirani, 1994] mean estimates and their corresponding 95% confidence intervals of the algorithms' performance data over those training runs where the specified success criterion is achieved. These statistics yield initial impressions of the algorithms' performance population distributions. In addition, we visualize the percentage of successful training runs for each candidate algorithm to determine whether it meets an acceptable threshold relative to the baseline.

For those experiments where the preliminary inspection indicates the possibility that an algorithm variant may have exceeded the baseline's performance, we plot both samples' mean per-episode reward alongside standard deviation and standard error statistics on the set of successful training runs. This allows us to visualize their relative temporal performance. Because of the differences in episodes required to achieve the task over a set of training runs, both dispersion statistics are calculated on a per episode basis.

Following this, we check for normalcy of both the baseline and contender algorithm's data's population distributions by applying the Anderson-Darling test ($\alpha = 0.05$). If the null hypothesis that the data follows a normal distribution is rejected for either, we perform the non-parametric left-tailed Wilcoxon rank-sums (Mann-Whittney U) test to evaluate whether the median of the differences of performance scores between the two is significantly less than zero. We prefer to use this test as opposed to the Kolmogorov-Smirnov (KS) test recommended in the literature, as the KS test statistic is susceptible to being influenced by outliers. Since we are soley interested in determining which algorithm has a better expected performance over the set of training seeds, we believe that a median based approach in the non-normal setting is appropriate. Otherwise, if both samples follow a normal distribution,

we employ the standard independent samples t-test to test whether the unknown population means are equal. These tests allow for statistically significant conclusions to be made about the relative performance between the baseline and contender algorithms.

4.3.2 PVC-A2C with Static Beta on CartPole

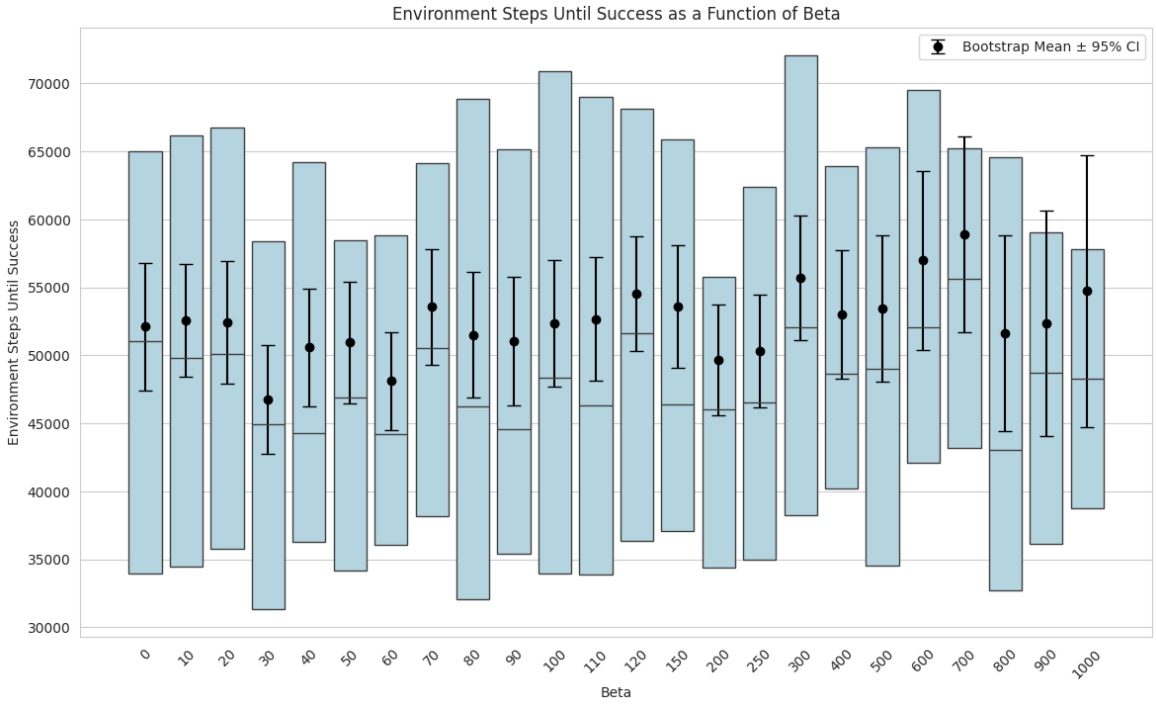

Figure 4.3: Boxplot showing interquartile range with median line of baseline ($\beta = 0$) and PVC-A2C agents' successful training run performance in the discrete CartPole environment. Each candidate algorithm was tested over the same set of $n = 100$ randomly seeded environments, with success defined as 10 contiguous episodes of greater than 500 total reward.

The results of the A2C experiments (Figure 4.3) demonstrate that the integration of our proposed PVC objective with an deep actor-critic RL algorithm can reduce the volume of environmental interactions required to learn an effective policy relative to baseline in a discrete low-dimensional control task.

In addition, the data show that the majority of tested PVC-A2C algorithms completed the task more often than the baseline (Figure 4.4) within the maximum number of allotted episodes.

The results of applying the Wilcoxon left-tailed rank-sum test to the performance data of baseline algorithm and PVC augmented variants show statistically significant ($p = 0.029 \pm 0.014$) central tendency differences between their population distributions for a number of

Beta	Test Statistic	p-value	Cliff's Delta	Vargha's A
30	-2.032	0.021	-0.1804	0.4098
60	-2.179	0.015	-0.1914	0.4043
80	-2.124	0.017	-0.1890	0.4055
200	-2.016	0.022	-0.1771	0.4114
250	-1.714	0.043	-0.1509	0.4245

Table 4.1: Significant Wilcoxon rank-sum test results and effect sizes.

β values (Table 4.1). The corresponding effect sizes ($\delta = -0.171 \pm 0.020, A = 0.414 \pm 0.010$) indicate a small difference in the proportion of non-overlapping values between the performance population distributions, as well as a slight expectation that a random value from the significant PVC-A2C populations is less than a random value from the baseline population, respectively.

The sample mean per episode reward curves with standard error (Figure 4.5) and standard deviation regions provide visual confirmation of the improved performance of the statistically significant PVC-A2C agents relative to the baseline.

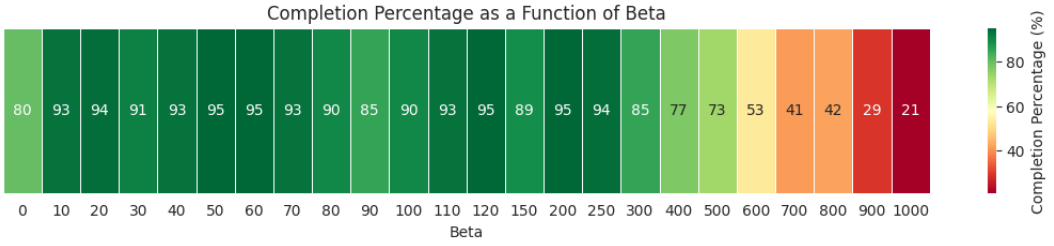

Figure 4.4: Heatmap displaying task success percentage of baseline and PVC-A2C agents over all 100 training runs.

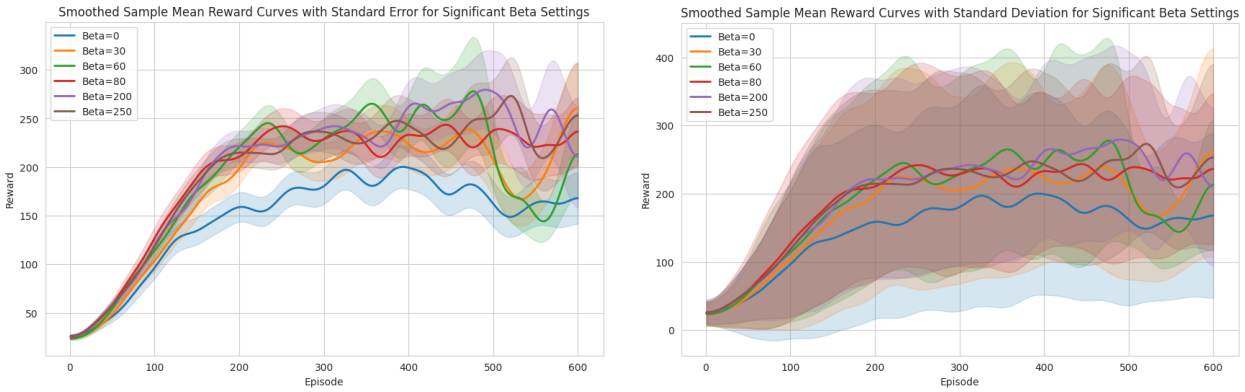

Figure 4.5: Reward curves for baseline and significant β PVC-A2C agents with standard error (left) and standard deviation (right). The mean curve and dispersion regions are smoothed via Gaussian filtering.

4.3.3 PVC-PPO with Linearly Hardened Beta on LunarLander

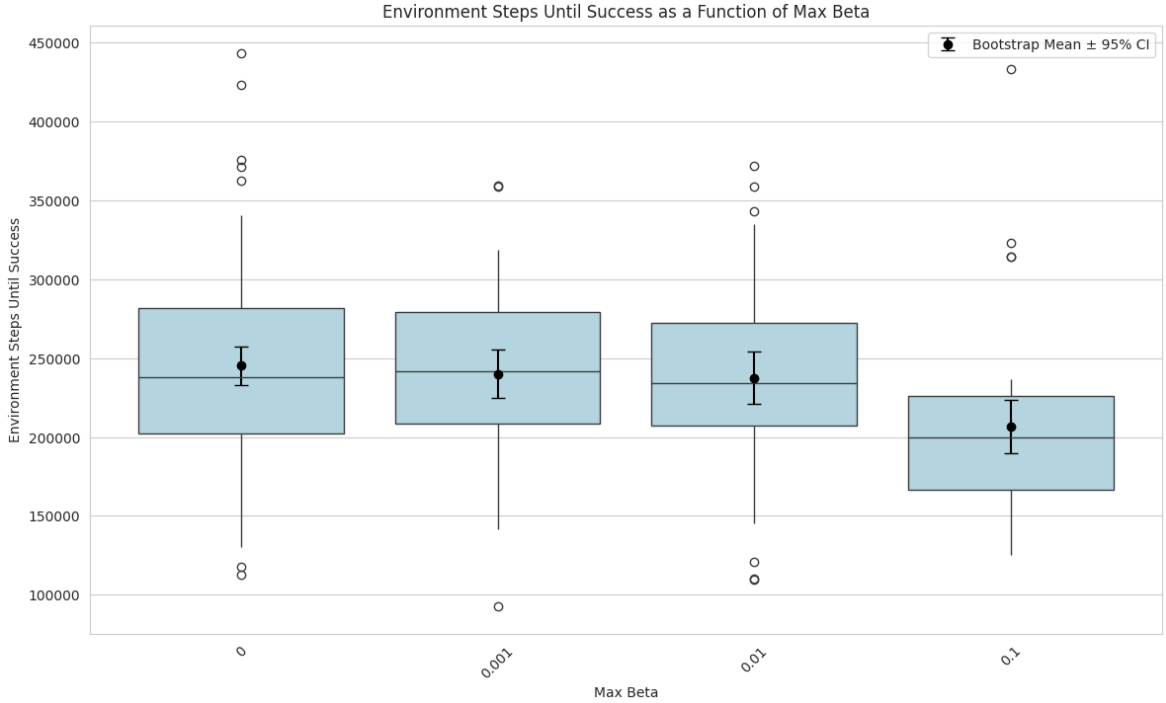

Figure 4.6: Boxplot showing interquartile range with median line of baseline (max beta = 0) and linearly hardened PVC-PPO agents' successful training run performance in our custom discrete Lunar Lander environment. The baseline was tested over $n = 100$ seeds and the PVC variants were tested over $n = 50$ seeds. The success condition is landing the lander ten times consecutively.

This experiment tests the performance of a PVC-enhanced PPO algorithm in our custom Lunar Lander environment. Figure 4.6 shows performance data corresponding to the baseline PPO agent and PVC-PPO algorithms that linearly harden β during the course of training. The data suggests that all considered PVC-PPO algorithms perform approximately equal to or better than the baseline.

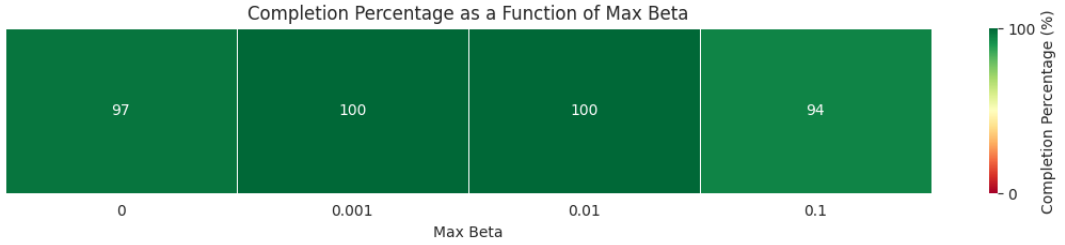

Figure 4.7: Heatmap displaying task success percentage of baseline and PVC-PPO agents over all training runs.

The visualization in Figure 4.7 shows that all considered algorithms succeed approximately in equal proportion over their corresponding training runs. In addition, the results of applying the Wilcoxon left-tailed rank-sums test to the performance data of the PVC-PPO agent with max beta $= 0.1$ and the baseline shows statistically significant ($p = 4.764e-06$) central tendency differences between their population distributions. The corresponding effect sizes ($\delta = -0.453, A = 0.273$) indicate a moderate difference in the proportion of non-overlapping values between the performance population distributions, as well as a moderate expectation that a random value from the significant PVC-PPO population is less than a random value from the baseline population.

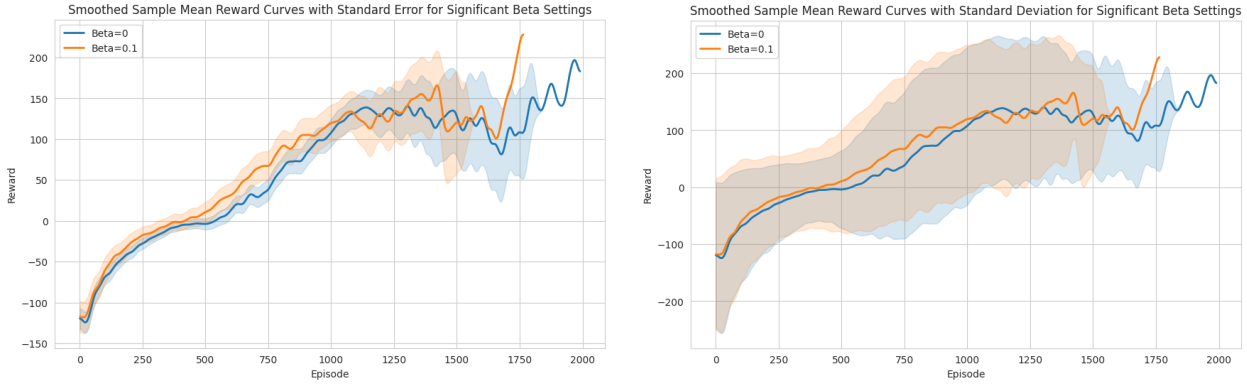

Figure 4.8: Reward curves for baseline and significant max beta $= 0.1$ PVC-PPO agent with standard error (left) and standard deviation (right).

4.4 Discussion

In this section we analyze our proposed PVC method by interpreting the results presented in the previous section. We describe the implications of the results on the suitability of the PVC objective as a loss term in deep actor-critic RL algorithm policy optimization strategies.

The PVC-A2C experimental results show that a majority of the PVC-A2C agents complete the specified task at a higher rate than the baseline. We speculate that this is a result of PVC-A2C exploiting more than A2C as a result of the concordance soft constraint, allowing faster convergence to an effective policy in this simple environment. The data does not provide meaningful insight into how β values are related to one another through performance or completion metrics, other than a rapid decline in completions for PVC-A2C algorithms with $\beta > 300$, where we believe the PVC loss begins to dominate the other optimization objectives, preventing the policy and value function from learning useful representations. We attribute the lack of a trend in β with regards to performance metrics to the high variance inherent in A2C's Monte-Carlo based policy optimization approach, wherein parameter updates occur only from completed trajectories. When A2C generates an informative low probability/high return episode in the initial phases of training when its policy is near random, future improvement is significantly accelerated. When no such event occurs, the policy updates are informed by lower information trajectories, slowing learning. This is corroborated by the high sample standard deviation statistics (Figure 4.5), as well as the large bootstrap mean confidence intervals. Therefore, we speculate that the sample distributions for the same PVC-A2C algorithm may vary considerably across different sets of random seeds. Despite this, and considering that the integration of the PVC objective with A2C was straightforward, i.e. β static for the duration of training, and still resulted in higher performing algorithms than the baseline (Figure 4.5), we infer that the results provide some evidence for the PVC objective's utility in deep actor-critic RL policy optimization.

Preliminary experiments integrating PVC with PPO using the same strategy as in the A2C experiments demonstrated that a more sophisticated approach towards balancing the PVC objective with other learning objectives was required. After searching over a range of β values and running experiments with these, the results showed that baseline performance was not being matched. We experimented with linear and geometric hardening schemas for β and found some marginal success, as shown in the results, however this is not a generalizable approach as the optimal β is environment dependent, suggesting that alternative methods for balancing β against other learning objectives are necessary.

In conclusion, while the results demonstrate that certain RL algorithms incorporating the PVC objective outperform baselines in certain environments, in its current form PVC's

general application to deep RL algorithms is limited by the complexity of β's tuning and dependence on the environment. Future work could consider adaptive mechanisms which take into consideration factors such as the entropy of the policy or changes in reward over time.

Chapter 5

Conclusion

This thesis has examined recent developments in sequential decision-making in computer science and has proposed a novel concept in the PVC objective, which addresses the primary limitation of state-of-the-art RL algorithms, namely their high sample complexity. This work hypothesized and provided preliminary evidence that the PVC objective may contribute meaningfully to policy convergence in deep actor-critic RL algorithms. The results showed that RL algorithms which incorporate the method complete tasks in some discrete, low dimensional environments with greater sample efficiency than their baselines, in a manner that is statistically significant.

However, several challenges and limitations emerged. These include difficulties in balancing the PVC objective with other conflicting learning objectives, such as entropy regularizers which desire high stochasticity in the policy. An additional challenge which became apparent was the relationship between the influence of the PVC objective on network parameter updates and the achieved reward characteristics of the algorithm during training; if the learning mechanism begins to emphasize concordance before the value function becomes accurate, the policy may converge to suboptimal solutions, or fail to improve at all. These limit PVC's generalizability in its current form, as β schedules must be tuned specifically for both the chosen algorithm and the environment such that hardening coincides with a threshold value function accuracy. The PVC objective can be interpreted as an exploitative learning signal which biases the rollout data's state distribution into regions which the current value function finds agreeable. If this is not correctly balanced against the need to explore regions of value function uncertainty, then the learning process might fail to yield effective policies. Thus, more sophisticated mechanisms than the ones introduced in this thesis are required in order for the PVC objective to generalize more effectively across tasks. The results presented in this thesis focused on discrete, low-dimensional environments. Attempts were made to extend PVC to high-dimensional, continuous-action tasks using SAC, however results were

inconclusive. Achieving this bridge would provide stronger evidence for PVC's utility in real-world problems, such as in robotics and control tasks where sample efficiency is critical. In order for progress to be made in more complex continuous action-space environments, challenges related to the convergence and stability of PVC would need to be addressed more rigorously, such as adapting the PVC loss to handle continuous distributions.

To address these limitations, future research could explore adaptive methods that dynamically adjust the PVC objective's influence on network parameter updates, based on learning indicators such as reward trends or the policy's entropy trends. Another avenue for future work is to mathematically analyze the PVC objective, e.g. its relationship to regret in RL or as a policy improvement operator in the generalized policy iteration design pattern. Applying more rigorous mathematical analysis may reveal characteristics of the PVC objective which this thesis failed to detect. Another area which was not sufficiently analyzed in this work was the influence of gradient backpropagation schemas on policy effectiveness and sample efficiency. No conclusions were made regarding the effect of either passing gradients backward through only the policy, or both the policy and the value functions on algorithm performance. Additionally, the construction of the PVC loss term could be further scrutinized, as it is not clear from the given analysis whether the signal for the optimizer could be improved. Different activation functions could be investigated in lieu of LeakyReLU, and a more efficient implementation could be sought in terms of space and time complexity. Additionally, alternative policy and value network architectures could be considered than the simple feed-forward ANNs in this work. Since PVC loss computation requires one-step lookahead value estimates for all actions given a state, an ideal Q-function architecture forward pass would output all of these estimates, such that multiple calls would not be required. The final goal of this research is to integrate the PVC term with state-of-the-art end-to-end MBRL algorithms such as MuZero derivatives, which this work takes a first step towards. However, the MBRL algorithms tested in this thesis are given the environment forward model, and the end-to-end algorithms learn the transition dynamics of a latent MDP. After devising a more suitable mechanism for balancing the relative impact of the PVC term, a next step would be to integrate it with such an end-to-end architecture. Lastly, to adapt PVC to continuous domains would likely require more complex loss constructions and sampling strategies, as in its current form, PVC requires all pairwise differences of policy and value function estimates to be evaluated.

Overall, this thesis contributes a review of recent advancements in computer planning, such as those in model-based reinforcement learning, and a new idea for enhancing sample efficiency in policy optimization in deep actor-critic RL algorithms, as well as including some evidence to support its further investigation.

Bibliography

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P. A., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: A system for large-scale machine learning. In Keeton, K. and Roscoe, T., editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 265–283. USENIX Association.

[Abbeel, 2020] Abbeel, P. (2020). Foundations of deep rl lecture series.

[Agarwal et al., 2021] Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. G. (2021). Deep reinforcement learning at the edge of the statistical precipice. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 29304–29320.

[Antonoglou et al., 2022] Antonoglou, I., Schrittwieser, J., Ozair, S., Hubert, T. K., and Silver, D. (2022). Planning in stochastic environments with a learned model. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

[Auer, 2003] Auer, P. (2003). Using confidence bounds for exploitation-exploration tradeoffs. *J. Mach. Learn. Res.*, 3(null):397–422.

[Barto et al., 1983] Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cybern.*, 13(5):834–846.

[Bellemare et al., 2013] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.*, 47:253–279.

[Bellman, 1957] Bellman, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684.

[Bengio et al., 2013] Bengio, Y., Courville, A. C., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828.

[Bertsekas and Tsitsiklis, 1996] Bertsekas, D. and Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Athena Scientific.

[Bick, 2021] Bick, D. (2021). Towards delivering a coherent self-contained explanation of proximal policy optimization.

[Bowling et al., 2023] Bowling, M., Martin, J. D., Abel, D., and Dabney, W. (2023). Settling the reward hypothesis. In *International Conference on Machine Learning*, pages 3003–3020. PMLR.

[Brown and Sandholm, 2019] Brown, N. and Sandholm, T. (2019). Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890.

[Browne et al., 2012] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

[Campbell et al., 2002] Campbell, M., Hoane Jr, A. J., and Hsu, F.-h. (2002). Deep blue. *Artificial intelligence*, 134(1-2):57–83.

[Chua et al., 2018] Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 4759–4770.

[Colas et al., 2019] Colas, C., Sigaud, O., and Oudeyer, P.-Y. (2019). A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms. *arXiv preprint arXiv:1904.06979*.

[Corballis, 2009] Corballis, M. C. (2009). Mental time travel and the shaping of language. *Experimental Brain Research*, 192:553–560.

[Coulom, 2006] Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer.

[Danihelka et al., 2022] Danihelka, I., Guez, A., Schrittwieser, J., and Silver, D. (2022). Policy improvement by planning with gumbel. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

[Degrave et al., 2022] Degrave, J., Felici, F., Buchli, J., et al. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602:414–419.

[Deisenroth and Rasmussen, 2011] Deisenroth, M. P. and Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In Getoor, L. and Scheffer, T., editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 465–472. Omnipress.

[d'Epenoux, 1963] d'Epenoux, F. (1963). A probabilistic production and inventory problem. *Management Science*, 10(1):98–108.

[Dickinson, 1985] Dickinson, A. (1985). Actions and habits: the development of behavioural autonomy. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 308(1135):67–78.

[Dulac-Arnold et al., 2021] Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., and Hester, T. (2021). Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468.

[Efron and Tibshirani, 1994] Efron, B. and Tibshirani, R. J. (1994). *An introduction to the bootstrap*. Chapman and Hall/CRC.

[Frostig et al., 2018] Frostig, R., Johnson, M. J., and Leary, C. (2018). Compiling machine learning programs via high-level tracing. *Systems for Machine Learning*, 4(9).

[Glimcher, 2011] Glimcher, P. W. (2011). Understanding dopamine and reinforcement learning: the dopamine reward prediction error hypothesis. *Proceedings of the National Academy of Sciences*, 108(supplement_3):15647–15654.

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

[Grimm et al., 2021] Grimm, C., Barreto, A., Farquhar, G., Silver, D., and Singh, S. (2021). Proper value equivalence. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 7773–7786.

[Grimm et al., 2020] Grimm, C., Barreto, A., Singh, S., and Silver, D. (2020). The value equivalence principle for model-based reinforcement learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5541–5552. Curran Associates, Inc.

[Gu et al., 2017] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE.

[Ha and Schmidhuber, 2018] Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2455–2467.

[Haarnoja et al., 2018a] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018a). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR.

[Haarnoja et al., 2018b] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018b). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR.

[Hafner et al., 2020] Hafner, D., Lillicrap, T. P., Ba, J., and Norouzi, M. (2020). Dream to control: Learning behaviors by latent imagination. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[Hafner et al., 2019] Hafner, D., Lillicrap, T. P., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019). Learning latent dynamics for planning from pixels. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565. PMLR.

[Hafner et al., 2021] Hafner, D., Lillicrap, T. P., Norouzi, M., and Ba, J. (2021). Mastering atari with discrete world models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

[Hafner et al., 2023] Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. (2023). Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*.

[Haith and Krakauer, 2013] Haith, A. M. and Krakauer, J. W. (2013). Model-based and model-free mechanisms of human motor learning. In *Progress in motor control: Neural, computational and dynamic approaches*, pages 1–21. Springer.

[Henderson et al., 2018] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In McIlraith, S. A. and Weinberger, K. Q., editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3207–3214. AAAI Press.

[Hill et al., 2018] Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2018). Stable baselines.

[Huang et al., 2022] Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and Araújo, J. G. (2022). Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18.

[Hubert et al., 2021] Hubert, T., Schrittwieser, J., Antonoglou, I., Barekatain, M., Schmitt, S., and Silver, D. (2021). Learning and planning in complex action spaces. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 4476–4486. PMLR.

[Hutter, 2005] Hutter, M. (2005). *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin.

[Hutter et al., 2024] Hutter, M., Quarel, D., and Catt, E. (2024). *An Introduction to Universal Artificial Intelligence*. CRC Press.

[Janner et al., 2019] Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12498–12509.

[Kahneman, 2011] Kahneman, D. (2011). *Thinking, fast and slow*. Farrar, Straus and Giroux, New York.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

[Knuth and Moore, 1975] Knuth, D. E. and Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial intelligence*, 6(4):293–326.

[Kocsis and Szepesvári, 2006] Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, page 282–293, Berlin, Heidelberg. Springer-Verlag.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

[Lai and Robbins, 1985] Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22.

[Lake et al., 2017] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and brain sciences*, 40:e253.

[Lambert et al., 2020] Lambert, N., Amos, B., Yadan, O., and Calandra, R. (2020). Objective mismatch in model-based reinforcement learning. *arXiv preprint arXiv:2002.04523*.

[Lambert, 2022] Lambert, N. O. (2022). *Synergy of Prediction and Control in Model-based Reinforcement Learning*. PhD thesis, University of California, Berkeley.

[Lanctot et al., 2019] Lanctot, M., Lockhart, E., Lespiau, J.-B., Zambaldi, V., Upadhyay, S., Pérolat, J., Srinivasan, S., Timbers, F., Tuyls, K., Omidshafiei, S., Hennes, D., Morrill, D., Muller, P., Ewalds, T., Faulkner, R., Kramár, J., Vylder, B. D., Saeta, B., Bradbury, J., Ding, D., Borgeaud, S., Lai, M., Schrittwieser, J., Anthony, T., Hughes, E., Danihelka, I., and Ryan-Davis, J. (2019). OpenSpiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[Lee et al., 2012] Lee, D., Seo, H., and Jung, M. W. (2012). Neural basis of reinforcement learning and decision making. *Annual review of neuroscience*, 35:287–308.

[Liang et al., 2018] Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M. I., and Stoica, I. (2018). Rllib: Abstractions for distributed reinforcement learning. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3059–3068. PMLR.

[Lillicrap et al., 2016] 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 Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

[Littman et al., 1995] Littman, M. L., Dean, T. L., and Kaelbling, L. P. (1995). On the complexity of solving markov decision problems. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, page 394–402, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Luo et al., 2022] Luo, F.-M., Xu, T., Lai, H., Chen, X.-H., Zhang, W., and Yu, Y. (2022). A survey on model-based reinforcement learning. *arXiv preprint arXiv:2206.09328*.

[Machado et al., 2018] 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 (extended abstract). In Lang,

J., editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 5573–5577. ijcai.org.

[Mankowitz et al., 2023] Mankowitz, D. J., Michi, A., Zhernov, A., Gelmi, M., Selvi, M., Paduraru, C., Leurent, E., Iqbal, S., Lespiau, J.-B., Ahern, A., et al. (2023). Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263.

[Markov, 2006] Markov, A. A. (2006). An example of statistical investigation of the text eugene onegin concerning the connection of samples in chains. *Science in Context*, 19(4):591–600.

[McCarthy, 1998] McCarthy, J. (1998). What is artificial intelligence.

[McCarthy and Feigenbaum, 1990] McCarthy, J. and Feigenbaum, E. A. (1990). In memoriam: Arthur samuel: Pioneer in machine learning. *AI Magazine*, 11(3):10–10.

[Mirhoseini et al., 2021] Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., et al. (2021). A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In Balcan, M. and Weinberger, K. Q., editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[Moerland et al., 2023] Moerland, T. M., Broekens, J., Plaat, A., Jonker, C. M., et al. (2023). Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118.

[Moore, 1998] Moore, G. E. (1998). Cramming more components onto integrated circuits. *Proc. IEEE*, 86(1):82–85.

[Nagabandi et al., 2018] Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. (2018). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 7559–7566. IEEE.

[Nasser et al., 2017] Nasser, H. M., Calu, D. J., Schoenbaum, G., and Sharpe, M. J. (2017). The dopamine prediction error: Contributions to associative models of reward learning. *Frontiers in Psychology*, 8.

[Niu et al., 2024] Niu, Y., Pu, Y., Yang, Z., Li, X., Zhou, T., Ren, J., Hu, S., Li, H., and Liu, Y. (2024). Lightzero: A unified benchmark for monte carlo tree search in general sequential decision scenarios. *Advances in Neural Information Processing Systems*, 36.

[OpenAI et al., 2019] OpenAI, :, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., d. O. Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning.

[Osband et al., 2020] Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvári, C., Singh, S., Roy, B. V., Sutton, R. S., Silver, D., and van Hasselt, H. (2020). Behaviour suite for reinforcement learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[Otto, 2021] Otto, F. (2021). *Model-Free Deep Reinforcement Learning—Algorithms and Applications*, pages 109–121. Springer International Publishing, Cham.

[Ouyang et al., 2022] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

[Papadimitriou and Tsitsiklis, 1987] Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of markov decision processes. *Mathematics of Operations Research*, 12(3):441–450.

[Parberry, 2017] Parberry, I. (2017). *Introduction to Game Physics with Box2D*. CRC Press.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z.,

DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

[Pérolat et al., 2022] Pérolat, J., Vylder, B. D., Hennes, D., Tarassov, E., Strub, F., de Boer, V., Muller, P., Connor, J. T., Burch, N., Anthony, T. W., McAleer, S., Elie, R., Cen, S. H., Wang, Z., Gruslys, A., Malysheva, A., Khan, M., Ozair, S., Timbers, F., Pohlen, T., Eccles, T., Rowland, M., Lanctot, M., Lespiau, J., Piot, B., Omidshafiei, S., Lockhart, E., Sifre, L., Beauguerlange, N., Munos, R., Silver, D., Singh, S., Hassabis, D., and Tuyls, K. (2022). Mastering the game of stratego with model-free multiagent reinforcement learning. *CoRR*, abs/2206.15378.

[Plaat, 2022] Plaat, A. (2022). *Deep Reinforcement Learning*. Springer.

[Plaat et al., 2023] Plaat, A., Kosters, W. A., and Preuss, M. (2023). High-accuracy model-based reinforcement learning, a survey. *Artif. Intell. Rev.*, 56(9):9541–9573.

[Pu et al., 2024] Pu, Y., Niu, Y., Ren, J., Yang, Z., Li, H., and Liu, Y. (2024). Unizero: Generalized and efficient planning with scalable latent world models. *arXiv preprint arXiv:2406.10667*.

[Puterman, 2014] Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[Racanière et al., 2017] Racanière, S., Weber, T., Reichert, D. P., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P. W., Hassabis, D., Silver, D., and Wierstra, D. (2017). Imagination-augmented agents for deep reinforcement learning. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5690–5701.

[Rosin, 2011] Rosin, C. D. (2011). Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230.

[Rumelhart et al., 1985] Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1985). Learning internal representations by error propagation.

[Russell and Norvig, 2010] Russell, S. J. and Norvig, P. (2010). *Artificial intelligence a modern approach*. Pearson Education, Inc.

[Samuel, 1967] Samuel, A. L. (1967). Some studies in machine learning using the game of checkers. ii—recent progress. *IBM Journal of research and development*, 11(6):601–617.

[Samuel, 2000] Samuel, A. L. (2000). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 44(1.2):206–226.

[Schaeffer et al., 2007] Schaeffer, J., Burch, N., Bjornsson, Y., Kishimoto, A., Muller, M., Lake, R., Lu, P., and Sutphen, S. (2007). Checkers is solved. *science*, 317(5844):1518–1522.

[Schaeffer et al., 1996] Schaeffer, J., Lake, R., Lu, P., and Bryant, M. (1996). Chinook the world man-machine checkers champion. *AI magazine*, 17(1):21–21.

[Schmidhuber, 2007] Schmidhuber, J. (2007). *Gödel Machines: Fully Self-referential Optimal Universal Self-improvers*, pages 199–226. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Schrittwieser et al., 2020] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

[Schrittwieser et al., 2021] Schrittwieser, J., Hubert, T., Mandhane, A., Barekatain, M., Antonoglou, I., and Silver, D. (2021). Online and offline reinforcement learning by planning with a learned model. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 27580–27591.

[Schulman et al., 2015] Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. (2015). Trust region policy optimization. In Bach, F. R. and Blei, D. M., editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1889–1897. JMLR.org.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423.

[Shannon, 1950] Shannon, C. E. (1950). Programming a computer playing chess. *Philosophical Magazine*, Ser.7, 41(312).

[Shapley, 1953] Shapley, L. S. (1953). Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100.

[Silver, 2015] Silver, D. (2015). Lectures on reinforcement learning.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489.

[Silver et al., 2017a] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017a). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

[Silver et al., 2021] Silver, D., Singh, S., Precup, D., and Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, 299:103535.

[Silver et al., 2017b] Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D. P., Rabinowitz, N. C., Barreto, A., and Degris, T. (2017b). The predictron: End-to-end learning and planning. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3191–3199. PMLR.

[Spaan, 2012] Spaan, M. T. (2012). Partially observable markov decision processes. In *Reinforcement learning: State-of-the-art*, pages 387–414. Springer.

[Sutton, 2019] Sutton, R. (2019). The bitter lesson. *Incomplete Ideas (blog)*, 13(1):38.

[Sutton, 1988] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44.

[Sutton, 1991] Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163.

[Sutton, 2004] Sutton, R. S. (2004). The reward hypothesis.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press.

[Sutton et al., 1999] Sutton, R. S., McAllester, D. A., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T. K., and Müller, K., editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 1057–1063. The MIT Press.

[Szepesvári, 2022] Szepesvári, C. (2022). *Algorithms for reinforcement learning*. Springer nature.

[Szepesvári, 2020] Szepesvári, C. (2020). Theoretical rl lecture notes. University of Alberta, CMPUT 653, Accessed: 2023-08-01.

[Tassa et al., 2018] Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T. P., and Riedmiller, M. A. (2018). Deepmind control suite. *CoRR*, abs/1801.00690.

[Tesauro et al., 1995] Tesauro, G. et al. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

[Thorndike, 1911] Thorndike, E. L. (1911). *Animal intelligence: Experimental studies*. Transaction Publishers.

[Todorov, 2006] Todorov, E. (2006). Optimal control theory.

[Todorov et al., 2012] Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pages 5026–5033. IEEE.

[Tolman, 1948] Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological review*, 55(4):189.

[Towers et al., 2024] Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., et al. (2024). Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*.

[Towers et al., 2023] Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G. d., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Shen, A. T. J., and Younis, O. G. (2023). Gymnasium.

[Turing, 1950] Turing, A. M. (1950). I.—Computing Machinery and Intelligence. *Mind*, LIX(236):433–460.

[Turing, 1953] Turing, A. M. (1953). Digital computers applied to games. *Faster than thought*.

[v. Neumann, 1928] v. Neumann, J. (1928). Zur theorie der gesellschaftsspiele. *Mathematische annalen*, 100(1):295–320.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

[Vinyals et al., 2019] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

[Watkins and Dayan, 1992] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8:279–292.

[Watkins, 1989] Watkins, C. J. C. H. (1989). Learning from delayed rewards.

[Wei et al., 2023] Wei, R., Lambert, N., McDonald, A., Garcia, A., and Calandra, R. (2023). A unified view on solving objective mismatch in model-based reinforcement learning. *arXiv preprint arXiv:2310.06253*.

[Werbos, 1990] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[White, 1993] White, D. J. (1993). A survey of applications of markov decision processes. *Journal of the operational research society*, 44(11):1073–1096.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8:229–256.

[Williams and Peng, 1991] Williams, R. J. and Peng, J. (1991). Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268.

[Ye et al., 2021] Ye, W., Liu, S., Kurutach, T., Abbeel, P., and Gao, Y. (2021). Mastering atari games with limited data. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 25476–25488.

Appendix A

Appendix

A.0.1 PVC-A2C Supplementary Data

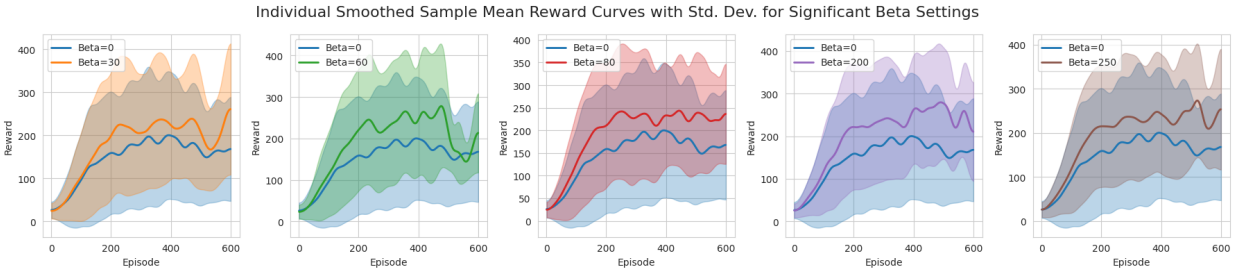

Figure A.1: One-to-one comparison of baseline and significant β PVC-A2C agents' standard deviation.

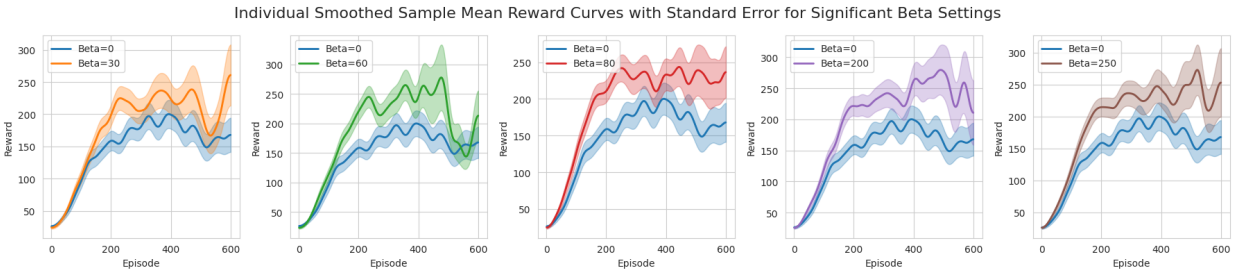

Figure A.2: One-to-one comparison of baseline and significant β PVC-A2C agents' mean and standard error.