university of groningen / faculty of science and engineering

ARTIFICIAL INTELLIGENCE

MASTER'S THESIS

# When to Explore: Guiding Deep Reinforcement Learning with State Counts and Value State Prediction Errors for Efficient Learning


**Author**
Marius Captari

**Internal Supervisor**
Dr. Matthia Sabatelli
Artificial Intelligence
University of Groningen

**External Supervisor**
Remo Sasso
Artificial Intelligence
Queen Mary University of London


January 4, 2024

# Contents

**Abstract**

Despite the considerable attention given to the questions of how much and how to explore in Deep Reinforcement Learning (DRL), the investigation into when to explore remains relatively unexplored. While more sophisticated exploration strategies exhibit success in specific, often sparse reward environments, existing simpler approaches, such as $\epsilon$-greedy, persist in outperforming them across a broader spectrum of domains. The appeal of these simpler strategies lies in their stationarity, which supports the agent's learning stability and mitigates hyper parameter complexity. The downside is that these methods are essentially a blind switching mechanism, which completly disregard the agent's internal state. In this research we introduce a novel exploration strategy that combines the agent's value state prediction errors with counts of the current hashed state, providing a nuanced approach to guide the timing of exploration decisions. Experiments conducted on a range of Atari games reveal that the proposed strategy outperforms traditional methods, facilitating faster learning rates, even when compared to using each exploration signal in isolation. This study contributes to a more intricate understanding of exploration dynamics, underscoring the significance of temporal considerations in the exploration-exploitation dilemma within DRL.

# *Acknowledgements*

First and foremost, I extend my appreciation to my main supervisor, Matthia, who's guidance and countless hours spent discussing research ideas in the office have been instrumental in shaping the direction of my research.

I would also like to express my gratitude to my second supervisor, Remo, for the insightful ideas and constructive feedback provided during our meetings.

To my parents, to whom I am deeply thankful for the opportunity and encouragement they have given me to pursue my studies abroad.

A special thanks goes out to Frederika and Guus, whose warmth, hospitality, and support have made a significant difference during my time in the Netherlands.

Lastly, I want to acknowledge my friends for their ability to steer conversations away from AI-related topics, providing a much-needed balance in my life.

# Chapter 1

# Introduction

Reinforcement Learning (RL) is a paradigm within machine learning where agents learn how to behave in an environment by taking actions and receiving rewards or penalties in return. This dynamic forms the basis of a trial-and-error learning process, allowing agents to adapt and optimize their behaviors over time. Rooted in the principles of behavioral psychology, RL operates on the fundamental idea that strategies, termed policies, which yield higher rewards should be reinforced, or made more probable (Sutton and Barto, 2018).

Over the decades, RL has manifested its prowess in numerous applications, from game playing to robotics, displaying its flexibility and potential. Nonetheless, traditional RL often faced challenges when dealing with high-dimensional state spaces or environments with sparse and delayed rewards.

The advent of Deep Learning marked a transformative phase in the field of artificial intelligence, particularly with the integration of Deep Learning into RL, forming Deep Reinforcement Learning (DRL). DRL leverages deep neural networks as function approximators for policies or value functions in model-free RL, whereas in model-based approaches neural networks are used for learning the environment dynamics, allowing them to create predictive models of how the environment will respond to different actions. These neural architectures excel at automatically extracting hierarchies of features from raw data, making them particularly suited for complex, high-dimensional environments.

Seminal works like the Deep Q-Networks (DQN) from DeepMind epitomized the potential of DRL, mastering a wide array of Atari 2600 games and frequently achieving performances surpassing human experts (Mnih, Kavukcuoglu, Silver, Rusu, et al., 2015a). Later, the same group achieved another breakthrough with AlphaGo, which defeated the world Go champion, pushing the boundaries of what was believed to be achievable with RL in game playing (Silver et al., 2016). In parallel, OpenAI demonstrated the power of RL in the domain of e-sports, training agents that could compete with human professionals in the complex game of Dota 2 (Berner et al., 2019). In robotics, DRL has been instrumental in empowering robots to execute complex tasks, previously unattainable with conventional control methodologies (Andrychowicz et al., 2020).

## 1.1 The Exploration-Exploitation Dilemma

Despite its successes, DRL still carries many of the main challenges of classic RL. Central to these challenges is the exploration-exploitation trade-off. Every agent encounters this dilemma: should it rely on its current knowledge and take actions that are known to yield good rewards (exploitation), or should it try new actions, potentially discovering better strategies but risking lower immediate rewards (exploration)?

Historical methods like the $\epsilon$-greedy approach, where the agent occasionally selects a random action over the best-known one, or Boltzmann exploration, where action probabilities are derived from their value estimates, have long been the standard. However, these methods, while effective in many scenarios, may not be the most efficient or adaptive, especially in intricate environments where the learning landscape is rugged and rewards can be sparse or deceptive. On the other hand, such naive methods are general since they do not make strong assumptions about the underlying environment, making them more appealing when compared to more complex variants.

In the domain of DRL, considerable effort has been directed toward understanding the optimal balance of exploration (*how much*) and devising techniques for executing exploratory actions (*how*). This encompasses studies on modulating the exploration rate over time and optimizing action selection strategies for better learning outcomes (Auer, 2002; Tokić, 2010; Osband, Blundell, Pritzel, and Roy, 2016). There is a rich body of work exploring various strategies, such as stochastic action selection, optimistic initialization, and intrinsic motivation driven exploration (Pathak, Agrawal, Alexei A. Efros, et al., 2017b; Burda, Harri Edwards, et al., 2018; M. Bellemare et al., 2016). However, the aspect of timing in exploration (*when*) has received far less attention. Investigating this under-explored area could lead to the development of more nuanced and context-sensitive exploration techniques in DRL.

## 1.2 Significance of the Study

The exploration-exploitation dilemma is not merely an academic conundrum; it has tangible implications in real-world DRL applications. Ensuring efficient exploration can drastically reduce the learning time, making DRL more feasible for time-sensitive applications. Moreover, in environments where exploration can be expensive or dangerous (like real-world robotics or finance), ensuring agents make informed decisions about when and for how long to explore while balancing risk is paramount.

Striking a balance between simple, more naive methods, and more complex exploration mechanisms in order to improve current DRL agents exploration effectiveness is the main goal of this thesis.

## 1.3 Proposed Solution

In this work, we primarily focus on the *when* aspect of exploration in DRL, with the goal of providing current agents with more intentional exploration strategies. Our approach is based on two distinct informed triggers.

The first trigger, termed the Value Promise Discrepancy (VPD), is inspired by insights from Pislar et al., 2021. VPD assesses exploration decisions by evaluating the divergence between an agent's prior value estimate of a state-action pair and the actual cumulative reward received over a set period. This assessment helps agents gauge the accuracy of their internal value predictions, informing their decision to explore further or exploit current knowledge. However, VPD may not adequately address issues like sparser environments and the novelty of a state. To remedy this, we introduce a second trigger for a richer exploration analysis.

This second trigger involves mapping states to hash codes using the SimHash algorithm (Charikar, 2002), allowing for the counting of individual hash occurrences. The counts are used to compute an exploration bonus, ensuring that frequently visited states are not overly prioritized and promoting a balanced exploration-exploitation

trade-off. Unlike in the original work (Tang et al., 2017) where this state bonus is added as an external reward, we propose tracking this value as an informed trigger, preserving the original reward structure and avoiding issues like novelty loops and reward distortion.

By combining these signals, we aim to utilize both long-term uncertainty and state-specific novelty to guide exploration timing more effectively.

Another aspect we address is setting a threshold for triggering exploration. Incorporating the concept of homeostasis, as described in Pislar et al., 2021, complements our approach. Homeostasis, borrowed from neuroscience and applied to reinforcement learning, focuses on maintaining stability through internal regulation. Due to the variability in signal scales across different domains and training durations, setting naive threshold hyperparameters can be impractical. However, homeostatic principles, applied as a binary switching mechanism, adapt the threshold based on recent signal values, ensuring a consistent average target rate. This adaptive thresholding method decouples the target rate of switching from the scales of the trigger signals, simplifying tuning and enhancing robustness.

Simpler exploration strategies like $\epsilon$-greedy rely on predefined rules for exploration, which may not always adapt to changing environments or the agent's learning progress. Our approach, using informed triggers combined with homeostatic principles, represents a more dynamic and responsive method. These triggers adjust exploration based on the agent's performance predictions and the novelty of encountered states, aiming to facilitate more deliberate and effective exploration strategies.

Further enriching the exploration paradigm, this thesis doesn't solely focus on the question of *when* to explore but also delves into *how long* such explorations should persist. Drawing inspiration from the concept of Temporally-Extended Exploration (Dabney, Ostrovski, and Barreto, 2020), this work proposes that once an exploratory action is chosen, it should be sustained for a duration determined randomly from a predefined distribution. Intriguingly, adopting distributions inspired by ecological studies on animal foraging behavior has previously manifested in notably robust performance.

## 1.4 Research Questions

In this thesis we aim to answer the following research questions:

- How does the proposed exploration strategy compare to traditional methods in terms of enhancing the performance of DRL agents?
- Is the learning performance of DRL agents improved by combining both exploration signals, as opposed to using each signal independently?
- What impact does the introduction of extended exploration periods have on the learning efficiency and effectiveness of DRL agents?

## 1.5 Overview of Experiments

To evaluate the effectiveness of the newly proposed exploration strategy, a comprehensive series of experiments has been conducted. These experiments employ a Rainbow agent, a state-of-the-art DRL architecture, across a diverse set of environments. This includes both classic control tasks, which represent simpler scenarios, and more complex

Atari games, offering a broader challenge spectrum. We compare the agent's performance with established exploration benchmarks such as $\epsilon$-greedy, Boltzmann exploration, and Noisy Networks, to gauge the relative improvements offered by our strategy. Furthermore, we incorporate the concept of temporally-extended actions into these experiments. This addition aims to examine its influence on exploration dynamics, specifically assessing how extending the duration of actions affects the overall learning performance and exploration efficiency of the agent.

In addition to the design and methodology of these experiments, it is essential to acknowledge the instrumental role of the Hábrók high-performance computing (HPC) cluster provided by the University of Groningen's Center for Information Technology (CIT). Access to this HPC cluster was crucial for the computational demands of our research, allowing for the extensive training and simulation of DRL agents. We extend our gratitude to the CIT for their support and for providing access to the Hábrók cluster, which was indispensable in carrying out this work.

## 1.6 Thesis Structure

This thesis is structured to offer a clear understanding of relevant exploration strategies in deep reinforcement learning (DRL), as well as present the newly proposed method. Chapter 1 introduces the research, laying out the primary objectives and the importance of exploration in DRL. Chapter 2 dives into a literature review, tracing the progression of exploration strategies in DRL. The key theories and concepts are laid out in Chapter 3, which covers the theoretical background. Chapter 4 goes over the proposed exploration strategy, whereas in Chapter 5, the methods and experiments are detailed, with a specific focus on the technical implementation of the Value Promise Discrepancy (VPD) and SimHash state counts as well as other relevant concepts. The findings from these experiments are then analyzed and discussed in Chapter 6. Finally, Chapter 7 provides a conclusion to the thesis, summarizing the key insights and suggesting potential avenues for future research.

# Chapter 2

# Literature Review

This chapter briefly discusses the growth of DRL algorithms, highlighting the key role of Deep Q-Networks (DQN) and how it evolved into the Rainbow agent. We tackle the big question in RL: how to balance trying new things (exploration) with sticking to what works (exploitation), discussing techniques from Bayesian methods to Thompson sampling. We also look at newer ideas like counting-based exploration, switching modes, and persistent exploration. The chapter concludes with an overview of the Dopamine RL framework, a tool utilized in our study for implementing and testing our proposed strategies.

## 2.1 Evolution of Deep Reinforcement Learning

The many recent successes in scaling reinforcement learning (RL) to complex sequential decision-making problems were kick-started by the Deep Q-Networks algorithm (DQN; (Mnih, Kavukcuoglu, Silver, Graves, et al., 2013; Mnih, Kavukcuoglu, Silver, Rusu, et al., 2015b)). This model-free approach, combining Q-learning with convolutional neural networks and experience replay enabled it to learn, from raw pixels, how to play many Atari games at human-level performance.

Since then, many extensions have been proposed that enhance its speed or stability. Double DQN (DDQN; (Hasselt, Guez, and Silver, 2016)) addresses an overestimation bias of Q-learning (Hasselt, 2010), by decoupling selection and evaluation of the bootstrap action. Deep Quality-Value Learning (DQV; (Sabatelli et al., 2018)) aims to address this same problem by training a value network which is in turn used for training the final quality value network, resulting in better value estimates and faster convergence. Prioritized experience replay (Schaul et al., 2015) improves data efficiency, by replaying more often transitions from which there is more to learn. The dueling network architecture (Wang et al., 2016) helps to generalize across actions by separately representing state values and action advantages. Learning from multi-step bootstrap targets (Sutton, 1988; Sutton and Barto, 1998), as used in A3C (Mnih, Adria Puigdomenech Badia, et al., 2016), shifts the bias-variance trade-off and helps to propagate newly observed rewards faster to earlier visited states. Distributional Q-learning (Marc G Bellemare, Dabney, and Remi Munos, 2017) learns a categorical distribution of discounted returns, instead of estimating the mean. Noisy DQN (Fortunato et al., 2017) uses stochastic network layers for exploration. This list is, of course, far from exhaustive.

Each of these algorithms enables substantial performance improvements in isolation. Since they do so by addressing radically different issues, and since they build on a shared framework, they could plausibly be combined. In some cases this has been done: Prioritized DDQN and Dueling DDQN both use double Q-learning, and Dueling DDQN was also combined with prioritized experience replay. The Rainbow agent, proposed by Hessel et al., 2018, combines all of these aforementioned improvements to

create a powerful and versatile DRL algorithm. In fact, their combination resulted in a new state-of-the-art results on the benchmark suite of 57 Atari 2600 games from the Arcade Learning Environment (Marc G Bellemare, Naddaf, et al., 2013), both in terms of data efficiency and of final performance. In this thesis, the Rainbow agent is used as the basis for all the benchmarks and experiments, showcasing its applicability and effectiveness across various problem domains.

On the other side of things, model-based DRL has emerged as a significant contributor to the field, running in parallel with the advancements in model-free DRL. The main difference between the model-free and model-based approaches lies in their learning strategies, where model-free methods directly optimize policies or value functions from interactions, while model-based methods optimize a learned model of the environment to guide decision-making.

Pioneering works, such as those by Janner, Fu, and Zhang, 2019 and Parisotto and Salakhutdinov, 2017 have demonstrated the potential of model-based approaches. Model-based DRL offers a unique approach by harnessing neural networks to create predictive models of the environment dynamics, enabling agents to plan and make informed decisions. This approach significantly improves sample efficiency, as agents can simulate and learn from hypothetical experiences before interacting with the real environment. While model-free DRL has dominated the scene with its impressive results in various benchmark tasks, model-based DRL's significance lies in its potential to reduce data requirements and enable faster learning in situations where data collection is costly or impractical.

## 2.2 The Complex Landscape of Exploration

At its core, RL presents the twin challenges of temporal credit assignment and exploration. The agent must accurately, and efficiently, assign credit to past actions for their role in achieving some long-term return. However, to continue improving the policy, it must also consider behaviours it estimates to be sub-optimal. This leads to the well-known exploration-exploitation trade-off.

Because of its central importance in RL, exploration has been among the most studied topics in the field. In finite state-action spaces, the theoretical limitations of exploration, with respect to sample complexity bounds, are fairly well understood (Azar, Osband, and Remi Munos, 2017; Dann, Neumann, and Peters, 2017). However, these results are of limited practical use for two reasons. First, they bind sample complexity by the size of the state-action space and horizon, which makes their immediate application in large-scale or continuous state problems difficult. Second, these algorithms tend to be designed based on worst-case scenarios, and can be inefficient on problems of actual interest. Bayesian RL methods for exploration address the explore-exploit problem integrated with the estimation of the value-function itself (Kolter and Ng, 2009). Generally such methods strongly depend upon the quality of their priors, which can be difficult to set appropriately. Thompson sampling methods (Thompson, 1933; Osband and Van Roy, 2013) estimate the posterior distribution of value-functions, and act greedily according to a sample from this distribution. As with other methods which integrate learning and exploration into a single estimation problem, this creates non-stationary, but temporally persistent, exploration. Other examples of this type of exploration strategy include randomized prior functions (Osband and Aslanides, 2018), uncertainty Bellman equations (O'Donoghue et al., 2018), Noisy Nets (Fortunato et al.,

2017), and successor uncertainties (Janz, Seijen, and Sutton, 2019). Although quite different from each other, they share key commonalities: non-stationary targets, temporal persistence, and exploration based on the space of value functions.

At the other end of the spectrum, there have recently been successful attempts to design algorithms with specific problems of interest in mind. Certain games from the Atari-57 benchmark (e.g. MONTEZUMA'S REVENGE, PITFALL!, PRIVATE EYE) have been identified as 'hard exploration games' (M. Bellemare et al., 2016), attracting the attention of the research community, leading to significant progress in terms of performance (Ecoffet et al., 2019; Burda, Harrison Edwards, et al., 2018). On the downside, these results have been usually achieved by algorithms with little or no theoretical grounding, adopting specialized inductive biases, such as density modeling of images (M. Bellemare et al., 2016; Ostrovski et al., 2017), error-seeking intrinsic rewards (Pathak, Agrawal, Alexei A Efros, et al., 2017a; Adrià Puigdomènech Badia et al., 2020), or perfect deterministic forward-models (Ecoffet et al., 2019). Generally, such algorithms are evaluated only on the very domains they are designed to perform well on, raising questions of generality. Recent empirical analysis showed that some of these methods perform similarly to each other on hard exploration problems and significantly under-perform $\epsilon$-greedy otherwise (Ali Taïga et al., 2020). One explanation is that complex algorithms tend to be more brittle and harder to reproduce, leading to lower than expected performance in follow-on work. However, these results also suggest that much of the recent work on exploration is over-fitting to a few hard games.

Another line of research takes a different approach to the exploration problem. Rather than designing algorithms with specific challenges in mind, it seeks to generalize and unify existing approaches. Randomized least-squares value iteration (RLSVI) (Osband, Russo, and Van Roy, 2017) generalizes Thompson sampling to Q-functions, and builds on the insights of posterior sampling for exploration (Strens, 2000; Osband, Blundell, Pritzel, and Van Roy, 2016). RLSVI has strong theoretical guarantees under certain conditions, but requires a factored state space, and its application to deep RL has so far been limited. A unifying perspective on exploration based on the principle of 'optimism in the face of uncertainty' is given by (Chen and Lihong, 2020). This perspective attempts to unify a large range of algorithms, from bandit algorithms to deep RL.

There is another emerging line of work that borrows ideas from human and animal learning. Humans and animals explore their environments in ways that often seem purposeful, curious, or even playful (Cohen, McClure, and Yu, 2007). There is increasing interest in understanding how such exploratory behaviours arise and how they might be incorporated into artificial agents. Some promising directions include understanding the neural basis of exploration, and the use of neuromodulators like dopamine in modulating exploration-exploitation trade-offs (Huber, 1991; Pathak, Agrawal, Alexei A Efros, et al., 2017a; Conti et al., 2017; Burda, Harrison Edwards, et al., 2018; Osband, Moerland, and Van Roy, 2014). In the natural world, exploratory behaviour is temporally structured: humans for example choose to explore in a more directed way when they detect high levels of uncertainty (Gershman, 2018a) or adjust the randomness of their choice as a function of the level of uncertainty (Gershman, 2018b). Similarly, monkeys use directed exploration to manage the trade-off between efficient foraging and exploration (Costa et al., 2019). And there are changes in exploration behaviour, for instance, with pathological states. Schizophrenia patients, for example, show increased randomness in their choices (Waltz et al., 2020; Cathomas et al., 2021).

To date, RL has primarily employed rather monolithic exploration techniques that,

while achieving substantial success in specific applications, may not capture the richness of exploration behaviours evident in real-world agents. Future developments may aim to incorporate such diverse strategies, tailored for specific application domains or designed for broad generality.

### 2.2.1 Value Promise Discrepancy

The Value Promise Discrepancy (VPD) is rooted in the idea of dissecting the difference between promised and actual rewards. The primary idea is that by investigating the difference between expected and achieved values, one can gain insights into areas where the agent's learning may be lagging or where it may be over-optimistic about certain actions or states. Pislar et al., 2021 delves deep into this concept, suggesting potential modifications in reward structures or learning processes to better align with the true value promises of an environment. This understanding can play a pivotal role, especially when designing agents for real-world scenarios where over-optimism can lead to catastrophic failures.

VPD is primarily viewed as a proxy for the agent's uncertainty in the literature (Schulz et al., 2019). The general sentiment is that a higher uncertainty value should encourage the agent towards more exploration. However, taking a high VPD value as a direct indicator of uncertainty might be an over simplification.

While a pronounced VPD might suggest an agent's unfamiliarity or misjudgment about certain aspects of its environment, it can be misleading in several contexts. An elevated VPD might merely indicate that the agent is in the early stages of its learning process, where prediction errors are naturally expected to be high. Another such context might be when certain state spaces might inherently have higher volatility or unpredictability, leading to elevated VPD values. Notice also that the VPD value does not capture other nuances such as the novelty of a state, which is a very relevant metric in the context of exploration.

Relying exclusively on VPD can therefore mask the underlying reasons for high prediction errors and miss out on other crucial information. To augment this uncertainty proxy, we explore incorporating an additional trigger that evaluates the novelty of the agent's current state, rather than just the prediction error.

### 2.2.2 Count-Based Exploration and SimHash

Count-based exploration is a foundational method in RL, leveraging the frequency of state visits to guide the agent's exploratory behaviour. The idea behind this approach is that states visited less frequently may offer greater informational or reward opportunities, making them prime candidates for exploration. This approach has seen varied applications, from simpler, discrete state spaces to more complex and high-dimensional environments (Strehl and Littman, 2008; Tang et al., 2017).

As RL agents typically operate in high-dimensional spaces, traditional tabular methods for tracking state visits are impractical. Hence, approximation techniques, like hashing, are often employed to achieve this goal (M. Bellemare et al., 2016).

Among the various hashing techniques, SimHash stands out due to its simplicity and effectiveness (Charikar, 2002). By generating a locality-sensitive hash for input states, SimHash offers a compact representation that gauges state novelty. The essence of SimHash lies in projecting states via random hyperplanes to produce binary hashes. Tracking each hash's frequency then allows the derivation of an exploration bonus, with rarer states yielding a more significant incentive for exploration (Tang et al., 2017).

Traditional count-based methods, while effective in simpler domains, struggle in environments with vast state spaces, continuous states, or non-stationary dynamics. SimHash, with its locality-sensitive hashing, alleviates these challenges by offering a scalable and adaptive means to gauge state novelty. The hashing mechanism inherently clusters similar states, providing a more generalized count and thereby smoothing the exploration landscape. Furthermore, the hashing function can be modified so that a proper balance between generalization across states, and distinguishing between states is achieved.

### 2.2.3 Persistent Exploration: Extended $\epsilon$-greedy

One critique of the standard $\epsilon$-greedy strategy is its non-persistent nature of exploration. With $\epsilon$-greedy, an agent tends to not deviate from the default trajectory for extended periods, making it challenging to escape local optima. This issue emphasizes the need for an exploration strategy that retains the simplicity of $\epsilon$-greedy but offers persistent exploration to navigate challenges effectively.

A recent approach seeks to bridge this gap by proposing a temporally extended form of $\epsilon$-greedy. This method retains the simplicity of the original algorithm but reduces its inductive bias that favors transitions that are likely under the policy being learned by introducing persistent actions. The core idea is to let the agent repeat the sampled exploratory action for a random duration, rather than changing the action every time step. This approach ensures that the exploration is more sustained, allowing the agent to deviate further from its default trajectory and potentially escape local optima (Dabney, Ostrovski, and Barreto, 2020).

Furthermore, this persistent approach to exploration maintains the beneficial properties of $\epsilon$-greedy. It remains stationary, meaning its operation does not rely on learning progress, ensuring stability. It also provides full coverage of the space of possible trajectories, preserving the essence that no solutions are excluded from consideration, a fundamental principle in RL (Singh et al., 2000).

## 2.3 Dopamine RL Framework

Dopamine, made by Google, is a tool for research in DRL (Castro et al., 2018). It is designed to be both flexible and easy to replicate. It focuses on being stable, simple, and having parts that can be easily swapped out.

Because of its design, researchers can easily change different parts of the system, from the structure of the agent to how they learn. This makes it easier to try out new ideas and understand new methods. As we see more advancements in the field of DRL, tools like Dopamine are crucial for making research smoother and more consistent.

In this thesis, we use Dopamine as our main development framework. It helps us set up, test, and compare different exploration algorithms, and ensures that our results can be replicated by other researchers.

# Chapter 3

# Theoretical Background

Reinforcement Learning (RL), a cornerstone of modern AI, has significantly influenced domains ranging from gaming to robotics. As Deep Learning began to merge with RL, innovative methods emerged that enhance the ability of agents to comprehend and navigate their environments. In this chapter, we will dive deep into the foundational concepts of RL, extending our discussion to the nuances of DRL techniques.

We begin by revisiting the fundamental principles of RL. We start by unpacking the concept of Markov Decision Processes (MDPs), examining how agents interact with and perceive their environments. From there, the focus shifts to function approximators in RL, including Deep Q Networks (DQN) and their subsequent advancements, culminating in the Rainbow agent – the basis for our experiments.

Additionally, this chapter delves into the relevant exploration strategies used for our research. These strategies are paramount as they dictate the learning trajectory of an agent within its environment. By the end of this chapter, our goal is to provide a comprehensive overview of the background on the different components used for the proposed exploration strategy.

## 3.1 Reinforcement Learning

Reinforcement Learning is a distinctive paradigm of machine learning that diverges from the conventional supervised and unsupervised learning approaches. In RL, agents learn how to behave in an environment by performing actions and receiving rewards in return, instead of learning from pre-labeled data or inherent data structures. At every time step, the agent chooses an action based on a policy, interacts with the environment, and receives feedback in the form of a reward. The agent's objective is to develop a policy that maximizes its expected cumulative reward over time, often referred to as the return.

This dynamic of action and feedback is inspired by behavioural psychology, reflecting the way living beings learn from their experiences. The agent's learning process revolves around the exploration-exploitation trade-off: determining when to explore new actions to discover their effects or exploit known actions that yield the highest rewards.

RL is especially useful for situations where we don't have clear guidance on the right decisions, but the choices made can have long-term consequences. This includes applications such as game playing, where agents learn strategies to defeat opponents, or robotics, where physical actions taken by a robot in its environment can lead to varying outcomes. The RL framework provides a foundation to train agents in these complex settings, enabling them to autonomously discover optimal strategies.

## 3.2 Markov Decision Processes

At the heart of RL lies the concept of Finite Markov Decision Processes (MDPs) (Puterman, 1994). An MDP provides a mathematical framework for describing the dynamics of an environment in RL, encapsulating the interaction between an agent and its environment.

The agent and environment engage in a continuous interaction across a sequence of discrete time steps $t = 0, 1, 2, \dots$. At each step $t$, the agent receives a representation of the environment's state, denoted as $s_t$ which belongs to a set $\mathcal{S}$. Based on this state representation, the agent selects an action, $a_t$, from the set $\mathcal{A}$. As a consequence of its action, the agent then obtains a numerical reward, $r_t$, and transitions to a new state, $s_{t+1}$. This interaction leads to trajectories that appear as: $s_0, a_0, r_1, s_1, a_1, r_2, \dots$.

Formally, an MDP is described by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$ in which:

- $\mathcal{S}$ is the state space: A finite set of states the agent might inhabit within the environment.
- $\mathcal{A}$ is the action space: A finite set of actions available to the agent, contingent on the current state.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function: It gives the expected reward for executing action $a_t$ in state $s_t$ at timestep $t$.
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability: It denotes the likelihood of transitioning from state $s$ to state $s_{t+1}$ given the action $a_t$.
- $\gamma$ is the discount factor: A scalar in the interval [0,1] that determines the present value of impending rewards.

A notable characteristic of the environment is its Markovian nature. This implies that the likelihood of transitioning to state $s_{t+1}$ is solely determined by the present state $s_t$ and the action $a_t$:

$$p(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \dots) = p(s_{t+1} \mid s_t, a_t), \tag{3.1}$$

similarly, the probability of observing a reward $r_t$ only depends on the current state $s_t$ and action $a_t$:

$$p(r_t \mid s_t, a_t, \dots s_1, a_1) = p(r_t \mid s_t, a_t). \tag{3.2}$$

An overview of this agent-environment interaction loop can be found in Figure 3.1.

### 3.2.1 Episodes, Rewards, and Returns

In RL, an episode encompasses a series of states, actions, and rewards from the start to the termination of an agent's interaction with the environment. An episode typically begins from an initial state, proceeds through a number of state-action transitions, and culminates in a terminal state. This might represent the end of a game, the completion of a task, or any other predefined termination criterion.

The primary objective of the agent in RL is to maximize its expected return. The return, denoted $G_t$, at time $t$ is the cumulative reward the agent expects to obtain from that time onward, considering both immediate and future rewards. Formally, it's represented as:

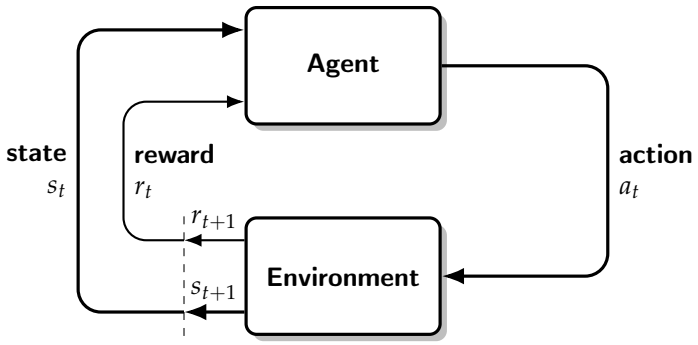


FIGURE 3.1: The agent–environment interaction in a Markov decision process.

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \end{aligned} \tag{3.3}$$

where $\gamma$ is the discount factor introduced in the MDP framework. It ensures that rewards obtained immediately are generally valued more than those received in the distant future. The discount factor lies between 0 and 1; a value close to 0 makes the agent myopic, focusing mainly on immediate rewards, while a value closer to 1 makes the agent more foresighted, valuing distant rewards nearly as much as immediate ones.

By maximizing the expected return, the agent learns to make decisions that account for both immediate benefits and long-term consequences. Balancing these two aspects is a core challenge in RL, and the formulation of the return ensures that the agent has a consolidated metric to guide its learning and decision-making process.

As a direct continuation of the presentation of MDPs, this understanding of episodes and returns further underscores the dynamic and forward-looking nature of RL. An agent is not just reacting to the present environment but is always strategizing for the future, constantly recalculating its expected returns to make optimal decisions.

### 3.2.2 Policies and Value Functions

Value functions are central to RL, estimating the expected return an agent can expect starting from a particular state or after taking an action. The quality of states or actions is gauged by the future rewards one can expect to receive.

A policy, denoted by $\pi$, defines the agent's behaviour. It is a mapping from states to the probabilities of selecting each possible action. If the agent is following policy $\pi$ at time $t$, then $\pi(a|s)$ is the probability that $a_t = a$ given $s_t = s$.

#### State-value Function

The state-value function, $V^\pi(s)$, for a policy $\pi$ provides the expected return when starting in state $s$ and following $\pi$ thereafter:

$$V^\pi(s) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, \pi]. \tag{3.4}$$

#### Action-value Function

The action-value function, $Q^\pi(s,a)$, for a policy $\pi$ provides the expected return starting from state $s$, taking action $a$, and then following $\pi$:

$$Q^\pi(s,a) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi]. \tag{3.5}$$

#### Advantage Function

By taking the difference between the action-value function and the state function for a certain action $a$ in a state $s$, we get the advantage function, which is defined as:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s). \tag{3.6}$$

The advantage function quantifies the relative quality of a particular action $a$ in a given state $s$ by comparing it to the average action's effectiveness in the same state. A positive value of the advantage function indicates that action $a$ is better than the average action in state $s$, while a negative value implies it is less effective.

The advantage function will be further discussed in the context of the dueling architecture used in the Rainbow agent in Section 3.3.2.

### 3.2.3 Optimal Policies and Optimal Value Functions

In RL we are interested in finding the best policy, also known as the optimal policy $\pi^*$. A policy $\pi$ is defined to be better than or equal to a policy $\pi'$ if its expected return is greater than or equal to that of $\pi'$ for all states. Such optimal policy realizes what is called the optimal state-value function $V^*$, which can be defined as:

$$V^*(s) = \max_\pi V^\pi(s), \text{ for all } s \in \mathcal{S}, \tag{3.7}$$

and the optimal state-action function $Q^*$ defined as:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a), \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}. \tag{3.8}$$

## 3.3 Solution Methods

RL solutions aim to determine the best policy that maximizes the expected cumulative reward. There two main methods to achieve this is by either learning a policy directly, without any leaning any value function (policy-search) or by learning the optimal state-action value function, and then acting greedly according to our learned policy (value-based). We will focus on the value-based approach, as it relates to the final algorithm used to train our agent.

There are two main approaches when dealing with value-based model-free RL, with tabular methods being one of the earliest. However, when the state or action space is large or continuous, tabular methods fall short, necessitating more sophisticated approaches. One such technique is function approximation, which is pivotal in extending RL to complex environments.

### 3.3.1 Tabular Methods

In environments with a small, finite state and action space, the value of each state-action pair can be stored in a table. This table gets updated iteratively based on the agent's experiences until convergence. One of the most popular algorithms utilizing this tabular approach is Q-learning, which stands as the foundation for the more sophisticated DQN algorithm and its extensions like Rainbow.

Q-learning is known as an off-policy method. In RL, there's a crucial distinction between on-policy and off-policy methods:

- **On-Policy Methods**: These algorithms learn the value of the policy being executed, meaning they learn about the current policy while executing it. A notable example is SARSA.
- **Off-Policy Methods**: These algorithms learn the value of a policy different from the one they are executing. Q-learning is a primary example. It updates its Q-values using the optimal action for the next state (i.e., the action that has the maximum current estimate of its Q-value), allowing it to learn about the optimal policy while following an exploratory or different policy.

#### Q-learning

Q-learning is a classic algorithm in RL that falls under the category of off-policy methods. It is particularly effective when the state and action spaces are small or finite. Q-learning aims to learn the optimal action-value function, denoted as $Q^*$. Having learnt the optimal action-value function, it is straightforward to derive the optimal policy, as we can simply select the action which has the highest value:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a) \text{ for all } s \in \mathcal{S}. \tag{3.9}$$

The optimal action-value function satisfies what is known as the Bellman equation, which expresses how the action-value function $Q^\pi(s, a)$ for a given policy $\pi$ can be computed recursively by considering the expected rewards and future action-values when transitioning from the current state-action pair $(s, a)$ to the next state $s_{t+1}$ and action $a_{t+1}$. Formally, the Bellman equation is defined as:

$$Q^\pi(s_t, a_t) = \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1} \mid s_t, a_t) \left( \mathcal{R}(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q^\pi(s_{t+1}, a_{t+1}) \right). \tag{3.10}$$

Q-learning approximates this equation using a temporal difference (TD) learning approach. It calculates the difference between the current estimate $Q(s_t, a_t)$ and the target value $r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ and updates the estimate towards this target. Over time, as the agent explores the environment and gathers more experiences, the Q-values converge to the optimal action-value function $Q^*$ (Sutton and Barto, 1998). The Q-value estimates are updated according to the following rule:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \left[ r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t) \right], \tag{3.11}$$

where $\alpha$ represents the learning rate. Notice the use of the max operator in the calculation of the target value, making Q-learning an off-policy algorithm. This off-policy

nature, which permits learning from past experiences, is crucial, especially in scenarios where extensive exploration or reuse of experiences is necessary. As we will see in the following section with DQN's experience replay, Q-learning's foundations enable DQN and its extensions to leverage these benefits, promoting effective and consistent learning in complex environments.

Q-learning's ability to find the optimal policy relies on its exploration strategy and the learning rate. The agent balances between exploring new actions and exploiting the current knowledge to maximize its cumulative reward. Adjusting the learning rate and exploration strategy is crucial for successful Q-learning in different environments.

### 3.3.2 Function Approximation in Reinforcement Learning

In many practical applications of RL, the state or action spaces can be very large or even continuous. This makes tabular representations of value functions, as in Q-learning, computationally infeasible or even impossible. Such challenges arise in real-world tasks like robot control, where the state might consist of a continuous set of joint angles, or in games where the state can be an image input representing the current frame. As the state or action space grows, so does the computational demand for storing and updating each unique state-action pair in a table. This scalability issue motivates the need for function approximation.

With function approximation, the idea is to use a parameterized functional form to represent the value function or the Q-function, rather than maintaining a table. The parameters of this function are then adjusted to best fit the observed data. The goal is to generalize from the experienced states and actions to unvisited ones, allowing the agent to make reasonable decisions in unfamiliar scenarios.

While there are multiple possible choices for function approximators, deep neural networks have emerged as a dominant choice in recent years. They offer a flexible functional form capable of representing complex functions, making them particularly suited to tackle the intricacies of many RL tasks.

The transition to function approximators also brings challenges. While tabular methods converge to an optimal policy under broad conditions, function approximation methods often don't have such strong guarantees. There can be stability issues, especially when combined with off-policy methods like Q-learning. This has led to a variety of enhancements and adjustments to make DRL more stable and effective, which we'll explore in subsequent sections.

#### Deep Q Networks (DQN)

DQN represented a significant breakthrough in RL by integrating the approximation capabilities of deep neural networks with the Q-learning algorithm. It could effectively handle high-dimensional input spaces, such as raw pixel data from video game screens, allowing for generalization across diverse environments.

The key concept of DQN is the use of a neural network to approximate the Q-values. For a given state $s_t$ and action $a$, the neural network approximates the Q-value as follows:

$$Q(s, a; \theta) \approx Q^*(s, a), \tag{3.12}$$

where $\theta$ represents the neural network weights.

However, applying neural networks to Q-learning posed challenges due to the correlation between consecutive samples and the non-stationary nature of the data. Two main solutions were proposed:

1. **Experience Replay**: To reduce correlation between sequential experiences and stabilize training, experiences (comprising state, action, reward, and next state) were stored in a replay buffer (Lin, 1992). Training involved sampling random mini-batches from this replay memory, thereby decorrelating the experiences.

2. **Target Network**: To address the moving target problem in Q-learning, DQN used a separate network (the target network) with weights $\theta^-$ for computing target Q-values. These weights are updated less frequently than the primary network, enhancing training stability.

The loss function for training the DQN network, parameterized by $\theta$, is:

$$L(\theta) = \mathbb{E}\left[\left(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta)\right)^2\right], \tag{3.13}$$

where $\mathbb{E}$ denotes the expectation over all sampled trajectories from the experience replay buffer $\mathcal{D}$. The DQN training algorithm is outlined in Algorithm 1.

**Algorithm 1** Deep Q-Network (DQN) Training Procedure

Initialize update frequency $C$
Initialize replay memory $\mathcal{D}$ with capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $Q$ with weights $\theta^- = \theta$
**for** episode $= 1, M$ **do**
  Initialize state $s_t$
  **for** $t$ in {1, ..., T} **do**
    With probability $\epsilon$ select a random action $a_t$
    Otherwise select $a_t = \arg\max_{a \in \mathcal{A}} Q(s_t, a; \theta)$
    Execute action $a_t$ in the environment
    Observe reward $r_t$ and new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{D}$
    Sample random mini-batch of transitions $(s_j, a_j, r_j, s_{j+1})$from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a \in \mathcal{A}} Q(s_{j+1}, a; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
    Update $\theta^- \leftarrow \theta$ every $C$ steps
  **end for**
**end for**

### Rainbow

The Rainbow algorithm brought together various advancements in DQN to enhance performance across the board (Hessel et al., 2018).

1. **Double DQN**: Traditional DQNs often overestimate action values due to maximization bias, where the highest Q-values can be inaccurately high. Double DQN (Hasselt, Guez, and Silver, 2016) addresses this by decoupling action selection

from its evaluation, using the equation:

$$Q(s, a; \theta) = r + \gamma Q(s_{t+1}, \arg\max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta); \theta^{-}), \tag{3.14}$$

where the action is selected using the online network ($\theta$) but its Q-value is estimated using the target network ($\theta^{-}$), thus mitigating overestimations.

2. **Prioritized Experience Replay**: This method assigns priorities to experiences based on the TD error, allowing the agent to focus on more informative experiences (Schaul et al., 2015). The priority of the $i$-th experience is given by:

$$p_i = |\delta_i| + \epsilon, \tag{3.15}$$

where $p_i$ is the priority, $\delta_i$ is the TD error, and $\epsilon$ is a small constant ensuring every experience has a chance of being chosen.

3. **Dueling Networks**: This architecture splits the Q-value into two streams: state value $V(s)$ and the advantage of actions $A(s, a)$ (Wang et al., 2016). The Q-value is then calculated as:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left( A(s, a; \theta, \alpha) - \frac{1}{|A|} \sum_{a_{t+1} \in \mathcal{A}} A(s, a_{t+1}; \theta, \alpha) \right), \tag{3.16}$$

where $\alpha$ and $\beta$ are parameters for the advantage and value streams, respectively. This separation allows for a more nuanced understanding of the value of states and the advantages of actions. Figure 3.2 provides a visual representation of the network architectures deployed for each tested environment.

4. **Noisy Nets**: Traditional DQN exploration typically uses the $\epsilon$-greedy approach. Noisy Nets (Fortunato et al., 2017), however, introduce stochasticity directly into the network weights for more consistent exploration, as shown in the equation:

$$w = \mu + \sigma \odot \varepsilon, \tag{3.17}$$

where $w$ is the weight, $\mu$ and $\sigma$ are parameters, and $\varepsilon$ is sampled noise. This method diversifies the network's decisions to encourage exploration. Since this component directly aims at improving exploration, it will be further discussed in the following section.

5. **Distributional RL**: Traditional methods estimate the mean expected reward, but distributional RL (Marc G Bellemare, Dabney, and Rémi Munos, 2017) focuses on learning the distribution of potential returns, using Bellman's equation, as follows:

$$Z(s, a; \theta) = \sum_{j} p_j \cdot z_j, \tag{3.18}$$

where $Z$ represents the return distribution, with $p_j$ as the probability and $z_j$ as the actual return. This approach provides insight into the range of possible outcomes.

6. **N-step Learning**: Rather than relying solely on immediate rewards, N-step learning (Sutton, 1988) updates Q-values based on rewards accumulated over $n$ steps,

leading to faster reward propagation:

$$G_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n Q(s_{t+n}, \arg\max_{a \in \mathcal{A}} Q(s_{t+n}, a; \theta); \theta^-). \tag{3.19}$$

This approach accelerates the learning process by considering a longer trajectory of rewards and outcomes.

By combining these enhancements, Rainbow manages to achieve superior performance over many tasks, showcasing the benefits of integrating multiple DQN improvements.

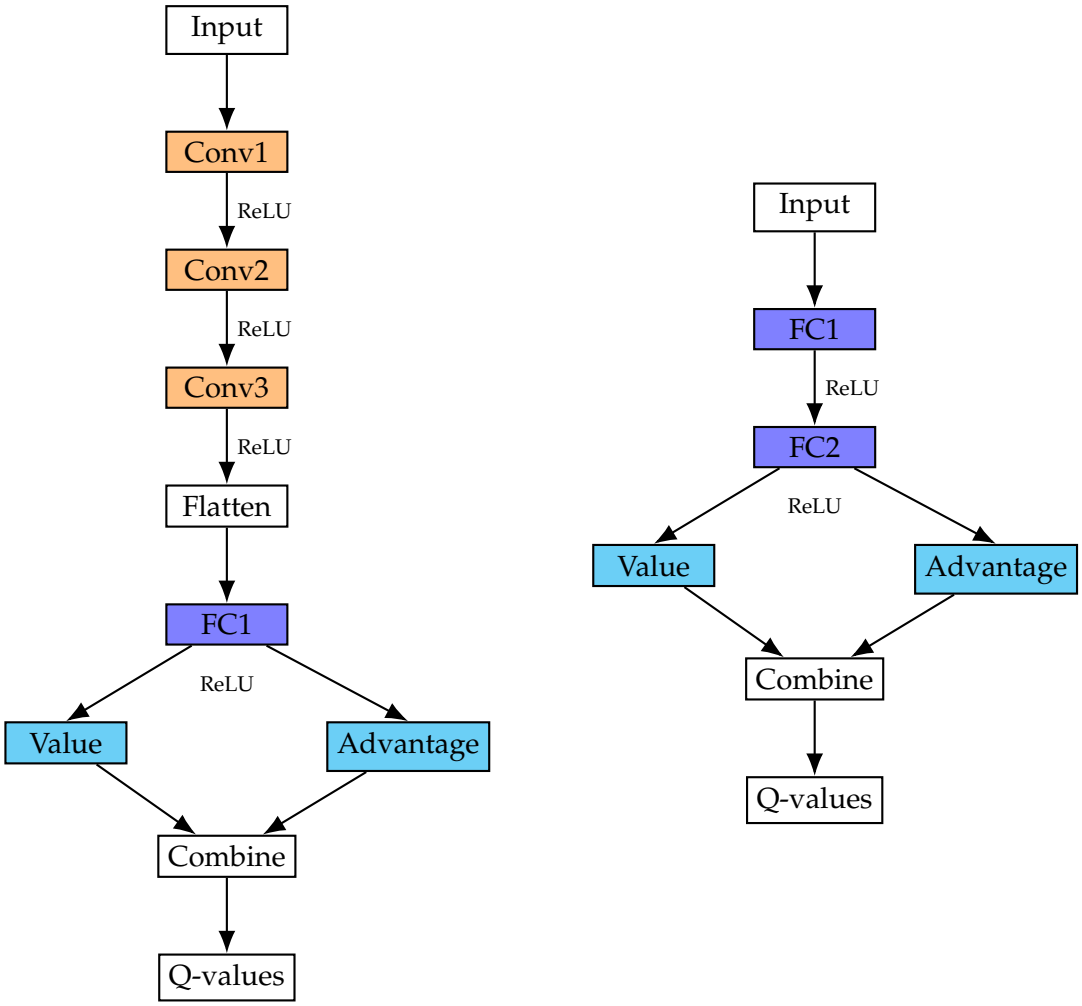


FIGURE 3.2: Neural network architectures for experimental setups: Left depicts the Atari games network with a series of convolutional layers followed by a single fully connected layer, branching into dueling streams and culminating in Q-value outputs. Right shows the classic control network, comprising two fully connected layers, also diverging into dueling streams and concluding with Q-value outputs.

## 3.4 Exploration Strategies

In RL, exploration refers to the act of an agent trying new actions to discover their effects. This is counterbalanced with exploitation, where the agent uses its current knowledge to maximize rewards. Striking the right balance between exploration and exploitation is pivotal to the agent's overall long term performance.

### 3.4.1 Blind Switching

A straightforward approach to exploration is blind switching, where the agent decides to explore without any reliance on its internal state or knowledge.

This method primarily revolves around:

1. Predetermined or stochastic periods during which the agent engages in exploration without any specific reasoning behind the decision.
2. The transition between exploration and exploitation might be based on external factors like time steps, episodes, or simply at random intervals.

While blind switching can be effective in certain scenarios, especially in the early stages of learning when little is known about the environment, it can lack the adaptability of informed switching mechanisms. It offers a more naive approach, which doesn't always guarantee optimal exploration, especially in complex environments.

However one of the main advantages of such methods is that they are stationary, i.e. they do not depend on the learning progress, which can aid in stability. This can avoid the possible circular scenario of bad exploration leading to a bad policy, thus leading to even more bad exploration.

#### Epsilon-Greedy Exploration

The $\epsilon$-greedy method is simple yet effective, especially in problems where the action space is discrete and not too large. At each time step, the agent with probability $\epsilon$ chooses an action uniformly at random (exploration), and with probability $1 - \epsilon$, it chooses the action that it believes to have the highest expected reward (exploitation) (see Algorithm 2).

**Algorithm 2** $\epsilon$-greedy Exploration

**Require:** Q-values $Q(s, a)$, Exploration rate $\epsilon$
1: Uniformly generate a random number $p$ between 0 and 1
2: **if** $p < \epsilon$ **then**
3: Choose a random action $a$
4: **else**
5: Choose $a = \arg\max_a Q(s, a)$
6: **end if**
7: **return** action $a$

Over time, the value of $\epsilon$ can be decayed to reduce exploration as the agent becomes more knowledgeable about the environment.

### Boltzmann Exploration

Boltzmann exploration, also known as softmax exploration, offers a more refined approach to action selection. Instead of the binary decision-making in $\epsilon$-greedy, the Boltzmann method selects actions probabilistically based on their estimated Q-values. Actions with higher Q-values are more likely to be chosen, but no action has a zero probability of being selected. The probability of selecting action $a$ in state $s$ according to our policy $\pi$ is given by:

$$\pi(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in \mathcal{A}} e^{Q(s,a')/\tau}}, \tag{3.20}$$

where $\tau$ is the temperature parameter. As $\tau$ approaches zero, the policy becomes more deterministic, and as $\tau$ increases, the action selection becomes more uniform.

The pseudocode for the Boltzmann exploration is detailed in Algorithm 3.

**Algorithm 3** Boltzmann Exploration

**Require:** Q-values $Q(s, a)$, Temperature $\tau$
1: **for** each action $a$ **do**
2: Compute $\pi(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in \mathcal{A}} e^{Q(s,a')/\tau}}$
3: **end for**
4: Sample action $a \sim \pi(a|s)$
5: **return** action $a$

The temperature $\tau$ controls the trade-off between exploration and exploitation. A high $\tau$ promotes exploration, while a lower $\tau$ leans towards exploitation. Like $\epsilon$ in the $\epsilon$-greedy method, $\tau$ can also be decayed over time to adjust the exploration-exploitation balance.

### Noisy Nets for Exploration

While traditional methods like $\epsilon$-greedy exploration alter the action space by taking random actions, Noisy Nets introduce stochasticity directly into the parameters of the neural network. The core idea behind Noisy Nets is that, by adding noise to the network's weights, different actions can be sampled even if the current state remains unchanged, leading to a more organic, state-dependent exploration strategy.

The noisy linear layer can be described as:

$$y = (w + \sigma_w \odot \epsilon_w) \cdot x + (b + \sigma_b \odot \epsilon_b), \tag{3.21}$$

where:

- $w$ and $b$ are the weights and biases of a standard linear layer.
- $\sigma_w$ and $\sigma_b$ represent the noise scales, learned alongside the main parameters.
- $\epsilon_w$ and $\epsilon_b$ are noise samples drawn from a predefined distribution (usually a zero mean, unit variance distribution like the normal or factorized normal distribution).

The noise variables $\epsilon$ ensure that, even in a deterministic environment, the policy exhibits exploration. This parameter space noise contrasts with traditional action space

noise, allowing for coherent exploration strategies that are sensitive to the agent's current state. By learning the noise scales $\sigma$, the network can adaptively determine the degree of exploration required at different parts of the state space. In areas where the Q-values are uncertain or change rapidly, a higher degree of noise encourages exploration, whereas in well-understood regions, the network can reduce noise to exploit known good policies.

Empirically, Noisy Nets have demonstrated improved exploration capabilities, especially in challenging environments where traditional exploration strategies falter. The method eliminates the need to manually tune exploration hyperparameters, like $\epsilon$ in $\epsilon$-greedy, since the exploration is learned intrinsically by the network.

#### Temporally-Extended $\epsilon$-greedy Exploration

The $\epsilon z$-greedy (Dabney, Ostrovski, and Barreto, 2020) method extends the standard $\epsilon$-greedy exploration approach by introducing a duration element. In the traditional $\epsilon$-greedy method, the agent chooses a random action with a probability $\epsilon$ or the best-known action with a probability $1 - \epsilon$. The $\epsilon z$-greedy approach enhances this by persisting with the random action for a sampled duration. The chosen distribution for this duration is the zeta distribution $z(n) \propto n^{-\mu}$ with $\mu = 2$, which is closely related to the Riemann zeta function.

Interestingly, research suggests that the foraging behaviour of animals, termed Lévy flights, adheres to a zeta distribution with a specific value of $\mu = 2$ (Gandhimohan M Viswanathan et al., 1996; Gandimohan M Viswanathan et al., 1999). Lévy flights represent a type of random walk where, after selecting a direction randomly, the duration for which this direction is followed is determined by sampling from a heavy-tailed distribution. Figure 3.3 shows an example of what such Lévy flights can look like.

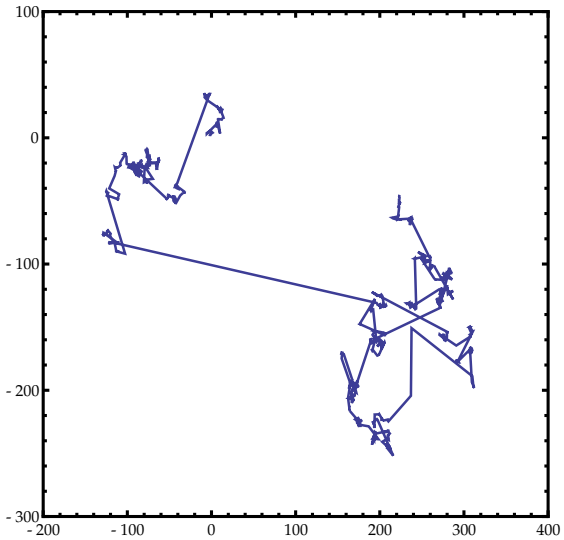


FIGURE 3.3: 1000 step Lévy flight example with origin at (0,0) in two dimensions using a Cauchy distribution. Image source: Wikipedia.[1]

Since the zeta distribution exhibits a long-tailed behaviour (Figure 3.4), this makes it particularly advantageous for our exploration context. A long-tailed distribution allows the agent to occasionally engage in extended periods of exploration, thereby probing distant regions of the state-action space, while typically committing to shorter, more

[1] https://en.wikipedia.org/wiki/Lévy_flight

localized explorations. This promotes in-depth exploration in certain episodes, enabling the agent to uncover potentially beneficial strategies, while still predominantly keeping its exploration confined to smaller steps.

This ensures that exploration is temporally extended and not limited to one-off actions, potentially enabling the agent to better gauge the outcomes of its exploratory actions. The full procedure can be found in Algorithm 4.

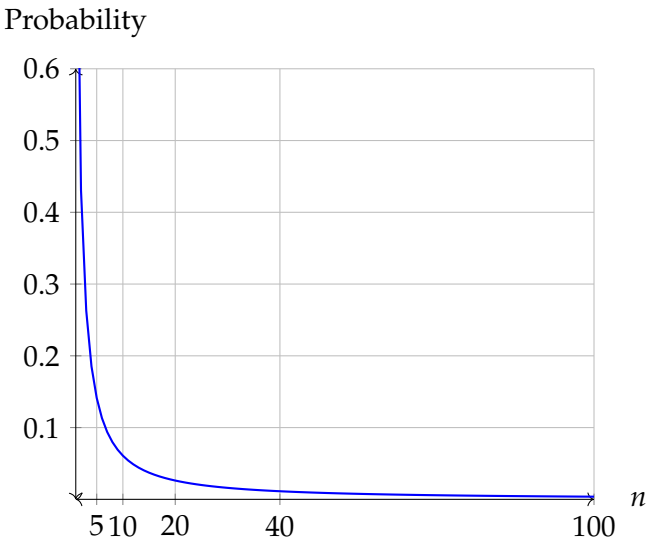


FIGURE 3.4: Zeta distribution with parameter $\mu = 2$ capped at $n = 100$ steps.

By incorporating a duration in the exploration strategy, $\epsilon z$-greedy seeks to understand the effects of an action over a period, not just at the immediate next state. This method can be particularly beneficial in environments where the consequences of actions unfold over a sequence of states rather than being immediately evident.

### 3.4.2 Informed Switching

In order to move past simple exploration mechanisms, one possible consideration is to have the decision to switch to exploration be informed by the agent's internal state.

These types of strategies can be broken down into two components:

1. Every step the agent takes, it produces a signal that's influenced by its current knowledge. This scalar trigger is used as a proxy to guide the choice of whether to explore or not.

2. Based on this signal, the agent decides whether to explore or not. This decision can be as simple as comparing the signal to a set threshold, or more complex as we will see in further sections.

We will now go over two main informed trigger types used for this study. Besides presenting the ways in which each signal is calculated we will also present a way for combining both in order to have a meaningful final trigger signal.

**Algorithm 4** $\epsilon z$-greedy Exploration

**Require:** Exploration rate $\epsilon$, Duration distribution $z$
Initialize exploration steps left $n \leftarrow 0$
Initialize last exploratory action $\omega \leftarrow -1$
**for** $t \in \{1, ..., T\}$ **do**
  Observe state $s$
  **if** $n == 0$ **then**
    Uniformly generate a random number $p$ between 0 and 1
    **if** $p < \epsilon$ **then**
      Sample duration $n \sim z$
      Choose a random action $\omega \sim U(A)$
      Assign action $a \leftarrow \omega$
    **else**
      Greedy action $a \leftarrow \arg\max_a Q(s, a)$
    **end if**
  **else**
    **return** action $a \leftarrow \omega$
  **end if**
  $n \leftarrow n - 1$
  **return** action $a$
**end for**

### Value Promise Discrepancy

The first of such informed triggers which we will explore is the Value Promise Discrepancy (VPD) (Pislar et al., 2021) which is meant to serve as a proxy an agents uncertainty (Schulz et al., 2019).

The core idea of this trigger is a an online measure comparing what the agent predicted its rewards would be $k$ steps ago to what it actually earned afterward. In simpler terms, in areas where the agent is unsure, this mismatch is likely bigger than in places where things go as expected.

Formally, the Value Promise Discrepancy over a time interval from $t - k$ to $t$ is defined as:

$$VPD(t-k, t) := V(s_{t-k}) - \sum_{i=0}^{k-1} \gamma^i r_{t-i} - \gamma^k V(s_t), \tag{3.22}$$

where $V(s)$ denotes the agent's value estimate for state $s$, $r$ is the observed reward, and $\gamma$ represents the discount factor.

While this approach can indeed be used as a proxy for the agent's uncertainty, it might not tell the full story. To combat this, we suggest incorporating an additional trigger that might help gauge other relevant aspects of the agent's current learning circumstances, such as the novelty of a state.

### SimHash and State Novelty

As a way to further enrich the decision of our agent whether to explore or not, we take inspiration from recent developments in count based exploration methods. In Tang et al., 2017, SimHash, a locality-sensitive hash is applied to input states, thus mapping

states to hash-codes, which in turn allows the counting of state occurrences with a hash table.

The SimHash function encodes a state $s \in S$ as $\phi(s) = \text{sgn}(A \cdot g(s))$, where $A$ is a $k \times D$ matrix with entries from $\mathcal{N}(0,1)$, and $g : S \to \mathbb{R}^D$ is an optional preprocessing function. The parameter $k$ (essentially the number of bits of the compressed state) influences collision likelihood and state differentiation. The hashing process can be visualized in Figure 3.5.

The counts from the hash table are used to calculate a reward bonus, which is added to the agent's final reward at each time step. The reward bonus $r^+$ is calculated according to the following formula:

$$r^+(s) = \frac{\beta}{\sqrt{n(\phi(s))}}, \tag{3.23}$$

where $\beta \in \mathbb{R}_{>0}$ and $\phi$ is the chosen appropriate hash function. The agent is then trained with the final reward $r$ returned by the environment plus the state bonus $r^+$.

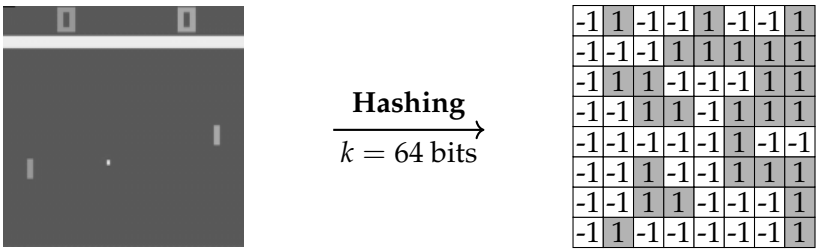


FIGURE 3.5: Visualization of a possible state hash conversion using $k = 64$ bits on an 8x8 grid. Transition from a preprocessed Pong game state (left) to the corresponding hashed state (right).

The pseudo code detailing the procedure that generates the state hashes at every timestep is detailed in Algorithm 5, while Figure 3.5 shows an example of what an hashed state conversion might look like.

**Algorithm 5** State Hashing using SimHash

1: Define state preprocessor $g : S \to \mathbb{R}^D$
2: Initialize $A \in \mathbb{R}^{k \times D}$ with entries drawn i.i.d. from $\mathcal{N}(0,1)$
3: Initialize a hash table with values $n(\cdot) \equiv 0$
4: **for** $t \in \{1, ..., T\}$ **do**
5: Store current state $s_t$
6: Compute hash code for SimHash $\phi(s_t) = \text{sgn}(A \cdot g(s_t))$
7: Update hash table $n(\phi(s_t)) \leftarrow n(\phi(s_t)) + 1$
8: **end for**

# Chapter 4

# Proposed Solution

In this chapter, we delve into our proposed solution to the challenge of effective exploration in DRL. Acknowledging the limitations of relying solely on Value Promise Discrepancy (VPD) as a single source of guidance, we introduce an approach that incorporates an additional trigger to enhance our exploration strategy. This additional trigger considers the novelty of a state to offer a more comprehensive view of the agent's learning circumstances. We present a detailed exploration strategy that combines VPD, state novelty, and homeostasis mechanisms to strike a balance between exploration and exploitation, aiming to improve the agent's learning performance in complex environments.

## 4.1 Previous Approaches

In addressing the challenge of guiding exploration effectively, we recognize that relying solely on VPD as the sole source of information may have limitations. While a high VPD can indicate uncertainty about certain environmental aspects, it may also reflect early training stages or the inherent volatility of specific state spaces. To enhance our exploration strategy, we propose integrating an additional trigger that accounts for other pertinent factors, such as state novelty.

To gauge the novelty of states, we introduce a reward bonus calculated using the hashed states, inspired by the SimHash method. This approach provides insights into the frequency of visits to specific states.

However, there are potential drawbacks to this method. Directly augmenting rewards with bonuses can distort the true environmental feedback, potentially impeding learning or leading to suboptimal policies. Another concern is the need to fine-tune the scale of the external reward using the parameter $\beta$ (refer to Equation 3.23). Moreover, there's a risk that agents constantly pursuing novelty bonuses might become trapped in "novelty loops," prioritizing novelty over task optimization.

Our proposed solution introduces the state bonus as a proxy for state novelty, distinguishing it from conventional reward shaping behavior. Higher state bonus values signify less-visited and less-known states. When combined with VPD's long-term discrepancy measure, this approach provides a more comprehensive view of the agent's uncertainty.

Unlike the conventional approach of adding a reward bonus directly to the final reward, our method is unaffected by the absolute scale of the exploration bonus. To simplify experimentation, we set the value of $\beta$ to 1 throughout all experiments.

Our goal is to have a metric that performs well in denser reward environments (VPD) while also increasing the exploration likelihood in novel states (state bonus). Our approach involves combining these signals to achieve a balanced exploration strategy that considers both long-term uncertainty and state-specific novelty.

## 4.2 Combining Triggers and Homeostasis

An issue that arises when deciding when to explore based on the values of incoming signals, there needs to be some sort of threshold, for which if it is surpassed, an exploratory action will be carried out. In practise, such signals can largely vary not only across different environments but also during training time. For instance, the overall scale of our VPD value will likely decrease as the agent learns and its accuracy improves overtime.

To surpass this obstacle, we deploy an extended version of the homeostasis mechanism (Turrigiano and Nelson, 2004), as presented by Pislar et al., 2021.

The unified homeostasis method aims to achieve adaptive exploration balance by referencing a target rate $\rho$, which denotes the desired exploration rate. Furthermore, a time scale of interest $\tau := \min(t, 5/\rho)$ is set, which is the only other aspect that can be further finetuned. This method factors in feedback from various triggers, notably VPD and the state bonus, and uses moving averages to track these triggers. At each step in the learning process, the method calculates the exploration probabilities for all triggers, averages them to determine the overall exploration probability $\overline{p}$, and then samples an exploratory decision from a Bernoulli distribution using $\overline{p}$ as the parameter. Detailed steps of this unified homeostasis mechanism are elaborated in the pseudocode provided in Algorithm 6.

**Algorithm 6** Unified Homeostasis

**Require:** target rate $\rho$, trigger types = {VPD, State Bonus}
Initialize $\overline{x} \leftarrow 0, \overline{x^2} \leftarrow 0, \overline{x^+} \leftarrow 0$ for each trigger type
**for** $t \in \{1, ..., T\}$ **do**
  Set time scale of interest $\tau \leftarrow \min(t, \frac{5}{\rho})$
  Set weight of the latest observation $\alpha \leftarrow \frac{1}{\tau}$
  **for** each trigger type $i$ **do**
    Get next signal value $x_t$
    Update mean $\overline{x} \leftarrow (1-\alpha)\overline{x} + \alpha x_t$
    Update variance $\overline{x^2} \leftarrow (1-\alpha)\overline{x^2} + \alpha(x_t - \overline{x})^2$
    Standardize $x_t \leftarrow \frac{x_t - \overline{x}}{\sqrt{\overline{x^2}}}$
    Exponentiate $x^+ \leftarrow \exp(x_t)$
    Update transformed average $\overline{x^+} \leftarrow (1-\alpha)\overline{x^+} + \alpha x^+$
    Compute exploration probability $p_i \leftarrow \min(1, \rho\frac{x^+}{\overline{x^+}})$
  **end for**
  Average probabilities $\overline{p} \leftarrow \frac{1}{n}\sum_{i=1}^{n} p_i$
  Sample $y_t \sim$ Bernoulli$(\overline{p})$
**end for**

To gain a clearer understanding of the raw value to probability transformation process and its objectives, let's examine an illustrative example, as visualized in Figure 4.1. The left plot presents the variation of trigger values over a certain time frame. In contrast, the right plot depicts the exploration probability output for each signal individually and their combined probability at each timestep. Notably, although the signal values on the left exhibit different initial absolute scales, the probabilities on the right align closely. This alignment results from the independent tracking of our homeostasis instances, which focus on generating exploration signals to meet a defined target rate, specific to each signal.

Furthermore, as the state bonus values decrease continuously on the left, we observe a corresponding stabilization and even a slight increase in the exploration probability on the right. This trend illustrates our system's underlying goal: to maintain a consistent trigger rate within a predetermined time frame. In this example, the time frame is set short enough to observe this self-regulating behaviour.

These self-regulating mechanisms enable the system to adapt its exploration strategies effectively: it facilitates continued exploration when both signals are high, leans towards more greedy actions when both are low, and strikes a balanced approach when only one of the signals is elevated. This adaptability is crucial for achieving effective decision-making in various states and scenarios.

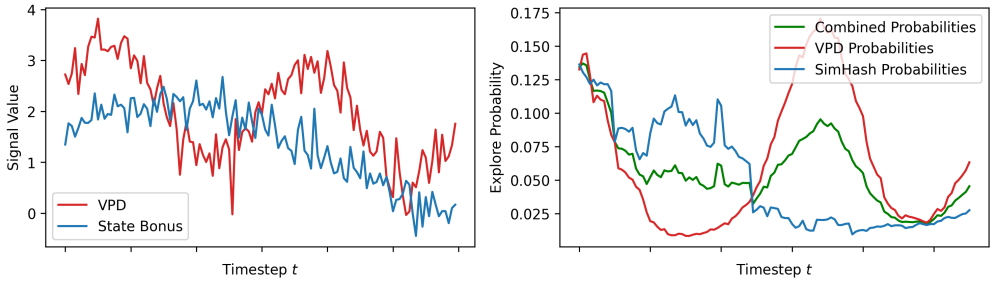


FIGURE 4.1: Sample of what the transformation of raw VPD and state bonus values (left) into exploration probabilities (right) looks like using the homeostasis mechanism.

## 4.3 Proposed Exploration Strategy

The proposed exploration strategy integrates several key methodologies to enhance the decision-making process of DRL agents. The first component is the calculation of Value Promise Discrepancy (VPD) values, which serve as a proxy for uncertainty in an agent's predictions. The VPD evaluates the divergence between the agent's expected rewards and the actual outcomes, providing a measure of how well the agent's expectations align with reality.

The second element involves computing the state bonus, a mechanism designed to incentivize the exploration of less visited, and thereby more novel, states. This approach is based on the premise that exploring unfamiliar states can unveil new learning opportunities and potentially rewarding experiences, crucial for the agent's development and adaptability.

Finally, the strategy incorporates a procedure that processes these raw signal values - VPD and state bonus - and translates them into a concrete exploration decision through the use of individual homeostasis instances. This decision-making process is structured to balance the need for exploration with the efficiency of exploitation, ensuring that the agent's actions are both exploratory and informed. The pseudocode detailing this comprehensive exploration strategy is outlined in Algorithm 7.

By combining these components, our strategy aims to address the inherent exploration-exploitation dilemma in DRL more effectively. It leverages the nuanced understanding of uncertainty and state novelty, guiding the agent towards a more balanced and informed exploration path. This approach is expected to lead to improved learning performance, especially in complex environments where traditional exploration methods may fall short. The subsequent chapters will delve into the empirical setup and evaluation of this strategy, showcasing its impact and efficacy in tested environments.

**Algorithm 7** VPD / SimHash Exploration

**Require:** VPD history length $k$, SimHash bits $\kappa$

Define state preprocessor $g : \mathcal{S} \to \mathbb{R}^D$
Initialize matrix $A \in \mathbb{R}^{\kappa \times D}$ with entries drawn i.i.d from $\mathcal{N}(0, 1)$
Initialize hash table with values $n(\cdot) \equiv 0$
Initialize unified homeostasis instance $\mathcal{H}$
**for** timestep $t \in \{1, ..., T\}$ **do**
    Get current observed state $s_t$
    Calculate state hash $\phi(s_t) = \text{sgn}(A \cdot g(s_t))$
    Calculate state bonus $(s_t) = \frac{1}{\sqrt{n(\phi(s_t))}}$
    Update counts $n(\phi(s_t)) \leftarrow n(\phi(s_t)) + 1$
    Calculate $VPD(t-k, t) = V(s_{t-k}) - \sum_{i=0}^{k-1} \gamma^i R_{t-i} - \gamma^k V(s_t)$
    Store current $V(s)$ and $R_t$
    Update $\mathcal{H}$ with $VPD$ and state bonus
    Act according to exploration decision from $\mathcal{H}$
**end for**

# Chapter 5

# Experimental Setup

In this chapter, we will discuss our experimental approach and give an in-depth look at the test environments used. This sets the stage for the results and discussions in the upcoming section. The Rainbow agent plays a key role in our tests, so we will dive into its settings, including the hyperparameters and neural network architectures. We will also detail the settings used for the baseline exploration techniques, the new methods we have proposed, and highlight any notable implementation specifics.

The experiments were specifically designed to investigate and answer the previously stated research questions:

1. How does the proposed exploration strategy compare to traditional methods in terms of enhancing the performance of DRL agents?
2. Is the learning performance of DRL agents improved by combining both exploration signals, as opposed to using each signal independently?
3. What impact does the introduction of extended exploration periods have on the learning efficiency and effectiveness of DRL agents?

To answer these questions, we test our agent across two types of RL benchmark environments: classic control and Atari games. The former presents a simpler challenge where the state space is usually relatively low and the agents can usually master the task with little to no planning, and rewards tend to be dense. The latter, presents a more challenging environment as the agent can only access the state through a raw pixel reading, and the tasks at hand usually require a certain degree of planing to master.

## 5.1 Environments

To answer the proposed research questions, we utilized a mix of classic control environments from OpenAI's Gym (Brockman et al., 2016) as well as games from the Atari Learning Environment (ALE) (M. G. Bellemare et al., 2013). Each of these environments offers unique challenges that test different facets of an agent's learning ability and adaptability.

### 5.1.1 Classic Control

The classic control tasks from OpenAI's Gym offer fundamental benchmarks in the RL domain. These environments model basic physics-driven systems where agents learn to control and manipulate given parameters to achieve specific objectives. Figure 5.1 provides sample screenshots from the used environments.

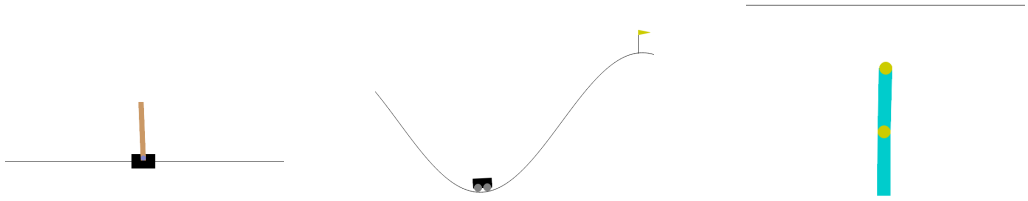

FIGURE 5.1: Sample starting states in the used classic control environments from left to right: CartPole, MountainCar and Acrobot.

### CartPole

CartPole simulates the task of balancing a pole on a moving cart. The state space includes the position ($x$), velocity ($v$), pole angle ($\theta$), and pole velocity at the tip ($\omega$). With two discrete actions, either pushing the cart left or right, the agent aims to keep the pole upright. A reward of +1 is granted for each time step the pole remains vertical, and the episode concludes if the pole deviates more than 12 degrees from the vertical or the cart moves more than 2.4 units from the center. All initial state space variables are assigned a uniformly random value in the $[-0.05, 0.05]$ range. The maximum episode length is set to 200, which also equates to the maximum possible end reward the agent can obtain in the used v0 version of the environment.

### CartPole Sparse

CartPole Sparse introduces a challenging aspect to the original task with a sparse reward structure. The particular version we used conforms to the implementation in Tassa et al., 2018, in which the agent only receives a non zero reward when $|x| < 0.25$ and $\cos\theta > 0.995$. The state space and action space remain consistent with the described CartPole environment. This adjustment in rewards adds complexity, requiring the agent to navigate the environment with less frequent feedback.

### MountainCar

MountainCar presents the agent with the task of moving a car uphill to reach a flag. The state space involves the position ($x$) and velocity ($v$) of the car. The action space offers three discrete choices—push the car left, no push, or push the car right. The standard reward is -1 at each time step until the car reaches the flag. This design encourages the agent to find an optimal strategy to reach the goal as quickly as possible. The position of the car is assigned a uniform random value $x \in [-0.6, -0.4]$ and the starting velocity of the car is always assigned to $v = 0$. Episodes terminate either when position of the car $x \geq 0.5$ (the goal position on top of the right hill is reached) or the maximum episode length of 200 is reached.

### Acrobot

In Acrobot, the agent faces the challenge of controlling a system with angular positions, velocities of the two joints, and the velocity of the center of mass. The action space provides three discrete choices — torque toward or away from the direction of the swing, or no torque. The reward system involves a penalty of -1 at each time step, motivating the agent to swing up as fast as possible. All initial state space variables are assigned a

uniformly random value in the $[-0.1, 0.1]$ range, resulting in a starting position were both links connected to the joints are facing a downwards direction. The maximum episode length is set to 200 in the used v0 version of the environment, however the reward threshold is set to -100.

### 5.1.2 Atari

The Atari Learning Environment (ALE) offers a suite of video games from the iconic Atari 2600 console. Beyond their nostalgic value, these games are used as benchmarks in the RL domain, offering a diverse set of challenges. One significant aspect of using Atari games is the visual input: agents perceive the game state from raw pixel values. To ensure efficient and effective learning, these pixel values are often subjected to preprocessing steps.

Before we dive into the specifics of each game, it's worth noting the general state preprocessing applied across all Atari games:

- **Grayscale Conversion:** The raw RGB frames are converted to grayscale to reduce the dimensionality, making it computationally cheaper without sacrificing important state information.
- **Downsampling:** The resolution of the original frames is reduced, again to decrease the computational load.
- **Frame Stacking:** Several consecutive frames are stacked together to give the agent a sense of motion and velocity which are crucial for decision-making in dynamic games.

The final state observed by the agent is thus converted from an initial sequence of $210 \times 160$ RGB images, to a stack of four 84 x 84 gray scaled images. Figure 5.2 provides sample screenshots of the tested Atari games.

As shown in Machado et al., 2018, different ALE parameter choices can heavily influence the research conclusions, thus a more standard methodology is proposed for evaluation. We follow these recommendations by enabling sticky actions across all tested Atari games. Contrary to the original versions in which the transactions are deterministic, sticky actions make it so that the agent executes its last action, as opposed to the one the agent just selected, with a certain probability $p$. This change aims to reduce the ability of the agent to simply memorize a sequence of actions that can result in a high score, by introducing a form of stochasticity in the transition process. The probability $p$ is set to 0.25, as suggested by Machado et al., 2018. Table 5.1 summarizes all the preprocessing hyperparameters used across all the Atari game experiments.

#### Pong

One of the earliest arcade video games, Pong, simulates a table tennis match. The agent controls a paddle and aims to bounce the ball past the opponent's paddle.

Pong is characterized by a simple visual state space, consisting of the ball, the agent's paddle, and the opponent's paddle. The motion and position of these elements define the state. The action space is straightforward, allowing the agent to move the paddle up, down, or keep it stationary. The rewards are defined by scoring, with the agent receiving a reward of +1 for scoring a point and -1 if the opponent scores. Pong is considered a relatively simple exploration game, requiring basic paddle movements and serving as an accessible introduction to Atari game environments.

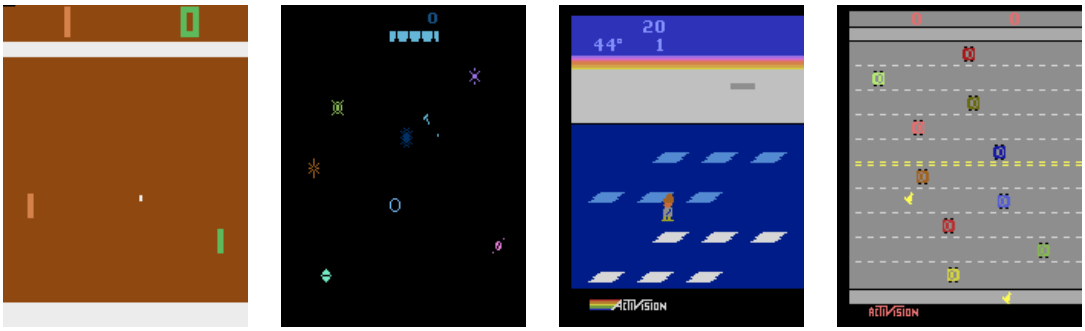

FIGURE 5.2: Atari game environments from left to right: Pong, Gravitar, Frostbite and Freeway.

| Hyper-parameter | Value |
|---|---|
| Grey-scaling | True |
| Observation down-sampling | (84, 84) |
| Frames stacked | 4 |
| Action repetitions | 4 |
| Reward clipping | [-1, 1] |
| Terminal on loss of life | False |
| Sticky actions probability | 0.25 |
| Max agent steps per episode | 27K |

TABLE 5.1: Preprocessing hyper-parameter values for the Atari experimental setup.

### Gravitar

In Gravitar, the agent controls a spaceship with the goal to destroy enemy bases while navigating planetary terrains and avoiding collisions. The game introduces elements of gravity, inertia, and limited fuel, making it a complex control task.

Gravitar presents a more complex visual state space, including the position of the spaceship, the player controls , the position of enemy bases, and terrain layout. The action space is rich, allowing the agent to shoot or move the spaceship in any of 8 possible directions. The rewards are earned by destroying enemies, but penalties are incurred upon collision with terrains or enemies. Gravitar is considered a hard exploration game, demanding precise control in a gravitational environment, and is known for its challenging dynamics and strategic gameplay.

### Frostbite

In Frostbite, the agent's goal is to help the main character build an igloo by jumping on floating ice floes, while avoiding obstacles and enemies.

Frostbite presents a game screen with moving ice floes, hazards like birds and fish, and the partially built igloo. The action space is defined by the agent's ability to move in four directions: up, down, left, and right. Points are gained by jumping on ice floes and completing the igloo, but penalties are given if the character falls into the water or collides with enemies. Frostbite is also considered a hard exploration game given the need for platforming skills and strategic planning to navigate the hazards and build the igloo successfully.

#### Freeway

Freeway is an Atari game that simulates the challenge of crossing busy highways as a chicken. The objective is for the agent to navigate from one side of the highway to the other, dodging fast-moving cars in the process.

The game environment for Freeway features a multi-lane road filled with various vehicles traveling at different speeds. The visual state space includes the position of the chicken, the lanes and the moving cars. The action space is limited to moving the chicken up or down, reflecting the simplicity of the game's controls. The reward structure is binary, with the agent receiving a point for successfully crossing to the other side and no reward—or potential penalty—for being hit by a car or failing to advance. Despite its seemingly straightforward premise, Freeway offers a complex exploration challenge due to the dynamic and unpredictable patterns of traffic, requiring the agent to develop timing and pattern recognition skills to succeed. This game is particularly notable for its representation of real-time decision-making under rapidly changing conditions.

## 5.2 Rainbow Agent

In this section, we detail the hyper-parameter settings and neural network configurations employed by the Rainbow agent in the classic control and Atari environments.

### 5.2.1 Hyper-parameters

Table 5.2 presents the key hyper-parameters governing the agent's behaviour, including discount factor, replay memory parameters, learning rate, and training details. Note that parameters such as target update frequency are stated in agent steps and not in in game frames, meaning that a single agent step is equivalent to four in game frames, due to the preprocessing and frame skipping described earlier to Atari games.

All the hyper-parameter values used for the Atari experiments follow Hessel et al., 2018, while the ones used on the classic control environments are taken from Castro et al., 2018.

### 5.2.2 Networks

The network used for the Atari games is composed of 3 initial convolutional layers, followed by a single hidden layer with 512 units, followed by the value and advantage streams from the dueling architecture which both contain a hidden layer with 512 units. As for the output layer, it contains the same number of units as there are actions, which varies across the tested games.

For the classic control environment, which is inherently simpler, we employ a less complex network consisting of two hidden layers, each with 512 units. This network also incorporates the dueling architecture used for the Atari games, ensuring that the number of output units corresponds to the agent's available actions.

The specific hyperparameters for each network, including the number of channels, filter sizes, and hidden layer units, are detailed in Table 5.3.

[1] In Acrobot and MountainCar, the distributional min/max values are instead set to $[-100, 100]$.

| Hyper-parameter | Classic Control | Atari |
|---|---|---|
| Discount $\gamma$ | 0.99 | 0.99 |
| Update horizon | 3 | 3 |
| Min replay history | 500 | 20k |
| Update period | 4 | 4 |
| Target update frequency | 100 | 8k |
| Optimizer | Adam | Adam |
| Minibatch size | 32 | 32 |
| Distributional number of atoms | 51 | 51 |
| Distributional min/max | $[-10, 10]$[1] | $[-10, 10]$ |
| Learning rate | 0.001 | 0.0000625 |
| Optimizer $\epsilon$ | 0.0003125 | 0.00015 |
| Number of iterations | 30 | 100 |
| Training steps per iteration | 1000 | 250k |
| Max steps per episode | 200 | 27k |
| Replay scheme | Prioritized | Prioritized |
| Replay capacity | 50k | 1M |
| Replay buffer batch size | 128 | 32 |

TABLE 5.2: Hyper-parameter settings for Classic Control and Atari environments using the Rainbow agent.

| Network parameter | Classic Control | Atari |
|---|---|---|
| Channels | - | 32, 64, 64 |
| Filter size | - | 8x8, 4x4, 3x3 |
| Stride | - | 4, 2, 1 |
| Hidden layers | 2 | 1 |
| Hidden units | 512 | 512 |
| Output units | Number of actions | Number of actions |

TABLE 5.3: Network hyper-parameter settings for classic control and Atari environments using the Rainbow agent.

### 5.2.3 Summarizing Learning Performance

All of the evaluation is done in regards to the number of frames which the agent experiences, and not total number of frames, meaning that due to the frame stacking and frame skipping parameters, one agent step is equivalent to 4 in game frames.

Training is done across multiple iterations, each iteration being bounded by a set number of maximum frames. In order to prevent episodes from ending abruptly, this set number of maximum frames is rounded up to allow for full episode termination.

The main metric which we track across our experiments is the total score obtained by the agent during training. Specifically, we only measure the average score during training, without interpolating in evaluation only runs, as suggested in Machado et al., 2018.

All the the experiments were carried out across multiple independent game runs, each with a distinct random seed. For the classic control environments, the runs were averaged across 50 distinct runs, as for the tested Atari games 3 random seeds were used, due to the longer training times and computational resources available.

## 5.3 Exploration Strategies

In this section we detail the specific parameters used for each exploration strategy. To allow for a more focused research on the question of *when* to explore, we aim to keep exploration rates between all the tested exploration strategies relatively similar.

Its worth noting that the parameters chosen for the various exploration strategies were not optimized for maximal task performance but were selected to ensure a reasonable level of performance. This decision was made to facilitate an initial study of the behaviour of these strategies across different scenarios, leaving room for further experimentation.

### 5.3.1 Blind Switching

The methods used as the baselines require a small amount of hyper-parameter tuning, making them straight forward to use across a wide array of environments and obtain good performance. Table 5.4 provides a quick overview of the exploration parameters used for the baseline methods.

| Exploration parameter | Classic Control | Atari |
|---|---|---|
| Initial $\epsilon/\tau$/target rate | 0.01 | 1 |
| Final $\epsilon/\tau$/target rate | 0.01 | 0.01 |
| Decay period in agent steps | - | 250k |
| VPD value of $k$ | 3 | 5 |
| SimHash number of bits | 16 | 256 |

TABLE 5.4: Exploration parameters used across all exploration strategies in classic control and Atari environments.

#### Epsilon-Greedy Exploration

In the case of $\epsilon$-greedy, the parameter epsilon $\epsilon$ dictates the exploration ratio, which can be decreased as the agent learns more about its environment, directly implying the need to decrease exploration as we learn about the dynamics of our environment. In cases where such linearly decaying exploration rate is used, this decaying rate is kept the same across all exploration strategies, to allow for fairer comparison.

#### Boltzmann Exploration

In Boltzmann exploration, the parameter tau $\tau$ plays a crucial role in shaping the exploration strategy. Unlike epsilon in $\epsilon$-greedy, which directly represents the probability of selecting a random action, tau in Boltzmann exploration influences the stochasticity of action selection through a temperature parameter. A higher tau leads to a more randomized selection of actions, akin to increased exploration, while a lower tau leans towards deterministic choices based on the action values. Essentially, tau serves as a knob to control the level of exploration-exploitation trade-off in Boltzmann exploration, offering a continuous spectrum of exploration intensities.

In contrast, epsilon in $\epsilon$-greedy serves as a fixed probability threshold, determining the likelihood of choosing exploration over exploitation in a discrete manner.

While both parameters aim to balance exploration and exploitation, they do so in different ways. For simplicity's sake, while these have different meanings, we use the

same value for both across all experiments, not only for the initial and final value, but also in case of linear decay during training.

#### Temporally-Extended $\epsilon$-greedy Exploration

The extended exploration strategy ($\epsilon z$-greedy) functions in the same way as $\epsilon$-greedy, sharing the epsilon parameter, the only difference being that this strategy repeats its exploratory actions according to a duration sampled from a certain distribution. As described in the earlier section, the used distribution is the *zeta* distribution with a parameter $\mu = 2$. The maximum number of continuous exploration steps is capped at 100, as it is done in Dabney, Ostrovski, and Barreto, 2020 when using the Rainbow agent.

#### Noisy Nets for Exploration

When using Noisy Nets, other exploration mechanisms such as $\epsilon$-greedy are not used (value of epsilon is set to 0), thus relying only on the noise added to the network for a source of actions selection. The main hyper-parameter $\sigma_w$ and $\sigma_b$, which refer to the noise scales for the weights and biases (as seen in Equation 3.21) are both set to a value of 0.5, as in Hessel et al., 2018.

As the explicit exploration rate is not set, no further parameter tuning is required. Furthermore, as the network learns to ignore the noise at different rates depending on its current state, this works as a form of self-annealing property, as defined explicitly in other exploration methods.

### 5.3.2 Informed Switching

One of the main goals with the newly proposed exploration strategy was to keep the need for hyper-parameters to a minimum, however there are a few that are needed which can have an impact on learning outcome.

#### Value Promise Discrepancy

When looking at the way we calculate the VPD (as defined in Equation 3.22), a relevant time interval has to be given in the form of $k$. Choosing an appropriate value of $k$ involves a trade-off between capturing long-term patterns and maintaining responsiveness to recent changes in the agent's behaviour. The optimal value of $k$ may be dependent on the characteristics of the environment. For simpler environments a smaller time frame might be best since rewards are often denser, which allows the agent to be more responsive in its error estimation. On the other hand, environments that are more complex and require more planning might benefit from a larger time frame, making the error calculation less prone to smaller deviations.

With these considerations in mind, we used a value of $k = 3$ on the simpler classic control environments, and a value of $k = 5$ for all the tested Atari games.

#### State Novelty

The state novelty signal produced by the agent does not require any parameter tuning as the value of $\beta$ in Equation 3.23, which influences the scale of the reward bonus in the original application, is no longer relevant, as this value is simply used as a signal where the only important factor is its relative change across time. For simplicity's sake, the value of $\beta$ is set to 1 for all experiments.

**Combined Homeostasis**

An important component of this proposed exploration strategy is the transformation of the raw incoming trigger signals into a exploration probability which is then later combined. This process is achieved through the use of a separate homeostasis instance for each of the trigger types.

The main parameter which needs to be set is relevant time scale of interest tau which is depended on the value of $n$, where $\tau := \min(t, \frac{n}{\text{target rate}})$, in which $t$ is the agent step counter and we have a pre-defined target rate, which corresponds to our desired exploration rate.

The value of tau plays a role in the calculation of the running average, specifically in determining how much weight to give to the latest observation in relation to the previous observations. It essentially defines a *time constant* that determines the window of influence for our exponential moving averages.

As seen in Algorithm 6, we use the parameter tau to derive the value of alpha, which represents the weight of the latest observation when updating the running averages. Exponential moving averages place more weight on recent observations and the alpha parameter controls how much weight is given to the latest observation compared to the accumulated average of past observed values.

In the early stages of training when the counter $t$ value is small, the value of alpha will be close to 1, meaning that the moving average will be adapting quickly to the new incoming values. As training progresses, alpha will decrease, thus making the moving average less sensitive to new values. The cap of the parameter tau to $\frac{n}{\text{target rate}}$, ensures that even when the agent is further into training, the incoming signal values still have an influence on the ruining average while keeping the stability given by the history of previous values. The value of $n$ was set to 5 across all experiments.

The last component of the newly proposed exploration strategy, is the combination of the individual trigger probabilities coming out of each homeostasis instance. As it was detailed in the previous chapter, we opted for simply taking the average of the probabilities.

# Chapter 6

# Results & Discussion

In this chapter, we explore the outcomes of the described experiments and discuss the insights gained. Our proposed exploration strategy is assessed and its performance is compared against traditional DRL methods as well as the isolated triggers.

Besides the analysis of quantitative performance metrics such as average reward across training, we also take a look at other less performance oriented metrics to aid us in drawing meaningful conclusions and identify trends.

## 6.1 Classic Control

Figure 6.1 presents the aggregated results for the four classic control games tested. The graphed lines represent the average episodic returns across 50 trials, while the shaded regions denote the 95% confidence intervals.

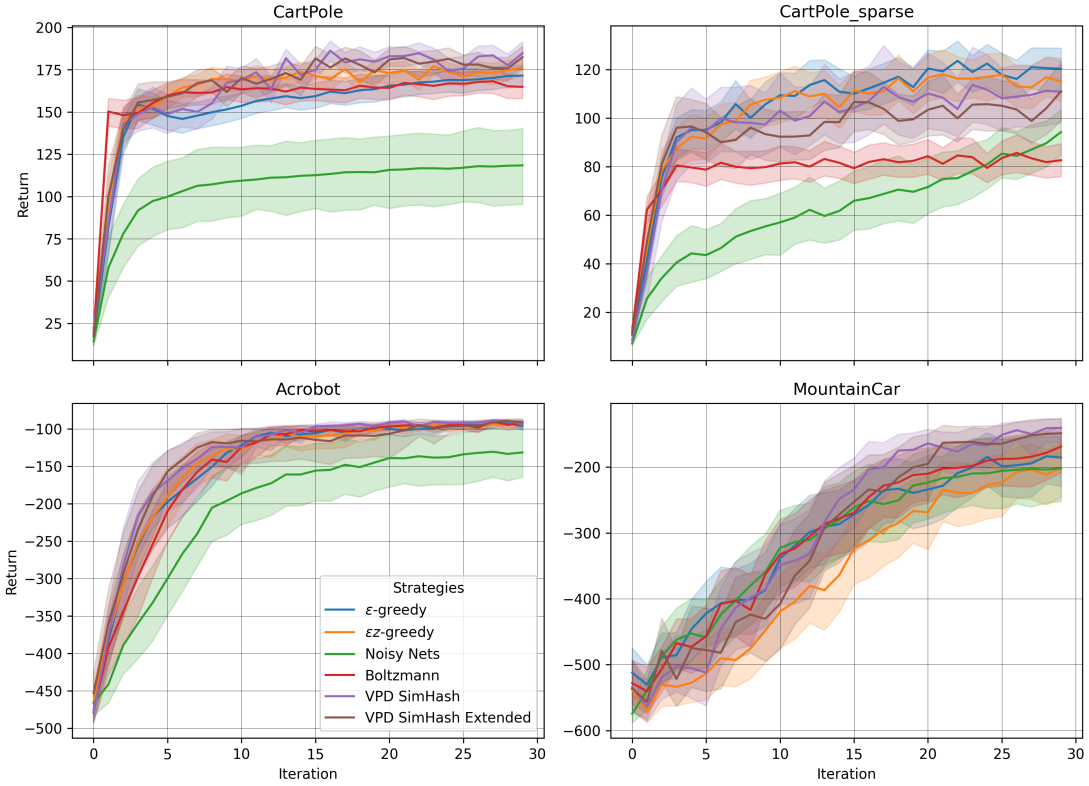


FIGURE 6.1: Comparative performance of different exploration strategies in classic control tasks.

### 6.1.1 Performance Analysis

We will now discuss the performance of our proposed exploration strategy relative to the baseline methods for each classic control game.

**CartPole:** All strategies tend to converge towards a comparable level of performance, while the proposed strategies exhibit a marginally higher mean return, hinting at a slight edge in performance. The Noisy Nets approach, on the other hand, lags behind, potentially due to excessive stochasticity in the network, which may not be suitable for this relatively straightforward task.

**CartPole sparse:** In the sparse reward variant of CartPole, the simpler $\epsilon$-greedy and $\epsilon z$-greedy methods surpass the VPD SimHash strategies, as evidenced by the higher average returns throughout the training process. Noisy Nets and Boltzmann exploration both fall short in this environment, where rewards are less frequent, thus emphasizing the challenge of such sparse reward environments.

**Acrobot:** The differences between strategies are minimal in the Acrobot game, but the VPD SimHash approaches demonstrate a slightly more pronounced learning curve, possibly indicating a more efficient learning process. Noisy Nets appears to underperform again when compared with the array of strategies tested.

**MountainCar:** In this task, the VPD SimHash strategies shows superior performance, particularly as training progresses, although the advantage is slight. This environment seems to be less sensitive to the nuances of exploration timing, as suggested by the broadly similar results across all strategies.

### 6.1.2 Exploration Behaviour Analysis

A deeper examination of exploration behaviour in specific tasks, such as the Acrobot scenario, offers valuable insights. Figure 6.2 illustrates the exploration trajectories for various strategies, including $\epsilon$-greedy, VPD/SimHash, and VPD/SimHash-Extended. This visualization helps us understand how exploration timings differ among these strategies. In the $\epsilon$-greedy approach, exploration appears quite random, as indicated by the scattered magenta bars. In contrast, the VPD/SimHash methods tend to concentrate exploration towards the latter half of an episode. This pattern suggests that higher VPD or state bonus values are being registered as the episode progresses, possibly due to increased uncertainty or encountering more novel states.

Observing the VPD and state bonus values in Figure 6.3, we note that VPD values increase as the agent nears its goal. This escalation triggers more frequent exploration in that time frame. Additionally, the exploration periods' length varies; while $\epsilon$-greedy and regular VPD/SimHash methods explore step-by-step, the extended version shows variable exploration periods, influenced by the sampled extended exploration length.

As the agent progressively learns the dynamics of these simpler classic control environments, its state value predictions become more precise, leading to a gradual decrease in VPD values across episodes. Concurrently, as exploration advances and the agent becomes more familiar with its environment, the hash table tracking state visit counts becomes increasingly populated. This results in a general decrease in the state bonus, as most states have been visited multiple times.

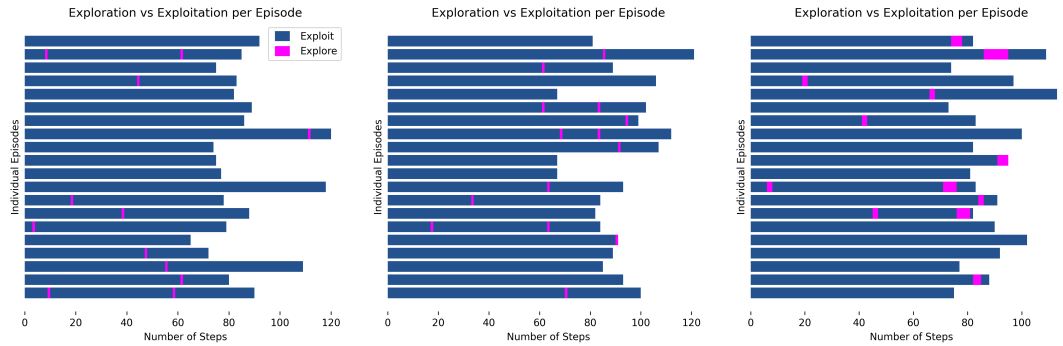


FIGURE 6.2: Exploration trajectories in Acrobot for different strategies. Left to right: $\epsilon$-greedy, VPD/SimHash, VPD/SimHash-Extended.

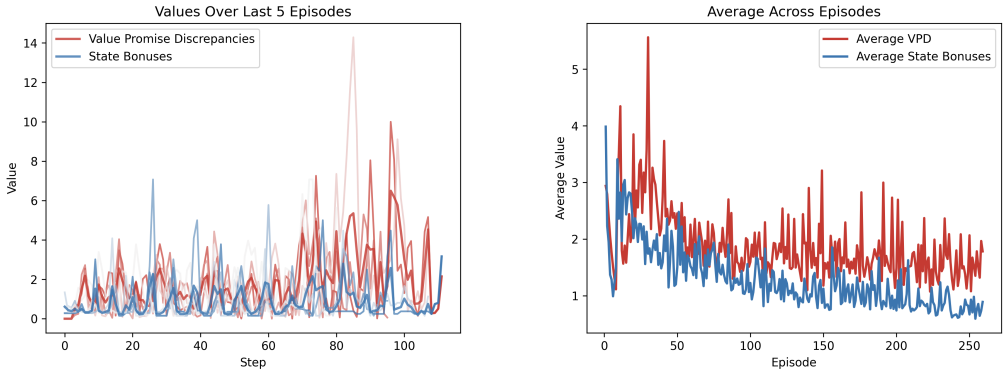


FIGURE 6.3: VPD and state bonus over the last 5 episodes (left) and averaged across all episodes (right) in Acrobot.

### 6.1.3 Results Overview

Overall, our proposed exploration strategy demonstrates subtle performance improvements in three out of the four classic control tasks, showing a slight dip in performance in the sparser reward task. Notably, extending the exploration period beyond the standard one-step approach did not yield significant performance gains.

This analysis underscores that while our exploration strategy is roughly as effective as traditional methods across a range of simpler tasks, it goes to show the diminished importance regarding the timing of exploration, in simpler and denser reward environments.

## 6.2 Atari

Transitioning to the more complex Atari domain, we evaluate the impact of our exploration strategy on more challenging exploration games.

### 6.2.1 Performance vs. Baselines

Figure 6.4 juxtaposes the performance of our exploration strategy against established baselines over the course of 3 independent trials.

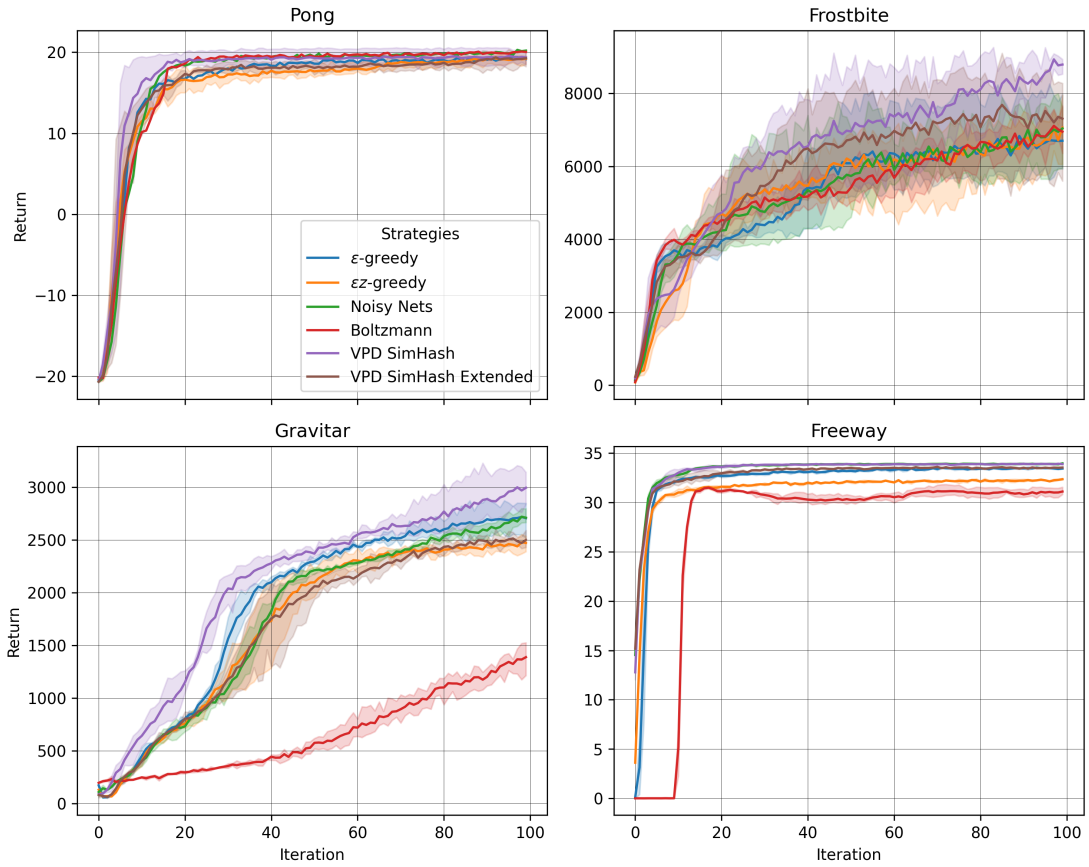


FIGURE 6.4: Comparative analysis of exploration strategies in Atari games. Results are averaged over 3 independent runs, with shaded areas depicting 95% confidence intervals.

**Pong:** Although all strategies quickly reached a similar level of performance, the VPD SimHash strategy initially achieved slightly higher returns. This early advantage can be attributed to its refined exploration timing, which, despite Pong's relative simplicity as an exploration task, might have provided an edge. Pong does not heavily penalize random exploration, which explains the eventual convergence of all strategies. The nuanced exploration timing of VPD SimHash offers a marginal benefit, but the game's straightforward nature may make the extended exploration strategy unnecessary.

**Frostbite:** Frostbite presents a more complex exploration challenge, requiring agents to balance short-term rewards with the discovery of long-term strategies for higher scores. The success of the VPD SimHash strategies in this game can be attributed to their effective management of exploration timing, which is critical in environments where rewards are distributed unevenly. The single-step exploration variant, in particular, accelerates learning by exploiting the immediate feedback from the environment effectively, without the need for extended exploration that might delay the exploitation of known high-reward states. Still, the extended exploration version of the proposed approach outperforms all other baselines, which seem to perform about equally in this game.

**Gravitar:** Gravitar is a hard exploration game that demands strategic discovery and exploitation of sparse rewards, making it an ideal candidate for showcasing the strengths of sophisticated exploration strategies. The superior performance of the standard VPD SimHash strategy throughout the learning process emphasizes the importance of well-timed exploration decisions. On the other hand, the Boltzmann method, which relies on a more static exploration approach, struggles in such a complex environment, further highlighting the benefits of a responsive exploration strategy. The extended exploration version seems to underperform throughout training, even when compared to some other blind strategies.

**Freeway:** In the Freeway game, the VPD SimHash strategy (non-extended) and Noisy Nets method emerged as the top performers. The VPD SimHash strategy excelled, likely due to its capability to fine-tune exploration timing in response to the unpredictable movement of vehicles, a crucial factor in the game's complex environment. Noisy Nets also performed well, suggesting that the stochasticity introduced by noise in the network parameters was beneficial in this context, perhaps by encouraging diverse exploratory actions that could lead to successful road-crossings. Conversely, Boltzmann exploration lagged behind, potentially due to its tendency to overemphasize exploitation based on current value estimates, which may not adapt quickly enough to the fast-paced changes in traffic. This pattern reaffirms the observation that the extended exploration methods do not always confer an advantage; in fact, in Freeway, the added complexity of extended exploration did not translate to improved outcomes, highlighting the efficacy of more straightforward, single-step exploration approaches in certain Atari game environments.

### 6.2.2 Performance vs. Individual Triggers

A more granular examination of performance is depicted in Figure 6.5, which reveals the differential impact of exploration strategies on individual games.

**Pong:** The game's simplicity allows for relatively uniform high performance among all strategy variations. Similar to the baseline results, no strategy showcases a clear performance gain, likely due to the simplicity of the game, and the lesser need to optimize the timing of exploration.

**Frostbite:** In the intricate exploration environment of Frostbite, the combination of both signals outperforms the individual strategies across all stages of learning. The isolated VPD signal closely rivals the combination, falling slightly behind only in the latter stages of learning.

**Gravitar:** Consistently with other findings, the combined approach in Gravitar excels, underscoring its superiority in facilitating timely exploration decisions, thereby accelerating the learning trajectory compared to singular signal strategies.

**Freeway:** In the last tested game of Freeway, all variations perform equally well, supporting the idea that each individual signal provides valuable information about the agent's internal state.

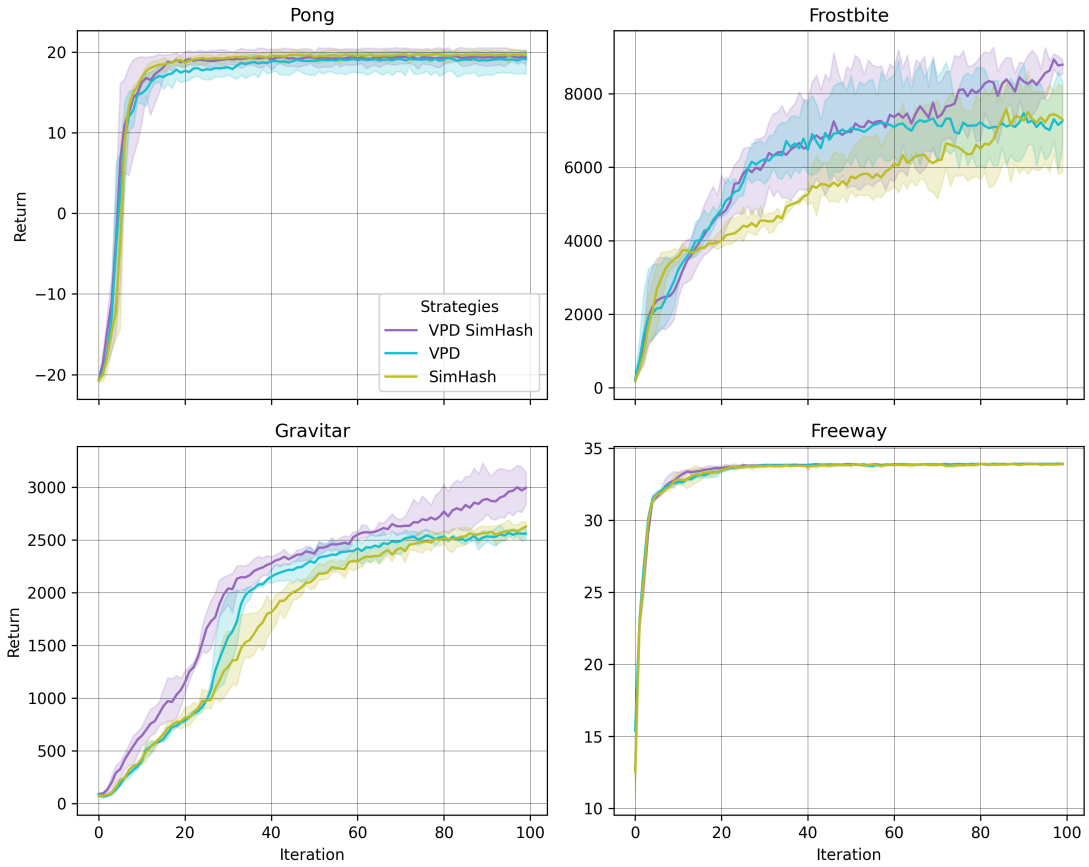


FIGURE 6.5: Performance breakdown by individual game in the Atari suite, demonstrating the effectiveness of the combined exploration strategy. Results are averaged over 3 independent runs, with 95% confidence intervals represented by the shaded areas.

### 6.2.3 Results Overview

The results obtained in the Atari domain reinforce the hypothesis that the strategic timing of exploration is pivotal in enhancing learning efficiency.

Primarily, the combination of the VPD and state bonus signals contributes to significant performance improvements, particularly in exploration-intensive games. This outcome affirms the robustness and versatility of our proposed exploration strategy in complex environments. We have also seen that the more frequent exploration provided by methods such as the Boltzmann strategy, seems to yield poorer results in more complex Atari games, suggesting that the optimal amount of exploration done has been crossed, resulting in poorer performing learned policies.

Furthermore, the ablation study contrasting the combined method against its individual components illustrates a distinct performance superiority. Empirically, this suggests that the synergy between these components results in an agent that adopts a more sophisticated and effective approach to exploring its environment. The nuanced interplay of predictive value discrepancies and state novelty, as captured by the combined strategy, informs a more dynamic and contextually responsive exploration behaviour.

Lastly, extended exploration steps showed not to have a positive effect on the learning efficiency of our tested agent. One of the main causes for this behaviour might be related to the usage of sticky actions on the Atari game environments. Sticky actions

add a level of stochasticity to the environment by ignoring the agent's current action (with probability 0.25) and repeating the previous action. This offers a similar but not identical type of behaviour provided by the extended exploration strategy.

The two main differences being that by using sticky actions the agent does not observe the action that was actually executed but only the action it intended to execute, and also the fact that a non-Markovian element is introduced, given that the effect of an action can depend on the previous action, creating a dependence on the action history. While it has been shown that in some environments sticky actions can degrade performance of agents, it is done so in order to improve their robustness and prevent memorization type of behaviour. Furthermore, sticky actions are a modification applied to the environment and not the agent itself, thus making it possible for both to coexist. In Dabney, Ostrovski, and Barreto, 2020, results across the Atari game suit suggested that when sticky actions are enabled, most of the performance benefit of the $\epsilon z$-greedy algorithm is lost, only making a significant difference in a set of harder exploration games.

Given the wide adoption for sticky actions to simulate/add stochasticity in to a particular environment, we argue for a need to further investigate the addition of extended exploration mechanisms, such that would work well in these given scenarios.

# Chapter 7

# Conclusion

In this thesis, we investigated the dynamics regarding the timing of exploration in DRL agents. We introduced a novel exploration strategy that combines the agent's value state prediction errors with counts of the current hashed state, thus balancing long-term uncertainty and state-specific novelty to guide the timing of exploration. The conducted experiments across different environments highlighted not only the significant role of exploration timing in agent performance, but also a significant performance gain over existing traditional exploration methods.

## 7.1 Answers to the Research Questions

1. In complex environments, the proposed strategy consistently outperformed traditional methods, reinforcing the importance of strategic exploration timing. However, results from simpler environments indicated a diminished relevance of exploration timing.

2. The integration of VPD and state bonus signals emerged as more effective than their independent use, particularly enhancing learning in challenging exploration scenarios.

3. Extended exploration periods did not enhance learning efficiency, highlighting the necessity for a context-specific approach to exploration duration, specifically when sticky actions are present.

## 7.2 Contributions

Our key contribution is the development of a novel exploration strategy that leverages two distinct distinct signals and a framework for combining them to produce relevant exploration triggers. This approach has demonstrated its potential in complex environments, outperforming traditional methods, particularly where a more refined exploration strategy is required.

## 7.3 Limitations and Future Directions

This study focuses primarily on value-based methods, and its applicability to game environments such as Atari.

Extending this exploration strategy to policy gradient methods offers a promising research direction. In policy gradients, exploration is typically governed by stochastic policy updates. By integrating VPD and state bonus values into the policy update mechanism, agents can be incentivized to explore more when these values are high,

indicating uncertainty or novelty in the environment. This could be achieved by modifying the policy's entropy. An increase in the entropy of the policy when uncertainty is elevated (VPD) or novel states are encountered (high state bonus values), should encourage the policy to explore more diverse actions. Such modifications would allow the policy to dynamically balance exploration and exploitation based on the uncertainty and novelty of the state space, potentially leading to more robust and efficient learning.

Another exciting avenue for future research would be the application of the proposed exploration strategy to risk-aware environments, such as robotics or finance, presents an exciting avenue for future research. In such domains, minimizing risk while exploring is crucial. Our proposed method, which typically promotes exploration under high uncertainty, could be adapted to prioritize safety in risk-averse scenarios. Additionally, state novelty information could serve as an indicator of potential high-risk situations, warranting more cautious exploration.

Lastly, and perhaps the most obvious extension to this work would be to experiment with different signals that can serve as relevant triggers. In addition, the way in which the signals are combined is also an open question, with some immediate candidates being taking the maximum probability value instead of the average, or perhaps introducing some sort of linear combination which is optimized by an external meta-controller.

In conclusion, this thesis not only advances our understanding of exploration dynamics in DRL but also lays the groundwork for future innovations in the field. It opens up avenues for more adaptive, efficient, and safe exploration strategies, suitable for a wide array of applications, from complex game environments to real-world scenarios demanding nuanced decision-making under uncertainty.

# Bibliography


Ali Taïga, Sid Ahmed et al. (2020). "Benchmarking Bonus-Based Exploration Methods on the Arcade Learning Environment". In: *arXiv preprint arXiv:2008.02330*.

Andrychowicz, Marcin et al. (2020). "Learning dexterous in-hand manipulation". In: vol. 39. 1. SAGE Publications Sage UK: London, England, pp. 3–20.

Auer, Peter (2002). "Using confidence bounds for exploitation-exploration trade-offs". In: *Journal of Machine Learning Research*, pp. 397–422.

Azar, Mohammad Gheshlaghi, Ian Osband, and Remi Munos (2017). "Minimax Regret Bounds for Reinforcement Learning". In: *arXiv preprint arXiv:1703.05449*.

Badia, Adrià Puigdomènech et al. (2020). "Never Give Up: Learning Directed Exploration Strategies". In: *arXiv preprint arXiv:2002.06038*.

Bellemare, M. G. et al. (2013). "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* 47, pp. 253–279.

Bellemare, Marc et al. (2016). "Unifying count-based exploration and intrinsic motivation". In: *Advances in neural information processing systems* 29.

Bellemare, Marc G, Will Dabney, and Rémi Munos (2017). "A distributional perspective on reinforcement learning". In: *International conference on machine learning*. PMLR, pp. 449–458.

Bellemare, Marc G, Will Dabney, and Remi Munos (2017). "A Distributional Perspective on Reinforcement Learning". In: *arXiv preprint arXiv:1707.06887*.

Bellemare, Marc G, Yavar Naddaf, et al. (2013). "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* 47, pp. 253–279.

Berner, Christopher et al. (2019). "Dota 2 with Large Scale Deep Reinforcement Learning". In: *arXiv preprint arXiv:1912.06680*.

Brockman, Greg et al. (2016). *OpenAI Gym*.

Burda, Yuri, Harri Edwards, et al. (2018). "Large-Scale Study of Curiosity-Driven Learning". In: *International Conference on Learning Representations*.

Burda, Yuri, Harrison Edwards, et al. (2018). "Large-Scale Study of Curiosity-Driven Learning". In: *Proceedings of the International Conference on Learning Representations*.

Castro, Pablo Samuel et al. (2018). "Dopamine: A research framework for deep reinforcement learning". In: *International Conference on Learning Representations*.

Cathomas, Flurin et al. (2021). "The translational study of apathy—an ecological approach". In: *Current Opinion in Behavioral Sciences* 38, pp. 39–45.

Charikar, Moses S (2002). "Similarity estimation techniques from rounding algorithms". In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 380–388.

Chen, Xi and Li Lihong (2020). "A Unified View on Tabular Reinforcement Learning: Bridging Model-Free and Model-Based Methods". In.

Cohen, Jonathan D, Samuel M McClure, and Angela J Yu (2007). "Should I stay or should I go? How the human brain manages the trade-off between exploitation and

exploration". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 362.1481, pp. 933–942.

Conti, E et al. (2017). "Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents". In: *Advances in Neural Information Processing Systems*, pp. 5027–5038.

Costa, Vincent D et al. (2019). "Dopamine modulates novelty seeking behavior during decision making". In: *Behavioral Neuroscience* 133.3, pp. 309–321.

Dabney, Will, Georg Ostrovski, and André Barreto (2020). "Temporally-extended $\epsilon$-greedy exploration". In: *arXiv preprint arXiv:2006.01782*.

Dann, Christoph, Gerhard Neumann, and Jan Peters (2017). "Policy Evaluation with Temporal Differences: A Survey and Comparison". In: *Journal of Machine Learning Research*. Vol. 15. 1, pp. 809–883.

Ecoffet, Adrien et al. (2019). "Go-Explore: a New Approach for Hard-Exploration Problems". In: *arXiv preprint arXiv:1901.10995*.

Fortunato, Meire et al. (2017). "Noisy Networks for Exploration". In: *arXiv preprint arXiv:1706.10295*.

Gershman, Samuel J (2018a). "Deconstructing the human algorithms for exploration". In: *Cognition* 173, pp. 34–42.

— (2018b). "The Computational Nature of Human Learning and Exploration". In: *Current Directions in Psychological Science* 27.5, pp. 375–380.

Hasselt, Hado van (2010). "Double Q-learning". In: *Advances in Neural Information Processing Systems*, pp. 2613–2621.

Hasselt, Hado van, Arthur Guez, and David Silver (2016). "Deep Reinforcement Learning with Double Q-learning". In: *AAAI*, pp. 2094–2100.

Hessel, Matteo et al. (2018). "Rainbow: Combining improvements in deep reinforcement learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1.

Huber, Anton (1991). "Central and peripheral effects of spiperone, raclopride, SCH 23390 and clozapine on acetylcholine and dopamine release in the rat brain: an in vivo microdialysis study". In: *Journal of Neural Transmission: Parkinson's Disease and Dementia Section* 3.1, pp. 59–69.

Janner, Michael, Justin Fu, and Marvin Zhang (2019). "Model-based Value Expansion for Efficient Model-Free Reinforcement Learning". In: *arXiv preprint arXiv:1903.00374*.

Janz, Dominik, Harm van Seijen, and Richard S Sutton (2019). "Are Option-Return Functions Needed?" In: *arXiv preprint arXiv:1901.01301*.

Kolter, J Zico and Andrew Y Ng (2009). "Near-Bayesian Exploration in Polynomial Time". In: *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 513–520.

Lin, Long-Ji (1992). *Reinforcement learning for robots using neural networks*. Carnegie Mellon University.

Machado, Marlos C et al. (2018). "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents". In: *Journal of Artificial Intelligence Research* 61, pp. 523–562.

Mnih, Volodymyr, Adria Puigdomenech Badia, et al. (2016). "Asynchronous Methods for Deep Reinforcement Learning". In: *arXiv preprint arXiv:1602.01783*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, et al. (2013). "Playing Atari with Deep Reinforcement Learning". In: *arXiv preprint arXiv:1312.5602*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, et al. (2015a). "Human-level control through deep reinforcement learning". In: *Nature*. Vol. 518. 7540. Nature Publishing Group, pp. 529–533.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, et al. (2015b). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, pp. 529–533.

O'Donoghue, Brendan et al. (2018). "The Uncertainty Bellman Equation and Exploration". In: *arXiv preprint arXiv:1709.05380*.

Osband, Ian and John Aslanides (2018). "Randomized Prior Functions for Deep Reinforcement Learning". In: *Advances in Neural Information Processing Systems*, pp. 8617–8629.

Osband, Ian, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy (2016). "Deep Exploration via Bootstrapped DQN". In: *Advances in Neural Information Processing Systems*.

Osband, Ian, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy (2016). "Deep Exploration via Bootstrapped DQN". In: *Advances in Neural Information Processing Systems*, pp. 4026–4034.

Osband, Ian, Tom Moerland, and Benjamin Van Roy (2014). "Deep Exploration via Randomized Value Functions". In.

Osband, Ian, Daniel Russo, and Benjamin Van Roy (2017). "Deep Exploration via Randomized Value Functions". In: *arXiv preprint arXiv:1703.07608*.

Osband, Ian and Benjamin Van Roy (2013). "On Lower Bounds for Regret in Reinforcement Learning". In: *Proceedings of the 30th International Conference on Machine Learning*, pp. 1390–1398.

Ostrovski, Georg et al. (2017). "Count-Based Exploration with Neural Density Models". In: *arXiv preprint arXiv:1703.01310*.

Parisotto, Emilio and Ruslan Salakhutdinov (2017). "Model-Based Reinforcement Learning for Atari". In: *arXiv preprint arXiv:1707.06887*.

Pathak, Deepak, Pulkit Agrawal, Alexei A Efros, et al. (2017a). "Curiosity-driven Exploration by Self-supervised Prediction". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17.

— (2017b). "Curiosity-driven Exploration by Self-supervised Prediction". In: *International Conference on Machine Learning*.

Pislar, Miruna et al. (2021). "When should agents explore?" In: *arXiv preprint arXiv:2108.11811*.

Puterman, Martin L (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*.

Sabatelli, Matthia et al. (2018). "Deep quality-value (DQV) learning". In: *arXiv preprint arXiv:1810.00368*.

Schaul, Tom et al. (2015). "Prioritized Experience Replay". In: *arXiv preprint arXiv:1511.05952*.

Schulz, Eric et al. (2019). "Structured, uncertainty-driven exploration in real-world consumer choice". In: *Proceedings of the National Academy of Sciences* 116.28, pp. 13903–13908.

Silver, David et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587, pp. 484–489.

Singh, Satinder et al. (2000). "Convergence results for single-step on-policy reinforcement-learning algorithms". In: *Machine learning* 38, pp. 287–308.

Strehl, Alexander L. and Michael L. Littman (2008). "An Analysis of Model-Based Interval Estimation for Markov Decision Processes". In: *Journal of Computer and System Sciences* 74.8, pp. 1309–1331.

Strens, Malcolm (2000). "A Bayesian Framework for Reinforcement Learning". In: pp. 943–950.

Sutton, Richard S (1988). "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1, pp. 9–44.

Sutton, Richard S and Andrew G Barto (1998). *Reinforcement learning: An introduction*. MIT press.

— (2018). *Reinforcement learning: An introduction*. MIT press.

Tang, Haoran et al. (2017). "# exploration: A study of count-based exploration for deep reinforcement learning". In: *Advances in neural information processing systems* 30.

Tassa, Yuval et al. (2018). "Deepmind control suite". In: *arXiv preprint arXiv:1801.00690*.

Thompson, William R (1933). "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples". In: *Biometrika* 25.3-4, pp. 285–294.

Tokić, Michel (2010). "Adaptive $\epsilon$-greedy exploration in reinforcement learning based on value differences". In: *Annual Conference on Artificial Intelligence*, pp. 203–210.

Turrigiano, Gina G and Sacha B Nelson (2004). "Homeostatic plasticity in the developing nervous system". In: *Nature reviews neuroscience* 5.2, pp. 97–107.

Viswanathan, Gandhimohan M et al. (1996). "Lévy flight search patterns of wandering albatrosses". In: *Nature* 381.6581, pp. 413–415.

Viswanathan, Gandimohan M et al. (1999). "Optimizing the success of random searches". In: *nature* 401.6756, pp. 911–914.

Waltz, James A et al. (2020). "Altered reinforcement learning in schizophrenia: neural and behavioral evidence". In: *Biological Psychiatry* 78.6, e17–e26.

Wang, Ziyu et al. (2016). "Dueling Network Architectures for Deep Reinforcement Learning". In: *arXiv preprint arXiv:1511.06581*.