Deep Exploration by Planning With Uncertainty in Deep Model Based Reinforcement Learning

Yaniv Oren

A thesis presented for the degree of
Master in Computer Science

Department of Computer Science
Technical University Delft
The Netherlands
July 10, 2022

Abstract

Deep, model based reinforcement learning has shown state of the art, human-exceeding performance in many challenging domains. Low sample efficiency and limited exploration remain however as leading obstacles in the field. In this work, we incorporate epistemic uncertainty into planning for better exploration. We develop a low-cost framework for estimating and computing the uncertainty as it propagates in planning with a learned model. We propose a new method, *planning for exploration*, that utilizes the propagated uncertainty for inference of the best action for exploration in real time, to achieve exploration that is informed, sequential over multiple time steps and acts with respect to uncertainty in decisions that are multiple steps into the future (deep exploration). To evaluate our method with the state of the art algorithm MuZero, we incorporate different uncertainty estimation mechanisms, modify the Monte-Carlo tree search planning used by MuZero to incorporate our developed framework, and overcome challenges associated with learning from off-policy, exploratory trajectories with an algorithm that learns from on-policy targets. *Our results demonstrate that planning for exploration is able to achieve effective deep exploration even when deployed with an algorithm that learns from on-policy targets, and using standard, scalable uncertainty estimation mechanisms.* We further provide an ablation study that illustrates that the methodology we propose for on-policy target generation from exploratory trajectories is effective at alleviating averse effects of training with trajectories that have not been sampled from an explotiatory policy. We provide full access to our implementation and our algorithmic contributions through GitHub.

Contents

Chapter 1

Introduction

In February this year (2022), one of the first successful deployments of a reinforcement learning (RL) algorithm has been achieved by MuZero [1], [2], reaching state of the art video compression on YouTube videos [3] and signifying a momentous milestone in RL. MuZero is the latest in a line of deep model-based RL (DMBRL) algorithms developed by DeepMind, utilizing a model for *planning* and accomplishing a series of breakthroughs in the field. Starting with an approach that was tailored specifically for the game of Go [4], [5] this line of work has been able to generalize and achieve superhuman performance in other games such as Chess and Atari, while retaining the same performance as the specifically-tailored algorithm in Go [6], with MuZero. These milestones illustrated RL's monumental capacity for learning effective behavior policies in complex domains, exceeding the performance of the best human experts in those fields. One of the main leaps introduced with MuZero to this line of work is that it is not only a general algorithm in the sense that it is not tailored to any specific domain, but it is also *model-learning* - rather than requiring a model of the environment (the rules of the game of Go, for example) in order to be able to conduct planning, MuZero learns an abstraction of the environment's dynamics by itself. Even with this additional challenge of learning a useful representation of the environment's dynamics, MuZero is able to achieve state of the art performance in all previously mentioned domains and many others. While able to achieve state of the art performance in extremely challenging domains, the exploration approach that is used by this family of algorithms is rudimentary, uninformed random exploration.

Effective, informed exploration is crucial in many problem settings [7]–[9]. Specifically, in any task where state-action pairs that are part of the optimal policy lie far from the agent's starting state, and are not lead to with a path of "breadcrumb" rewards, the agent cannot be expected to learn the optimal policy from the execution of random exploration, without executing a large number of interactions with the environment. The number of interactions with the environment required to randomly find a far reward can grow as fast as exponentially with the length of the path to the reward [7], which is extremely sample inefficient. The current state of the art in exploration that is informed both locally as well as over multiple time steps (so-called *deep exploration*) revolves around utilizing *epistemic uncertainty* to drive exploration into previously-unexplored areas of the environment's state-action space or the learner's parameters' space [8], [10]. Epistemic uncertainty refers to uncertainty that is sourced in lack of information [11]. It is usually considered side by side with *aleatoric uncertainty*, which refers to uncertainty sourced in stochasticity inherent to the system. A standard example is the toss of a die - if we have tossed a die a large number of times and

we observe it to be fair (rather-evenly-divided number of occurrences of each outcome), it is standard to model the die toss using probability theory as a random variable from a uniform distribution. This modelled stochasticity (or *predictive-uncertainty* about the die's outcome), inherent to the toss of a die, is referred to as aleatoric uncertainty. If we have never tossed this die before however, we do not know - is the die fair? Is it biased? This uncertainty, that can be considered as separate from the stochasticity inherent to the toss of the die, and is sourced specifically from our *lack of experiences tossing the die*, is referred to as epistemic uncertainty. MuZero possesses a wide range of function approximators in which epistemic uncertainty can be estimated. First, due to the abstracted nature of the model learned by MuZero, when planning with the model MuZero can simulate states that are not grounded in the actual reality of the environment, thus leading to unreliable estimates in the planning. These estimates can further be expected to grow increasingly unreliable with the depth of a planning trajectory as the uncertainty propagates through the planning tree. Additionally, this source of uncertainty can be expected to propagate directly into the other predictors used by MuZero, namely the reward and value functions, predicting the rewards expected from the environment in transitioning between the abstracted states and the values of those states. This rich bed of epistemic uncertainty sources in planning provides a unique opportunity for demonstrating the potential of harnessing epistemic uncertainty in planning to achieve advanced, informed exploration, as well as more informed planning in general. We explore additional possibilities for harnessing epistemic uncertainty in planning in the future work section.

In order to harness the different sources of epistemic uncertainty taking part in planning, the first contribution of this work is a framework to efficiently compute approximations of the propagated uncertainty, first forward as uncertainty in state influences uncertainty in reward, value and future state predictions, and then back as later planning predictions improve earlier planning predictions. In order to achieve that, a method for quantifying local uncertainty estimates must first be chosen. Methods for quantifying uncertainty, and specifically epistemic uncertainty, have been an active field of research in machine learning (ML) in general and RL specifically for some time [11], aiming to develop reliable, effective, theoretically supported and computationally efficient uncertainty measures [12]–[15], for the purposes of safety [16], [17], reliability [18], [19], as well as effective exploration [7], [8]. While there is yet to exist one standard framework for formulating uncertainty from different sources, the classic approach remains probabilistic modelling and Bayesian inference, and quantifying the uncertainty as either the variance or the entropy in a probability distribution. In this work we choose the classic probabilistic approach to demonstrate that even with straightforward modelling choices, that are associated with very low additional computational cost, the uncertainty can be incorporated sufficiently to achieve efficient informed exploration. In addition, the framework that we propose for incorporating the uncertainty is not limited to enabling informed exploration, and presents an opportunity for harnessing uncertainty in planning for additional purposes, such as pessimistic exploitation in offline RL [20]. We discuss additional possible uses for the incorporated uncertainty in the future work chapter.

Attempting to achieve advanced exploration with MuZero poses a set of additional challenges, due to MuZero's training with on-policy targets. On-policy learning generally prevents an algorithm from being able to learn from decisions that *have not* been taken by its exploitation policy. Following an advanced exploration policy is prone to inducing such decisions. This effect may cause instability, hinder learning and have other detrimental effects. With this motivation in mind, in this work we aim to answer the following research questions:

1. Given classic probabilistic modelling of epistemic uncertainty in an otherwise deterministic model-learning DMBRL algorithm, **how does the epistemic uncertainty propagate in**

a planning tree?

2. Using this framework, **how can the propagated uncertainty be utilized in planning to achieve effective exploration** in DMBRL?

3. As an additional challenge, **how can the proposed methodology for exploration be extended to on-policy training approaches** as well as off-policy training?

1.1 Contributions

In order to answer the research questions listed above, as well as provide algorithmic improvements resulting from the use of epistemic uncertainty over the chosen baseline of MuZero, our contributions consist of the following: First, we propose a probabilistic formulation of the otherwise deterministic planning phase. This formulation allows for quantifying of epistemic uncertainty, as well as development of update rules to approximate the propagation of the uncertainty from different sources in planning in deterministic model-learning DMBRL. We proceed to introduce modifications to the classic Monte-Carlo tree search (MCTS) approach to allow for propagation of the epistemic uncertainty in MCTS according to the developed uncertainty propagation update rules. Next, we develop a new approach for utilizing the propagated epistemic uncertainty in planning, coined *planning to explore*. Our method aims to achieve exploration that is **directed** (making informed decisions towards action that are associated with higher relevant-information gain), **consistent** (directed over multiple time steps) and **farsighted** (informed with uncertainty associated with future decisions as well as local decisions). In this work, we refer to exploration that achieves all three as *deep* exploration. Our method is tailored around our use case of MuZero, but is extendable to other planning algorithms. To enable MuZero, which learns from on-policy targets, to learn from the off-policy exploratory decisions executed by our method, we propose approaches for generating on-policy targets from off-policy exploratory trajectories. Lastly, we provide an empirical evaluation of our method. We evaluate our method against two different tasks that were chosen explicitly as hard-exploration tasks - the classic Mountain Car [21] exploration benchmark, as well as a toy environment developed specifically to evaluate the agent's capacity for farsighted directed exploration over time. In these tasks we evaluate the agent against two different reward schemes, to illustrate the challenges that can arise from learning from exploratory decisions in different settings, and with two different epistemic uncertainty estimation mechanisms, to evaluate the soundness of the method as well as its resilience to less reliable uncertainty estimates. In addition, we conduct an ablation study to evaluate the effect of the different approaches we propose for generating on-policy targets from exploratory trajectories.

The results of the evaluation demonstrate that efficient exploration that achieves all three targets can be achieved by harnessing epistemic uncertainty in planning with our method. The ablation study we conduct illustrates the beneficial effects of the different approaches developed for learning from off-policy exploratory trajectories in an algorithm that is trained with on-policy targets. An open source, documented implementation of our method - incorporated into MuZero - is provided at [22], based on the MuZero implementation by [23].

1.2 Outline

The rest of the thesis paper is organized as follows:

1. Chapter 2 provides background to RL, deep RL, model-based RL and exploration in RL (section 2.2), MCTS (section 2.3), and MuZero (section 2.4). Last, a brief survey of approaches for modelling and quantifying uncertainty in deep learning is provided, along with an overview of the specific uncertainty estimation methods used in this work (section 2.5).

2. Chapter 3 details the contributions of this work. First, a probabilistic formulation of the algorithm's deterministic planning phase is described, to enable modelling the uncertainty according to standard approaches (section 3.1). The probabilistic formulation is followed by our proposed methodology for efficiently propagating uncertainty inside planning trees (section 3.2). The propagation methodology is followed by proposed adaptations to MCTS to incorporate the propagation of the uncertainty (section 3.3). Next, we present our method for harnessing the propagated uncertainty for informed exploration, *planning for exploration* (section 3.4). Finally, we propose different approaches for generating on-policy targets that are used to train MuZero, from the off-policy exploratory trajectories executed by the *planning for exploration* method (section 3.5).

3. Chapter 4 describes the experimental setup used in the evaluation. The chapter opens with a with a description of how the uncertainty estimation mechanisms were incorporated into MuZero's infrastructure 4.1). We follow with a description of the agents used to evaluate the method, as well as the ablation study conducted to evaluate the effect of learning from different targets (section 4.2). Next, we describe the tasks used to evaluate our approach, as well as the specifics of the reward schemes chosen (section 4.3). Last, we discuss the evaluation metrics, statistical significance and hyper-parameter optimization used (sections 4.4 and 4.5).

4. Chapter 5 presents the results achieved by our method employed with MuZero, demonstrating the capacity of the agent for deep exploration (section 5.1). Next, results evaluating the agent against the same tasks with a different reward scheme are provided, illustrating the challenges of learning stably from off-policy, exploratory trajectories, as well as the agent's retained-capacity to solve even such tasks (section 5.2). Last, we present the results of the ablation study, demonstrating the gain from incorporating the modified targets proposed in our work (section 5.3).

5. Chapter 6 summarizes and compares to related work in the field of utilizing uncertainty in MBRL (section 6.1), as well as a discussion of the strengths and limitations expected from our method (section 6.2). The related work section describes other approaches for model-based reinforcement learning under uncertainty (section 6.1.1), including an approach that explicitly plans to explore (section 6.1.2). Work on planning with uncertainty in MCTS (section 6.1.3), and work utilizing epistemic uncertainty for exploration (section 6.1.4). Finally, work on on-policy training with experiences from off-policy trajectories (section 6.1.5) is summarized.

6. Chapter 7 concludes the paper, and discusses future work.

Chapter 2

Background

In this chapter we describe background that is relevant for our work in incorporating epistemic uncertainty into planning, to drive decision making that optimizes for efficient deep exploration. This chapter starts by describing relevant notation for decision making problems and planning. It follows by introducing RL, including specific concepts in RL that are relevant to our work, such as value functions, deep RL, MBRL and exploration in RL. A description of MCTS follows, focusing on the variation of MCTS used by MuZero as well as trajectory selection heuristics. Next, the MuZero algorithm is described in detail. Finally, a discussion of uncertainty definition, quantification and estimation is provided, including a description of the two uncertainty estimation approaches used in our work: ensembles and visitation counting.

2.1 Formulating decision making and planning problems

It is standard to formulate decision making and planning problems using a Markov Decision Process (MDP) [24]. An MDP M is defined as a tuple (S, A, ρ, T, R). S denotes a state space, A an action space and ρ a probability distribution over starting states. $T : S \times A \to P(S)$ denotes a possibly-stochastic transition function, where $P(S)$ denotes a probability distribution over states S. $R : S \times A \times S \to P(\mathbb{R})$ a possibly-stochastic reward function. At each (discrete) time-step t, an agent operating in M observes a state $s_t \in S$, executes an action $a_t \in A$ and receives reward $r_t \sim R(s_t, a_t, s'_t) \in \mathbb{R}$. The agent than transitions according to the transition function T to a new state $s_{t+1} \sim T(s_t, a_t)$.

In the setting of decision problems, solving an MDP, whether by planning with a model of the transition and reward functions or by learning from interactions, refers to finding a policy $\pi : S \to P(A)$ for choosing actions a_t in states s_t, that optimizes with respect to some objective $J[\pi]$. The standard objective to optimize for is the expected cumulative reward over time or *expected episodic return*, with respect to an horizon H:

$$J[\pi] = \mathbb{E}_\pi\left[\sum_{t=0}^{H} \gamma^t r_t\right] := \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r_t | r_t \sim R(s_t, a_t), a_t \sim \pi(s_t), s_t \sim T(s_{t-1}, a_{t-1}), s_0 \sim \rho\right]$$

Where we the use notation $\mathbb{E}_\pi[\cdot]$ to denote the expectation with respect to a policy and the dynamics of the environment: $r_t \sim R(s_t, a_t), s_t \sim T(s_{t-1}, a_{t-1}), a_{t-1} \sim \pi(s_{t-1}), s_0 \sim \rho$. An important

component of objective definition is the discount factor $\gamma \in (0, 1]$. When $\gamma \to 0$, the induced optimization is completely short-sighted, and only takes into account immediate rewards. When $\gamma = 1$, the optimization is completely far-sighted, which in infinite horizon problems can prevent convergence. As a result, it is standard to solve the optimization problem under choices of γ in the range: $0.95 \leq \gamma < 1$.

In some instances, it is useful to consider an extended formulation of this setting, using *partially-observable* MDPs (POMDPs) [25]. In a POMDP $M = (S, A, \rho, T, R, \Omega, O)$, the agent does not directly receive the state information $s_t \in S$ when transitioning to step t, but instead receives an observation $o_t \in \Omega$. $O : S \to P(\Omega)$ specifies the probability of receiving the observation $o_t \in \Omega$ at state s_t.

2.2 Reinforcement Learning

Reinforcement learning (RL) we consider an agent g that operates in an environment M formulated as an MDP. In RL, the agent generally aims to learn a policy π that optimizes the objective $J[\pi]$ from *interactions*. There are different RL approaches to learning such a policy. Some rely on searching the policy space directly [26], while others rely on learning *value functions* that induce an implicit optimal policy. In the following sections we will describe value functions in more detail.

2.2.1 The Value function and Q-value function

The *value function* $V^\pi : S \to \mathbb{R}$, evaluates the expected value $V^\pi(s_t)$ of being in state s_t and following policy π from time step t onward:

$$V^\pi(s_t) = \mathbb{E}_\pi\left[\sum_{k=t}^{H} \gamma^{k-t} r_k\right] = \mathbb{E}_\pi\Big[r_t + \gamma V^\pi(s_{t+1})\Big]$$

Another value function that is often used in RL is the *Q-function* $Q : S \times A \to \mathbb{R}$. The Q function describes the value of executing each possible action in each state, following the policy π from that point forward:

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi\Big[r_t + \gamma Q^\pi(s_{t+1}, a_{t+1})\Big]$$

Where s^π_{t+1} denotes the state arrived at from executing action a_t at state s_t, and a^π_{t+1} the action to execute following policy π at state s_{t+1}. The optimal Q function $Q^{\pi^*} = Q^*$ provides an implicit policy as well, and is defined as:

$$Q^*(s_t, a_t) = \max_{a_t} Q(s_t, a_t) = \mathbb{E}_\pi\Big[r_t + \gamma \max_{a_t \in A} Q^*(s_{t+1}, a_{t+1})\Big]$$

The optimal policy π^* can be immediately extracted from Q^* as follows:

$$\pi^*(s_t) = \arg\max_\pi \mathbb{E}_\pi\Big[r_t + \gamma Q^*(s_{t+1}, a_{t+1})\Big]$$

Many approaches to RL focus on learning an optimal-value function approximation, or an optimal-Q function approximation, in an attempt to directly improve the implicitly induced policy. Learning

these functions in environments with large or continuous state spaces require advanced function approximation techniques, the most standard of which has become artificial neural networks (ANNs, or NNs for short).

2.2.2 Deep learning and deep reinforcement learning

Deep learning is the term used in ML for learning with deep NNs (DNNs) [27]. In deep reinforcement learning (DRL), the agent employs DNNs as function approximators. One of the first successful incorporations of DNNs into RL was the deep Q-networks (DQN) algorithm [28], using a DNN to learn the Q values of an optimal policy. Other approaches have been proposed since, learning a range of functions, from the value and policy functions [29] to a dynamics model of the environment [30].

2.2.3 Model based reinforcement learning

Model based RL (MBRL) and deep MBRL (DMBRL) assume the agent has access to a model of the dynamics of the environment: the transition function with or without the reward function. The model is either directly provided to the agent or learned by the agent, and is not generally guaranteed to model the environment exactly. Once the agent has access to a model, it can utilize *planning*. The term planning is used to describe the process of a series of simulations of state transitions and received rewards. The planning phase is often represented as a tree, rooted at the state the agent is in s_t, where each node k represents an action a_{t+k} and the transition (or estimated transition, or distribution over transitions) associated with this action trajectory $a_{t:t+k}$ and starting state s_t, or starting observation o_t. The purpose of the planning phase is often to identify the best action or sequence of actions to execute at the state the agent is currently in, as well as improve the agent's local estimates of the state's value, or Q-value. In settings where the problem's planning tree is finite and not too large, the optimization problem of finding the optimal action can be solved using exhaustive search of the planning tree. In general however, the complete planning tree is exponential in the planning horizon H and thus even for finite horizons $H < \infty$, exhaustive search is hardly ever tractable. In such cases other search methods are used for planning. A dominant tree-search method used in DMBRL is the Monte-Carlo tree search (MCTS) family of algorithms [31].

2.2.4 Exploration in reinforcement learning

In order to find the rewards that are part of the optimal policy, the agent must explore the environment. The faster those rewards are found, the faster the agent can be expected to learn the optimal policy. The most basic of exploration approaches is *epsilon-greedy*. In epsilon-greedy, with probability $1 - \epsilon$, the agent follows its *exploitation* policy greedily. The exploitation policy is the best policy currently learned by the agent, expected to lead to the highest expected discounted episodic return. With probability ϵ the agent explores the environment by sampling an action with uniform probability over the action space. This form of exploration, while very stable around the exploitation policy, is obviously very inefficient in terms of exploring the environment, and in many environments is insufficient to reliably learn effective policies in reasonable time [7].

The current standard for state of the art exploration attempts to achieve three main targets: first, that the exploration is locally *directed* towards regions of the state-action space that appear attractive to explore. Second, that it is directed over multiple sequential time steps, or *consistent*, enabling the agent to direct itself towards regions that are more than one action away. Last, that

the exploration is *farsighted*, in the sense that the local directed decisions take into account information from future transitions. Following terminology in other literature, we call such exploration *deep exploration*. The current standard in methods that aim to achieve such exploration rely on quantifying local estimates of epistemic uncertainty, and propagating them in time. One approach to achieve uncertainty propagation in time propagates the uncertainty through the value function with *intrinsic reward* [32]. Another approach estimates an upper bound on the propagated uncertainty from local uncertainty estimates, with the uncertainty-Bellman-equation (UBE) exploration method [8]. Both of those methods essentially aim to *learn* the uncertainty associated with a certain state or value.

2.3 Monte-Carlo Tree Search

The general MCTS algorithm is a tree-search algorithm, designed to estimate the best local action to execute in a certain state, using planning [31]. Due to the properties of the algorithm, it can be extended to provide a trajectory of actions towards a goal, as well as better value estimates, both of which may be relevant in RL. In the setting of MuZero, MCTS is used to estimate a distribution over actions to execute at every time step, as well as a Q-value prediction for every action. The *quality* or *reliability* of the Q-value estimates is expected to be highest for the Q-value associated with the *best* action: the action with respect to the MCTS has conducted the most planning. The best action is the action that maximises the objective the algorithm is searching with respect to. In our work we rely on the property that this objective does not have to be an exploitation policy, but can optimise for different targets: for example, explicitly for an exploration objective. In standard MCTS, when used in RL, the MCTS algorithm is provided access to a model of the transition function and the reward function, as well as current state to plan from, which we denote as $s_0 \equiv s_t$ for simplicity of notation. k denotes the index of a node in the MCTS planning tree, associated with a action trajectory $a_{0:k}$. s_k denotes the state, or state estimate associated with node k. MCTS approximates the values (or Q values) by iterating a four step process: 1. Trajectory-**selection**. 2. Leaf-node **expansion**. 3. Monte-Carlo (MC) **simulations**. 4. **Backup**. Each step may be governed by a separate policy.

Selection

Each iteration of MCTS starts by searching for a not-fully-expanded node k, representing the expected optimal trajectory of actions $a_{0:k}$ to execute from state s_0. A not-fully-expanded node is a node in the tree that does not have children associated with each one of the actions in the action space $a \in A$. The trajectory-selection step starts at the root $k = 0$ node. It proceeds by evaluating quantities associated with the children of the node (for example, approximated value and number of times that node has been visited in the tree), and choosing the child with the optimal function of the quantity, according to a *trajectory-selection policy* $\pi_{selection}$. A common trajectory-selection heuristic is UCT:

$$\pi_{selection}(s_k) = \arg\max_a Q(s_k, a) + 2C_p\sqrt{\frac{2\log n_k}{n_{k+1}}}$$

where n_k is the number of times the current node has been visited, n_{k+1} is the number of times each child node $k+1$ has been visited, and C_p is a constant [31]. The selection step then repeats from the child chosen on, until a not-fully-expanded node is selected.

Expansion

Once a not-fully-expanded node k has been selected, MCTS expands the node by choosing an action a_k according to some *expansion-policy* $\pi_{expansion}$, and adding the node $k+1$ that is associated with this transition according to the MCTS' model to the tree. Once the node $k+1$ has been added to the tree MCTS requires a local approximation of the value of this node $V(s_{k+1})$. In vanilla MCTS, MC simulations are used to arrive at such an approximation.

Simulation

The MC-simulation step simulates the trajectory $s_{k+1:k+\tau}$ from state s_{k+1} following some *sampling-policy* $\pi_{sampling}$, which is usually stochastic, up until a terminal state s_τ, and saves the simulated discounted return $R^\pi_{k:\tau} = \sum_{i=k+1}^{i=k+\tau} \gamma^{i-k-1} r_i$. This process repeats N times for a finite, pre-specified hyper-parameter N. The expected return is approximated as the average of all returns $\frac{1}{N}\sum_i^N R^\pi_{k:\tau}(i) \equiv \overline{R}_{t+k}$ and is saved in the node's statistics. In other variations of MCTS, other approaches for approximating the value the newly-added node $k+1$ are used. For example, MuZero uses a learned reward function $r(s_k, a_k)$ and value function $V(s_{k+1})$ to approximate the values of nodes, instead of MC simulations.

Backup

Following the simulation step, the new approximation is back-propagated up along the trajectory back to the root, in order to better estimate the value of the state represented by the root. In MuZero, where a local reward function and value function are used, the backup process is done as follows: At each node i in reverse along the action trajectory $a_{0:k}$, two quantities are updated. The first, is the sum c_i of values of the children of node i that have been back-propagated in all iterations so far. The second is the actual value v_i estimate of node i, estimated as the sum of the reward leading to this nodes and the discounted average value of the children:

$$c_i \leftarrow c_i + v_{i+1}$$
$$v_i \leftarrow r_i + \gamma \frac{c_i}{n}$$

n denotes the number of times that node has been visited in the planning tree, and thus $\frac{c_i}{n}$ provides an approximation of the value of following an optimal (or adaptively improving) policy from node i.

Termination

After L nodes have been expanded, or some other pre-specified computation resource consumed, the algorithm terminates. Usually this is followed by sampling an action to execute in the environment from an *action-selection policy* $a_t \sim \pi_{act}$. Common heuristics used for action selection are the action associated with the largest number of visitations, or the action associated with the largest value estimate, or sampling approaches based on distributions built proportionally from these quantities.

2.3.1 Trajectory selection heuristics & Upper Confidence Bound

The heuristic used for trajectory selection is a crucial component in MCTS. The directions in which the MCTS plans are the (only) directions in which the agent will be provided with improved

(or even any) estimations for the value of actions (or states). Further, this trajectory selection policy is the policy whose value the MCTS tree approximates: $V^{MCTS}(s_t) \approx V^{\pi_{selection}}(s_t)$. If the MCTS plans in directions that do not maximize the objective at hand, the resulting action-selection at the root cannot be expected to lead to optimal actions with respect to this objective. The most common objective for MCTS in RL is the standard *exploitation* objective: optimize for the policy that is associated with the highest expected discounted episodic return. To plan *robustly* with respect to this objective, many heuristics have been proposed for trajectory selection. These heuristics generally aim to achieve two goals at the same time: first, exploration *in the planning tree*, so that the estimates found by the tree are reliable. Second, the actual exploitatory objective of maximized discounted episodic return. One of the dominant heuristics for trajectory selection is Upper Confidence Bound (UCB) [33] from the realm of multi-armed bandits, or in MCTS Upper Confidence Bound for Trees (UCT) [34]. This heuristic proposes to choose the next node in the trajectory during trajectory-selection based on the sum of two quantities: the value v_k of the child node k, and a *tree-exploration* bonus, incentivizing the trajectory-selection policy to explore state-actions that have not been chosen often in the tree, based on the number of visitations n_k to each child node k, as well as the number of visitations to the parent n_{k-1}. It is important to note that this exploration is not in the environment, but entirely in the planning tree, and its purpose is to improve the approximations of the values of the nodes in the planning tree. The UCT score UCT_k of node k is computed as follows [31]:

$$UCT_k = v_k + c_{uct}\sqrt{\frac{2\ln n_{k-1}}{n_k}}$$

$c_{uct} >= 0$ is a constant which provides control over the relative weight of the tree-exploration bonus in the expression.

2.4 MuZero

MuZero [2] is a state of the art model-learning DMBRL algorithm, using MCTS for planning with the learned model. In this section we first describe the different approximators learned by MuZero, followed by a description of the methodology with which they are used in action selection in the environment, and finally we describe the training process of the algorithm, all three components modified in our work, to achieve informed, deep & directed exploration.

2.4.1 Modeling

In MuZero, the agent learns five different functions over the POMDP $M(S, A, \rho, T, R, \Omega, O)$:

1. A *representation function* $g : \Omega \to \hat{S}$. The function takes an observation $o_t \in \Omega$, and produces a latent state representation $\hat{s}_t$ for the observation:

   $$g(o) = \hat{s}_0.$$

   $\hat{s} \in \hat{S}$ represents not the real state s_t, but rather an internal abstraction learned by the agent, based on the observation o. $\hat{S}$ represents the implicitly-abstracted state-space. It is not generally assumed that the representation $\hat{S}$ learned by the agent exactly matches the state space of the environment, or a complete abstraction. Instead, this latent representation

is only trained to be useful for the algorithm in planning, with respect to maximizing the expected episodic return.

2. A *dynamics function* $f : \hat{S} \times A \to \hat{S}$. The function is implemented using a recurrent NN (RNN), such as an LSTM [35]. The RNN at planning step k takes as the recurrence input a current-state representation $\hat{s}_k \in \hat{S}$ and as regular input an action chosen at this planning-state $a_k \in A$, and outputs the expected latent state $\hat{s}_{k+1}$:

$$f(\hat{s}_k, a_k) = \hat{s}_{k+1}$$

3. An *expected-reward function* $r : \hat{S} \times A \to \mathbb{R}$. The expected reward function learns the expected reward for executing the action a_k in abstracted-state $\hat{s}_k$:

$$r(\hat{s}_k, a_k) \approx E[R(s_k, a_k)]$$

Note that as $\hat{s}_k$ does not necessarily represent well a real state s_k, this function approximation may become very unreliable, especially the deeper index k is in the planning phase.

4. A *policy function* $\pi : \hat{S} \to P(A)$, learning to associate a probability with each legal action $a_k \in A$ at internal state $\hat{s}_k$, to maximize the expected episodic return for acting according to this policy in the environment.

5. A *value function* $v : \hat{S} \to \mathbb{R}$. The value function learns the values of *latent* states $\hat{s}_k \in \hat{S}$ following a policy π: $v(\hat{s}_k) \approx V^{\pi}(\hat{s}_k) \approx V\pi(s_k)$, in planning steps k. The value function is used in the planning with MCTS to replace the MC simulations step.

2.4.2 Acting

At each environment step t, the agent executes a planning phase. The planning phase uses a variant of the MCTS algorithm to plan the next action to choose in the environment a_t, and to create a new, improved target $v_{s_t}^{target}$ to later use to train the value function $v(s_t)$. The agent executes the action a_t chosen based on the planning phase, receives a reward r_t and transitions to a new state s_{t+1}. The action a_t is chosen probabilistically with respect to visitation counts of the children of the root of the planning tree executed at time step t. The agent stores the experience in an experience replay-buffer memory, and repeats the process for the next environment step, $t + 1$. Every N steps, the agent is trained on a batch of transitions-trajectories from its replay-buffer. The MCTS variant used by MuZero uses the policy π as its *expansion policy*. The MC simulation step for not-fully-expanded node k at the end of a planning trajectory $a_{0:k}$ is replaced with a direct approximation from the value function $v(\hat{s}_k)$. The selection step is done using a variant of UCT. The action selection in the environment is done by sampling an action from a probability distribution constructed from the number of visitations to each child of the root in the planning tree. During evaluation, the action sampling can be replaced by deterministically choosing the action associated with the highest visitation count.

2.4.3 Training

The replay buffer generates targets associated with each transition in the agent's history. The target value v_t^{target} for the transition experienced at time t is the sum of the attained n discounted

rewards from this transition t forward, and the discounted final approximated value at $t+n$:

$$v_t^{target} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n v_{mcts}(\hat{s}_{t+n})$$

The value approximation $v_{mcts}(\hat{s}_{t+n})$ used in the target generation is the value approximated by the root of the tree planned at step $t+n$, planning from the state representation $\hat{s}_{t+n} = g(o_{t+n})$. The targets for the reward function $r(\hat{s}_t)$ are the observed rewards:

$$r_t^{target} = r_t$$

MuZero uses categorical representation of value and reward [2]. For this reason, the loss that is used to train the value and reward function is a cross-entropy loss. The targets for the policy network are the visitation counts of the children of the root of the tree planned at each step t. The polcy network is trained using a softmax loss as well. During learning, a batch of h-length trajectories is sampled randomly from the replay buffer based on priority. The priority is computed based on the error between the prediction of the value function and the computed value-target.

2.5 Epistemic Predictive Uncertainty In Deep Learning

Uncertainty is at the very heart of many decision making processes. Specifically, *predictive uncertainty*, the uncertainty associated with a model's prediction, is a very useful quantity when using the model's predictions for decision making, for example to decide whether the prediction is sufficiently reliable to act based on. Defining, representing and distinguishing sources of predictive uncertainty is a live field of research, both in general, as well in ML and DL specifically. There is no general consensus in the field as to how exactly uncertainty should be defined, how its sources should be categorized or distinguished, and how it should be modelled or quantified [11]. One standard approach to categorizing sources of predictive uncertainty is the division into two sources, termed **aleatoric uncertainty** and **epistemic uncertainty**. Aleatoric uncertainty refers to any uncertainty that arrives from internal stochasticity of the system, while epistemic refers to uncertainty that is caused by lack of knowledge. As an example, let us consider the toss of a die. Without using precise physics including exact initial conditions and much computation, it is standard to model the outcome of a toss of a fair die with a uniform probability distribution: $P(Y = y_i) = \frac{1}{N}$, for y_i any number on the die (the sample space Ω), Y the random variable representing the die, and N the number of faces of the die. The uncertainty about the prediction of the outcome of a die toss is formulated inside the distribution, and can be quantified using different measures: for example, it is standard to use both variance $Var[Y]$ as well as entropy $H(Y) = -\sum_{i=1}^{N} P(Y = y_i) log P(Y = y_i)$. However, regardless of how we choose to quantify the uncertainty, this uncertainty **does not** represent lack of *evidence* about the die - at least, not evidence that we expect to gain from additional die tosses. Instead, this uncertainty exists simply due to *inherent stochasticity of the system*, or inherent to how we *model* the system.

We can consider an additional example of attempting to encapsulate the uncertainty about the predictions of a die toss however. In this setting, we are not sure that the die is fair. All we are sure about is that we can model it as random: $P(Y = y) = \rho_y$, $0 < \rho_y < 1, \forall y \in \Omega$, $\sum_{y \in \Omega} \rho_y = 1$. The exact distribution however is unknown. In this setting, we can speak about *epistemic* uncertainty: while we will never be sure about the outcome of the die, we may be sure (or increasingly sure)

about the exact distribution of the outcomes, with increased number of experiences of tossing the die. This increase certainty is the motivation for the idea of *epistemic* (un)certainty. In general, both sources of uncertainty are relevant for decision making. Aleatoric uncertainty helps us to estimate the chance of possible outcomes, and epistemic uncertainty helps us to estimate confidence in our prediction of those outcomes. In this work we use this categorization of sources of uncertainty, and specifically we focus on epistemic uncertainty, in order to drive better decision making and better planning with respect to reliability of predictions, as a function of amount of experiences.

2.5.1 Modelling & quantifying epistemic predictive uncertainty

Approaches to the modelling of epistemic uncertainty range from distributional representations, the most common of which is perhaps Bayesian inference, to set based approaches [36], as well as in-between approaches such as evidence theory [37]. Following previous work [38], [39], we choose to quantify the epistemic predictive uncertainty as the variance in a distribution over predictive means $\boldsymbol{\mu}$, with respect to a specific data point x, and learned from sampled data D:

$$Var[\boldsymbol{\mu}|x, D]$$

To clarify which variables is viewed as random and which is viewed as deterministic we use bold $\boldsymbol{\mu}$ to denote random variables and regular font x, D to denote deterministic variables. We motivate the choice to predict *means* with the argument that the mean averages the aleatoric uncertainty out of the distribution. The remaining distribution over means is varied (uncertain) with respect to the variation between the different predictive means that can be learned from the data, and the probability associated with each by the distribution $p(\boldsymbol{\mu}|x, D)$. The more interactions the learner has with certain data points $(x, y) \in D$, the more *reliable* the prediction $\mu(x, D)$ becomes, and thus the learned distribution $p(\boldsymbol{\mu}|x, D)$ for the specific x is expected to become more *concentrated* around the prediction of expected-mean $\mathbb{E}[\boldsymbol{\mu}|x, D]$, which in turn is expected to result shrinking of the variance in the distribution $p(\boldsymbol{\mu}|x, D)$. Because this approach is essentially modular from *how* this distribution is learned (Bayesian approaches, frequentist approaches, and even possibly others), we do not specify any specific dependence on parameters θ. We choose to take the *variance* of the distribution $p(\boldsymbol{\mu}|x, D)$, instead of possible alternatives such as entropy, for two reasons. First, because the variance is already in the same units of the prediction (or more specifically, the standard deviation $\sqrt{Var[\boldsymbol{\mu}|x, D]}$ is in the same units), which is useful for our purposes, as described in more detail in chapter 3. Second, because propagating variance is a standard approach for propagating uncertainty, which enables us to propagate the epistemic uncertainty in MuZero's planning using standard tools of probability theory.

This approach is by no means the only approach for quantifying epistemic uncertainty, both in general, as well as specifically when using a distribution of predictive means $p(\mu|x, D)$. As an example, an alternative would be to view $\boldsymbol{D}$ as a random variable, and to model $\mu(\boldsymbol{D}, x)$ as a function of each possible realization of the data $D \sim \boldsymbol{D}$. Taking the variance in this distribution $Var[\mu(\boldsymbol{D}, x)]$ will yield a slightly different quantity, that will relate to the confidence in the prediction of mean with respect to the representability of the data $D \sim \boldsymbol{D}$. Such a quantity can also be said to be quantifying uncertainty that is epistemic. However, estimating the variance in such a distribution $Var[\mu(\boldsymbol{D}, x)]$ will likely require to have repeated samples of data $D \sim \boldsymbol{D}$. Since in this setting our main target is to improve sample-efficiency, we do not choose this approach.

2.5.2 Estimating epistemic predictive uncertainty in deep learning

The variance in a distribution of means can be estimated in different ways [39]. One standard approach is to learn an ensemble of mean estimators over different bootstraps of the same dataset [7], and then compute the variance in a prediction as the variance between the different predictors. This approach however is more in line with the formulation of epistemic uncertainty as $\mu(x, \boldsymbol{D})$. Another approach is to train an ensemble of estimators over the same dataset, but drive that their predictions are varied on untrained inputs [40], and compute the variance in the predictive distribution in the same manner. These and similar approaches [13], [41] are categorized as ensembling methods. Other approaches for epistemic uncertainty estimation range from standard Bayesian inference or Bayesian variational inference [14], [15], [42]–[45], to more evidential approaches such as state-visitation counting and pseudo counting [46],[47], exploiting properties of randomly initialized DNNs with Random Network Distillation [48], and others [38], [49]–[51].

Ensembling methods have been shown to be scalable, simple to operate, and effective at identifying out-of-distribution samples, and are a natural choice for driving exploration in the face of the unknown in RL [52]. Ensembles are not known for being reliable variance estimators however, which is the quantity we are interested in in this work. A simple and reliable epistemic uncertainty estimation method that is available in low dimensional settings is sample-counting (or state-action-visitation counting in RL). To evaluate the soundness of our method, we employ visitation counting. To further evaluate the resilience of our method to unreliable but scalable uncertainty estimation, we employ ensembles as well. Ensembles and visitation-counting are covered in more detail in the following sections.

Visitation count based epistemic uncertainty estimation

Counting visitations to state-action pairs can serve as a direct and reliable measure for epistemic uncertainty in RL. We will motivate this estimate as a direct estimate of the variance in a prediction of mean, using the example of variance in the prediction of mean reward for a transition s, a:

$$Var[\frac{1}{n}\sum_{i=1}^{n} r_{s,a}^{i}]$$

Where $r_{s,a}^{i} \sim R(s, a, \cdot)$ denotes rewards encountered in transitions from state action pair (s, a), and n is the number of samples of $r_{s,a}^{i}$ in the training set, and $\frac{1}{n}\sum_{i=1}^{n} r_{s,a}^{i}$ is the empirical mean which approximates the real mean. If the sampled rewards $r_{s,a}^{i}, i \in \{1, ..., n\}$ are assumed to be i.i.d sampled according to a distribution $r_{s,a} \sim R(s, a, \cdot) = R_{s,a}(\mu, \sigma^2)$, with constant mean μ and constant variance σ^2, the variance in the prediction of mean can be approximated as proportional to the number of visitations n to this state action pair s, a:

$$Var[\frac{1}{n}\sum_{i=1}^{n} r_{s,a}] = \frac{1}{n^2}\sum_{i=1}^{n} Var[r_{s,a}] = \frac{n}{n^2}Var[r_{s,a}] = \frac{\sigma^2}{n} \propto \frac{1}{n}$$

The first transformation is based on the fact that the variance of a sum of independent variables is the sum of the variances, and the variance of a constant a times random variable X equal the constant squared times the variance: $Var[aX] = a^2 Var[X]$. The second transformation is based on the assumption of the i.i.d. sampling of $r_{s,a}$. When $R(s, a, s') = R(s'), \forall s \in S, \forall a \in A$, which is very common in reward functions in RL, the visitation count can be simplified to counting only

state visitations. Thus, as long as the assumption that the distribution with respect to s, a has a constant mean and variance, it is reasonable to estimate the variance in mean prediction with respect to s, a as directly proportional to visitation counting.

Ensemble variance epistemic uncertainty estimation

Ensembling methods use a set of estimators to provide more reliable predictions. This set can consist of similar estimators (for example, a set of randomly initialized DNNs with identical architectures), or different estimators (for example, ensembling different ML models to predict the same quantity). When the ensemble with members $i \in \{1, \ldots, N\}$ is trained to predict means $\mu_i(x)$, in addition to taking the average prediction of the ensemble $\frac{1}{N}\sum_i^N \mu_i(x)$ as a more reliable prediction of the mean $\mu(x)$, one can take the average-squared disagreement between the different ensemble members as a direct estimate for the variance in the mean prediction:

$$Var\left[\boldsymbol{\mu}|x, D\right] \approx \frac{1}{N}\sum_i^N \left(\mu_i(x) - \frac{1}{N}\sum_i^N \mu_i(x)\right)^2$$

Due to properties inherent to DNNs, different ensemble members have a tendency to converge to different function approximations (different network weights), that tend to agree in the trained area of the input space, enforced by the loss function, and arbitrarily diverge in untrained areas. At least, this behavior is expected under limited training. This property is very useful for detecting not-previously seen areas of the input space. This property is by no means generally guaranteed however, and can be hamstrung in different settings. For example, when the network is trained on an input space with mostly-constant output, such as an RL environment with 0 rewards for every transition except the transition to the goal-state. In situations such as these the different ensemble members may learn to predict a constant output for all inputs, and thus lose all prediction diversity, even in unknown areas of the state space [40].

To overcome such detrimental effects, ensembles with randomized prior networks were proposed [40]. This method pairs an individually-randomly initialized untrained DNN to every ensemble member. The different ensemble members' loss is modified such that instead of the classic loss: $L\big(\mu_i(x), y\big)$ for $\mu_i(x)$ the prediction of ensemble member i over input x, and some target y, the loss is modified to: $L\big(\mu_i(x) - c_p f_i(x), y\big)$. $f_i(x)$ represents the (arbitrary) prediction of the prior-network f_i, and c_p is a constant deciding the weight of the prior in the computation. As a result of the modified loss, the randomized prior networks drive prediction diversity in un-encountered areas of the state space.

Chapter 3

Planning With Epistemic Uncertainty

In this chapter we describe our main contributions: 1) Probabilistic formulation of the deterministic estimators used by MuZero, to enable quantifying and utilizing epistemic uncertainty of different quantities used by MuZero specifically, and a use case for those often used in model-learning DMBRL in general (section 3.1). 2) Development of practical epistemic uncertainty propagation methodology in planning trees (section 3.2). 3) Extensions to MCTS to allow MCTS to propagate uncertainty (section 3.3). 4) Our proposed method, *planning for exploration*, harnessing the propagated uncertainty for effective and efficient deep exploration (section 3.4). 5) Adaptations to MuZero's target generation, to enable stable learning from off-policy, exploratory trajectories with MuZero's on-policy-targets' learning (section 3.5).

3.1 Probabilistic model formulation

In this chapter, we are interested in developing a computable approximation for the epistemic uncertainty, as it propagates through *planning*. We denote planning k steps into the future from step t with *planning-trajectories*: for example, $a_{t:t+k}$ denotes a trajectory of planned actions from real step t to k imaginary planning steps into the future. o_t denotes the observation received at step t, based on which the planning phase is initialized and the current-state is estimated. To reduce clutter in notation we remove the index t from the notation of planning trajectories, and instead denote the trajectory with $a_{0:k}$. In order to utilize the uncertainty measure we opted for in section 2.5.1, the variance in a probability distribution over predictive means $\boldsymbol{\mu}$, conditional on input x and training data D:

$$Var[\boldsymbol{\mu}|x, D]$$

We choose to view MuZero's deterministic point-estimators as estimators of *mean*, rather than arbitrary point estimators. This decision is natural for two of MuZero's predictors: the expected-reward function $r(\hat{s}_t, a_t) \approx \mathbb{E}[R(s_t, a_t)]$, and the value function $v(\hat{s}_t) \approx V^{\pi}(s_t)$, which already predict an expected-reward and expected-return, respectively. This choice is less obvious for the dynamics

predictions $f(\hat{s}_t, a_t) = \hat{s}_{t+1}$, which is implicitly deterministic because the function f is a (deterministic) DNN. Nonetheless, we choose to model $f(\hat{s}_t, a_t) = \hat{s}_{t+1} \approx \mathbb{E}[s_{t+1}|s_t, a_t]$, as approximating an expected-transition in the *true* model of the environment. This choice will enable us to propagate the epistemic uncertainty in state through the planning using the same mathematical machinery used for reward and value.

The *propagated* uncertainty through a planning trajectory $a_{0:k}$ of length k starting at observation o can be formulated as the variance in the distribution $p(\boldsymbol{\mu}_k|o, a_{0:k}, D)$, for the distribution over predictions $\boldsymbol{\mu}_k$ at planning step k. In the rest of this section, we develop computable approximations for such variance, using the variance in the *local* transition prediction-distribution $p(\boldsymbol{\mu}_k|a_k, \hat{\boldsymbol{s}}_k, D)$. To further reduce clutter in notation, we remove the dependence on D in the notation, as well as the $\hat{\cdot}$ symbol for abstracted states: $s_k := \hat{s}_k$. All distributions considered in this chapter are learned from data and thus have dependence on D. All states s_k considered in the rest of this section are inside the agent's planning, and thus are all abstracted states. In addition, in the next section we will use both the regular variance $Var[Y|X = x]$, which is a number, as well as the conditional variance $Var[Y|X]$ [53], which is a random variable. In order to clarify which instance is which, we introduce the notation ; to introduce conditionality on *deterministic* quantities. For example: the distribution $p(\boldsymbol{\mu}; x, D)$ is the distribution of $\boldsymbol{\mu}$ conditional on deterministic variables x, D. In contrast, the distribution $p(\boldsymbol{s}_k; a_k|\boldsymbol{s}_{k-1})$ is the distribution of the random variable $\boldsymbol{s}_k; a_k$ conditional on the random variable $\boldsymbol{s}_{k-1}$. To propagate the uncertainty in reward, value and state-abstraction prediction through the planning, we consider the following four distributions:

1. $p(\boldsymbol{s}_k; a_{0:k-1}, o)$ represents the probability distribution over predicted states $s_k \in \mathbb{R}^m$ at planning step k, conditional on the action trajectory $a_{0:k-1}$, as well as the initial observation o. As these are mean states, the information in the variance of this distribution reflects epistemic uncertainty.

2. $p(\boldsymbol{v}_k; a_{0:k-1}, o)$ represents the probability distribution over predicted values $\boldsymbol{v}_k \in \mathbb{R}$ at planning step k when k is the end of a planning trajectory, conditional on the action trajectory $a_{o:k-1}$, as well as the initial observation o.

3. $p(\boldsymbol{r}_k; a_{1:k}, o)$ represents the probability distribution over predicted *expected* rewards $\boldsymbol{r}_k \in \mathbb{R}$ attained by executing action a_k at planning step k, conditional on the action trajectory $a_{0:k}$ as well as the initial observation o.

4. $p(\boldsymbol{v}_k; a_{0:k+h}, o)$ represents the probability distribution over predicted values $\boldsymbol{v}_k \in \mathbb{R}$ at planning step k, when k is a decision *along* the planning trajectory $a_{0:k+h}$, and the end of the planning trajectory is planning step $k + h$.

In the next section we develop approximations for the variance in each one of these distributions, as it propagates through planning.

3.2 Uncertainty propagation in planning with a learned model

This section develops computable update rules for the propagation of predictive epistemic uncertainty, modelled as variance, in planning with a learned model. We first develop approximations for the the variance of $p(\boldsymbol{s}_k; a_{0:k-1}, o)$ (section 3.2.1). We then proceed to develop approximations for the variances of $p(\boldsymbol{v}_k; a_{0:k-1}, o)$ (section 3.2.2) and $p(\boldsymbol{r}_k; a_{0:k}, o)$ (section 3.2.3), based on the

approximation for the variance of $p(\boldsymbol{s}_k; a_{0:k-1}, o)$. Finally, we develop a computable backwards-update for the variance of $p(\boldsymbol{v}_k; a_{0:k+h}, o)$ (section 3.2.4) based on the variances of the distributions $p(\boldsymbol{s}_k; a_{0:k-1}, o)$, $p(\boldsymbol{v}_{k+h}; a_{0:k+h-1}, o)$ and $p(\boldsymbol{r}_k; a_{0:k}, o)$. The main tool used in this section is the law of total variance [54]. As a result, while the target of this work is to propagate specifically epistemic uncertainty as a variance, this framework can be used to approximate the propagation of any uncertainty that can be formulated as a variance and represented in a sufficiently similar manner.

3.2.1 State uncertainty

For the purpose of investigating explicitly how state *uncertainty* propagates in planning, we are interested only in the (co-)variance of the distribution $Cov(\boldsymbol{s}_k; a_{0:k-1}, o)$. In order to clarify when a certain formulation is a result of direct equality $=$, approximation $\approx$, definition $:=$, or modelling choice $\equiv$, all four notations will be used. In order to arrive at a computable approximation for the covariance at planning step k, we split the co-variance into more directly-approximable quantities, using the law of total variance:

$$Cov(\boldsymbol{y}) = \mathbb{E}\big[Cov(\boldsymbol{y}|\boldsymbol{x})\big] + Cov\big(\mathbb{E}[\boldsymbol{y}|\boldsymbol{x}]\big)$$

Note that as both $\boldsymbol{x}, \boldsymbol{y}$ are random variables, $\mathbb{E}[\boldsymbol{y}|\boldsymbol{x}]$ is the *conditional* expectation [55], and $Cov[\boldsymbol{y}|\boldsymbol{x}]$ is the *conditional* co-variance. As $a_{0:k-1}, o$ are not considered random variables, the expectation $\mathbb{E}[\boldsymbol{s}_k; a_{0:k-1}, o]$ is the regular expectation with respect to the random variable $\boldsymbol{s}_k; a_{0:k-1}, o$. We use $\boldsymbol{y} = \boldsymbol{s}_k; a_{0:k-1}, o$ and $\boldsymbol{x} = \boldsymbol{s}_{k-1}; a_{0:k-2}, o$, and formulate:

$$\Sigma_{\boldsymbol{s}_k; a_{0:k-1}, o} := Cov(\boldsymbol{s}_k; a_{0:k-1}, o)$$

$$= \mathbb{E}[Cov(\boldsymbol{s}_k; a_{0:k-1}, o|\boldsymbol{s}_{k-1}; a_{0:k-2}, o)] + Cov(\mathbb{E}[\boldsymbol{s}_k; a_{0:k-1}, o|\boldsymbol{s}_{k-1}; a_{0:k-2}, o])$$

$$= \mathbb{E}[Cov(\boldsymbol{s}_k; a_{k-1}|\boldsymbol{s}_{k-1}; a_{0:k-2}, o)] + Cov(\mathbb{E}[\boldsymbol{s}_k; a_{k-1}|\boldsymbol{s}_{k-1}; a_{0:k-2}, o]) \tag{3.1}$$

Where we assume independence of $\boldsymbol{s}_k$ from $a_{0:k-2}, o$ conditional on $\boldsymbol{s}_{k-1}, a_{k-1}$. This assumption is implicitly made by MuZero, which bases all of its predictions on the last state to be predicted $\boldsymbol{s}_{k-1}$ and the action opted for at this state a_{k-1}. We model the conditional expectation from the second term $\mathbb{E}[\boldsymbol{s}_k; a_{k-1}|\boldsymbol{s}_{k-1}; a_{0:k-2}, o]$ as *the distribution of the outputs of* f on the random variable $\boldsymbol{s}_{k-1}; a_{0:k-2}, o$ and the specific action along the trajectory a_{k-1}:

$$\mathbb{E}[\boldsymbol{s}_k; a_{k-1}|\boldsymbol{s}_{k-1}; a_{0:k-2}, o] \equiv f\big((\boldsymbol{s}_{k-1}; a_{0:k-2}, o), a_{k-1}\big)$$

In general the expectation $\mathbb{E}[\boldsymbol{s}_k; a_{k-1}|\boldsymbol{s}_{k-1}; a_{0:k-2}, o]$ would need to be approximated with the means of many randomized outputs (for example by sampling with a dropout network [14]). However, since MuZero's state transitions are determinsitic, it is sufficient to use the output of the network.

We denote the conditional covariance $Cov(\boldsymbol{s}_k; a_{k-1}|\boldsymbol{s}_{k-1}; a_{0:k-2}, o) := \Sigma_{\boldsymbol{s}_k; a_{k-1}|\boldsymbol{s}_{k-1}}$ to simplify notation. Note the difference between $\Sigma_{\boldsymbol{s}_k; a_{0:k-1}, o}$, the *propagated* co-variance, which is the quantity for which we develop an approximation, and $\Sigma_{\boldsymbol{s}_k; a_{k-1}|\boldsymbol{s}_{k-1}}$, the *local* conditional covariance. We can now proceed to formulate a first approximation to the complete expression for the propagated covariance from equation 3.1:

$$\mathbb{E}\big[Cov(\boldsymbol{s}_k; a_{k-1}|\boldsymbol{s}_{k-1}; a_{0:k-2}, o)\big] + Cov\big(\mathbb{E}[\boldsymbol{s}_k; a_{k-1}|\boldsymbol{s}_{k-1}; a_{0:k-2}, o]\big) \equiv$$

$$\mathbb{E}[\Sigma_{\boldsymbol{s}_k;a_{k-1}|\boldsymbol{s}_{k-1}}] + Cov\Big(f\big((\boldsymbol{s}_{k-1};a_{0:k-2},o),a_{k-1}\big)\Big)$$

We will now formulate approximations for each of the two terms. We first motivate an approximation for the expectation over the conditional covariance:

$$\mathbb{E}[\Sigma_{\boldsymbol{s}_k;a_{k-1}|\boldsymbol{s}_{k-1}}]$$

$\mathbb{E}[\Sigma_{\boldsymbol{s}_k;a_{k-1}|\boldsymbol{s}_{k-1}}]$ denotes the *expected local uncertainty* - the average uncertainty associated with s_k conditional on the distribution of different possible s_{k-1}. This is a quantity we can approximate using local uncertainty estimation mechanisms (for example, the covariance between ensemble members predicting the state s_k). In principle, our uncertainty mechanisms may be designed to approximate $\Sigma_{\boldsymbol{s}_k;a_{k-1},s_{k-1}}$, where s_{k-1} is *not* a random variable (see section 2.5.1). However, we assume that the average-covariance can be approximated using this estimate: $\Sigma_{\boldsymbol{s}_k;a_{k-1},s_{k-1}} \approx \mathbb{E}[\Sigma_{\boldsymbol{s}_k;a_{k-1}|\boldsymbol{s}_{k-1}}]$. We denote this estimate with: $\Sigma^{local}_{\boldsymbol{s}_k} \approx \mathbb{E}[\Sigma_{\boldsymbol{s}_k;a_{k-1}|\boldsymbol{s}_{k-1}}]$, where $\Sigma^{local}_{\boldsymbol{s}_k}$ denotes any appropriate local uncertainty-estimation mechanism. We approximate the second term, the covariance over the modeled conditional-expectation $Cov\Big(f\big((\boldsymbol{s}_{k-1};a_{0:k-2},o),a_{k-1}\big)\Big)$ using first-order Taylor approximation [56]:

$$Cov\Big(f\big((\boldsymbol{s}_{k-1};a_{0:k-2},o),a_{k-1}\big)\Big) \approx J_s \Sigma_{\boldsymbol{s}_{k-1};a_{0:k-2},o} J_s^T$$

J_s denotes the Jacobian matrix for $f(s_{k-1},a_{k-1})$, and $\Sigma_{\boldsymbol{s}_{k-1};a_{0:k-2},o}$ the covariance approximation computed at the previous step $k-1$. For a more elaborate motivation for the approximation of the propagation of the covariance of a random variable through a nonlinear function, see appendix A. **We arrive at a final computable approximation for the state-uncertainty propagated in a planning trajectory $a_{0:k-1}$:**

$$\Sigma_{\boldsymbol{s}_k;a_{0:k-1},o} := Cov(\boldsymbol{s}_k;a_{0:k-1},o) \approx \Sigma^{local}_{\boldsymbol{s}_k} + J_s \Sigma_{\boldsymbol{s}_{k-1};a_{0:k-2},o} J_s^T \tag{3.2}$$

The covariance $\Sigma_{\boldsymbol{s}_0|o}$ at the root of the planning tree can be approximated by applying the same principle of using a local uncertainty estimation mechanism on $g(o)$.

3.2.2 Leaf-node value uncertainty

We proceed to use the approximated propagated covariance in the state $\Sigma_{\boldsymbol{s}_k;a_{0:k-1},o}$, to develop a computable approximation to the variance of the state-estimate at the end of a planning trajectory $\boldsymbol{v}_k;a_{0:k-1},o$, again using the law of total variance:

$$Var\big[\boldsymbol{v}_k;a_{0:k-1},o\big] = \mathbb{E}\Big[Var\big[\boldsymbol{v}_k|\boldsymbol{s}_k;a_{0:k-1},o\big]\Big] + Var\Big[\mathbb{E}\big[\boldsymbol{v}_k|\boldsymbol{s}_k;a_{0:k-1},o\big]\Big]$$

And assuming independence of $\boldsymbol{v}_k$ from $a_{0:k-1},o$ conditional on $\boldsymbol{s}_k$, which is in line with MuZero's functional approximation $v(s_k)$, which only take s_k as input to compute the value at the end of a planning trajectory k. We approximate the *expectation of the conditional variance*:

$$\mathbb{E}[Var(\boldsymbol{v}_k|\boldsymbol{s}_k;a_{0:k-1},o)] \approx (\sigma^{local}_{\boldsymbol{v}_k})^2$$

with the same motivation and process used in section 3.2.1, with a local uncertainty-estimation mechanism. We model the conditional expectation $\mathbb{E}[\boldsymbol{v}_k|\boldsymbol{s}_k; a_{0:k-1}, o]$ as the output of the value network on the distribution of states $\boldsymbol{s}_k$, as follows: $\mathbb{E}[\boldsymbol{v}_k|\boldsymbol{s}_k; a_{0:k-1}, o] \approx v(\boldsymbol{s}_k; a_{0:k-1}, o)$, again with the same motivation used in section 3.2.1. We can now formulate the variance of the value of the state approximated at planning step k in those terms:

$$Var[\boldsymbol{v}_k; a_{0:k-1}, o] \approx (\sigma^{local}_{\boldsymbol{v}_k})^2 + Var[v(\boldsymbol{s}_k; a_{0:k-1}, o)]$$

The term $Var[v(\boldsymbol{s}_k; a_{0:k-1}, o)]$, approximating the propagation of the covariance in the state through the value distribution, can again be approximated with first order Taylor expansion:

$$Var[v(\boldsymbol{s}_k; a_{0:k-1}, o)] \approx J_v \Sigma_{\boldsymbol{s}_k; a_{0:k-1}, o} J_v^T$$

Where J_v is the one-dimensional Jacobian matrix of $v(s_k)$, and an approximation for $\Sigma_{\boldsymbol{s}_k; a_{0:k-1}, o}$ has been developed in equation 3.2. Finally, a complete computable expression for an approximation of the forward update step for the propagation of the variance into the value function can be formulated, which we denote by $(\sigma^{leaf}_{\boldsymbol{v}_k})^2$:

$$(\sigma^{leaf}_{\boldsymbol{v}_k})^2 := Var[\boldsymbol{v}_k; a_{0:k-1}, o] \approx (\sigma^{local}_{\boldsymbol{v}_k})^2 + J_v \Sigma_{\boldsymbol{s}_k; a_{0:k-1}, o} J_v^T \tag{3.3}$$

3.2.3 Reward uncertainty

We follow the same process again to build an approximation for the propagated variance in the distribution over rewards $\boldsymbol{r}_k; a_{0:k}, o$ at planning step k:

$$(\sigma_{\boldsymbol{r}_k})^2 := Var[\boldsymbol{r}_k; a_{0:k}, o] = \mathbb{E}\big[Var[\boldsymbol{r}_k; a_k|\boldsymbol{s}_k; a_{0:k-1}, o]\big] + Var\big[\mathbb{E}[\boldsymbol{r}_k; a_k|\boldsymbol{s}_k; a_{0:k-1}, o]\big]$$

$$\approx (\sigma^{local}_{\boldsymbol{r}_k})^2 + Var(r((\boldsymbol{s}_k; a_{0:k-1}, o), a_k)) \approx (\sigma^{local}_{\boldsymbol{r}_k})^2 + J_r \Sigma_{\boldsymbol{s}_k; a_{0:k-1}, o} J_r^T \tag{3.4}$$

Where similarly, $(\sigma^{local}_{\boldsymbol{r}_k})^2$ is the uncertainty estimation applied to $r(\boldsymbol{s}_t, a_t)$, J_r is the one-dimensional Jacobian matrix of $r(\boldsymbol{s}_t, a_t)$ and $\Sigma_{\boldsymbol{s}_k; a_{0:k-1}, o}$ has been approximated in equation 3.2.

3.2.4 Tree-node value distribution

Using the approximation for the forward-propagated variance of the reward value and state computed in the previous sections, we will now construct a computable *backward* propagation update, to approximate the effect of planned trajectories on the variance in the values of states s_k *along* a planning trajectory $a_{0:k+h}$, where $k+h$ is the current end of this planning trajectory. The value of state s_k following a policy π can be formulated as: $V^\pi(s_k) = \mathbb{E}_\pi[r_k + \gamma V^\pi(s_{k+1})]$, where $\mathbb{E}_\pi[\_]$ denotes the expectation with respect to the Markov chain imposed by actions sampled from the policy π, and γ is the discount factor. We now assume that the action trajectory $a_{0:k+h}$ follows some policy π. We represent the resulting $V^\pi(s_k)$ as a distribution $\boldsymbol{v}_k; a_{0:k+h}, o$, representing the uncertainty in the agents' prediction of the value at planning step k, with policy choosing actions $a_{0:k+h}$. We model the distribution as follows:

$$\boldsymbol{v}_k; a_{0:k+h}, o = (\boldsymbol{r}_k + \gamma \boldsymbol{v}_{k+1}); a_{0:k+h}, o = (\boldsymbol{r}_k; a_{0:k}, o) + \gamma(\boldsymbol{v}_{k+1}; a_{0:k+h}, o)$$

First by definition of the value function, and second based on an assumption of conditional independence of $\boldsymbol{r}_k$ from $\boldsymbol{v}_{k+1}$ conditional on $a_{0:k+h}, o$, as well as independence of $\boldsymbol{r}_k$ from $a_{k+1:k+h}$ conditional on $a_{0:k}$, modelling the reward estimate as depending only on past actions, and not future actions. The variance of the distribution $Var[\boldsymbol{v}_k; a_{0:k+h}, o]$ can be expressed as follows:

$$(\sigma^{node}_{\boldsymbol{v}_k})^2 := Var[\boldsymbol{v}_k; a_{0:k+h}, o] = Var\big[(\boldsymbol{r}_k; a_{0:k}, o) + \gamma(\boldsymbol{v}_{k+1}; a_{0:k+h}, o)\big]$$

Using the property that the variance of a sum of independent random variables equals the sum of the individual variances, as well as properties of the variance of a product of a constant and a random variable:

$$(\sigma^{node}_{\boldsymbol{v}_k})^2 = Var[\boldsymbol{r}_k; a_{0:k+h}, o] + \gamma^2 Var[\boldsymbol{v}_{k+1}; a_{0:k+h}, o]$$

We proceed to approximate the variance in the reward with the expression developed in equation 3.4:

$$\sigma^{node}_{\boldsymbol{v}_k} \approx (\sigma_{\boldsymbol{r}_k})^2 + \gamma^2 Var[\boldsymbol{v}_{k+1}; a_{0:k+h}, o] \tag{3.5}$$

This term can be computed recursively backwards, starting from the state approximation at the end of the planning trajectory s_{k+h+1}, for which $Var[\boldsymbol{v}_{k+h+1}; a_{0:k+h}, o]$ has been approximated at equation 3.3. We now have a complete set of update rules for the propagation of uncertainty along planning trajectories in a planning tree - first forwards, to approximate the effect of state-uncertainty estimates on value and reward uncertainty estimates, and then backwards, to approximate the effect of the uncertainty in future reward and values on value uncertainty estimates along a planning trajectory. In the following sections we first propose modifications for MCTS to enable propagating the uncertainty based on the approximations developed, and then describe our approach for using the propagated uncertainty to achieve deep exploration.

3.3 Propagating uncertainty in Monte-Carlo Tree Search

While variations of MCTS sometimes consider aleatoric uncertainty, they do not generally consider epistemic uncertainty: the model used for planning is usually considered to be exact. We extend this perception with the assumption that the model or parts of it are not necessarily exact, and have epistemic uncertainty associated with them. This assumption is naturally relevant in settings of planning with *learned* models, but is applicable in general for any setting where non-zero epistemic uncertainty can be quantified in the model. Building on this assumption, we propagate epistemic uncertainty in the different components of the model (reward prediction and dynamics), as well as the value prediction, based on the approximations developed in section 3.2. In order to incorporate these approximations into MCTS, we modify the following steps in the MCTS algorithm:

1. In the **expansion** step, we compute the local uncertainties associated with the value, reward and state estimates based on the uncertainty estimates chosen. We compute the complete state, value and reward uncertainty estimates according to the equations developed in section 3.2. We save these quantities with the rest of the information stored with the node.

2. In the **backup** step, in addition to the value estimate, we back-propagate the value-uncertainty estimate following the update rules developed in section 3.2.

3. We introduce a new function that estimates the average epistemic uncertainty in a value prediction of a node in a similar manner to the computation that estimates the average value (see section 2.3).

A description of the algorithm can be found in algorithm 1.

Algorithm 1 Modified MCTS for uncertainty propagation

function EXPAND(v_k, r_k, $\sigma^{r_k}_{\text{local}}$, $\sigma^{v_k}_{\text{local}}$, $\Sigma^{local}_{\boldsymbol{s}_k}$, J_s, J_r, J_v)
    Execute unmodified MuZero-MCTS expansion step
    $\Sigma^{node}_{\boldsymbol{s}_k} \leftarrow \Sigma^{local}_{\boldsymbol{s}_k} + J_s \Sigma^{parent}_{\boldsymbol{s}_{k-1}} J_s^T$
    $\sigma^{node}_{\boldsymbol{v}_k} \leftarrow \sigma^{local}_{\boldsymbol{v}} + J_v \Sigma^{node}_{\boldsymbol{s}_k} J_v^T$
    $\sigma^{node}_{\boldsymbol{r}_k} \leftarrow \sigma^{local}_{\boldsymbol{r}_k} + J_r \Sigma^{node}_{\boldsymbol{s}_k} J_r^T$
function BACKUP($\sigma^{leaf}_{v_k}$, *search_path*, unmodified original parameters)
    Execute unmodified MuZero-MCTS backpropagation step
    $\sigma_{\boldsymbol{v}} \leftarrow \sigma^{leaf}_{v_k}$
    for *node* `in` *reverse*(*search_path*) **do**
        $\sigma^{node}_{\boldsymbol{v}} \leftarrow \sigma^{node}_{\boldsymbol{v}} + \sigma_{\boldsymbol{v}}$
        $\sigma_{\boldsymbol{v}} \leftarrow \sigma^{node}_{\boldsymbol{r}} + \gamma^2 \sigma_{\boldsymbol{v}}$
function GETVALUEUNCERTAINTY(node)
    return $\sigma^{node}_{\boldsymbol{v}} / node_c$ ▷ $node_c$ denotes the visitation count of the node in the tree

3.4 Planning for deep exploration by harnessing epistemic uncertainty

In this section we describe our methodology for incorporating uncertainty into the agent's planning to achieve deep exploration, which we term *planning for exploration*. During exploration episodes, in the trajectory-selection step in the MCTS planning phase, we incorporate the total value uncertainty $\sigma_{\boldsymbol{v}_k} \approx Var[v_k]$ associated with node k into the UCB computation to incentivize the tree to plan in the direction of *most promising exploration*: high expected return as well as high associated uncertainty. We incorporate uncertainty into the UCB computation as follows:

$$\text{UCB}_\sigma(\text{Node}_k) = \text{UCB}(\text{Node}_k) + c_\sigma \sigma^{node}_{\boldsymbol{v}_k} \tag{3.6}$$

c_σ is a hyperparameter, deciding how much standard deviation $\sigma^{node}_{\boldsymbol{v}_k}$ is taken into account in the UCB_σ computation.

To achieve deep exploration in the environment, we do not modify MuZero's action selection in the environment directly, but execute the action selection process in accordance with the *planning for exploration MCTS tree*, rather than a regular, planning-to-exploit MCTS planning tree. As described in section 2.4.2, MuZero's action selection is done probabilistically with respect to the visit-counts of the children of the root. As our planning induces different visitation counts to the children of the root that are expected to reflect the direction in the environment of most promising exploration, we expect these visitation counts to be sufficient to achieve deep and directed exploration.

3.5 Dedicated exploration with on-policy training

The main caveat of on-policy algorithms is their inability to learn from trajectories in the environment that were not sampled from their own policy. Whether MuZero is precisely an on-policy algorithm or not is under debate [57]. However, a detailed look at the targets generated to train the policy and value functions illustrates the problem of attempting to learn from off-policy, exploratory trajectories. As described in section 2.4.3, the value targets used by MuZero v_{target} are *n-step targets*. This induces that the algorithm is trained to learn to approximate the value $v^{\pi}(s)$ of the policy π *that has been executed in the environment*. If the policy executed in the environment was not exploitatory (for example, because it was an exploratory policy aiming to find new information, rather than optimize for maximum return), the approximation $v^{\pi}(s)$ approximates the value of a policy which is a mix of exploratory and exploitatory policies. This effect can be expected to slow learning, cause instability, or even prevent the agent from converging to certain policies, depending on the precise interactions that the agent samples from its reply buffer.

We illustrate this with the following example: an agent is standing in a hallway with two doors. Opening the door to the right yields reward of 1, and terminates the episode. The agent has tried this action sufficiently many times, and predicts the value and reward correctly. Opening the door to the left yields reward of -1 and also terminates the episode. However, the left door has not been opened before by the agent. A sufficiently-exploratory policy (for example, using sufficiently large c_{σ}) will prioritize opening the door on the left, due to the high epistemic uncertainty associated with it. If the agent then utilizes n-step targets to learn the value of being in the state between the two doors, and it samples both trajectories equally often, it can learn that the value of following its policy in this position is 0, which is both correct as well as hindering learning. It is correct, because if the agent indeed takes either door in this position with equal probability, the average value is indeed 0. This learned value of 0 is not useful either for exploration or for exploitation, and the effect it has on either policy is entirely detrimental. Learning this value is expected to be preventable: if the agent can observe that the decision to open the door on the left is "worse", it should be able to learn to open the door on the right, regardless of the actions taken in the trajectories existing in the replay buffer. Such capacity to learn conclusions that potentially-conflict with the policy executed in the environment is what we aim to achieve.

Just as this challenge exists with the on-policy *value-targets* used by MuZero, a similar challenge exists with the on-policy *policy-targets* used by MuZero. As described in section 2.4.3, the policy network $\pi(s_t)$ is trained to predict the relative visitation-counts of the children of planning trees executed at state s_t. Trees that have been planning for exploration cannot be expected to necessarily coincide with trees that have been planning for exploitation (if they do, there would be no point to plan for exploration). As a result, the policy-targets that are generated based on exploratory-planning trees can have adverse effects on the agent's learning: they teach the agent to execute an exploratory policy. If the exploratory policy yielded worse returns than the exploitatory policy, this effect can be expected to be detrimental. In the following sections we propose different solutions to these problems. In chapter 5 we conduct a detailed ablation testing to evaluate the effects of the different solutions proposed. Before proceeding to describe the proposed modifications, we will define several terms that will be useful in the following sections: *exploration episodes*, *regular episodes*, *positive exploration trajectories*, and *negative exploration trajectories*.

Exploration episode is any episode that followed an exploration policy throughout the episode, or for extended periods.

Regular episode or exploitation episode, or later *reevaluation* episode is any episode that followed

an exploitatory policy, either greedily, or with limitedly-stochastic action selection.

Positive exploration trajectory We term *exploration* trajectories any trajectories that followed a dedicated-exploration policy, such as the one induced by executing actions based on planning-trees induced by the *planning for exploration* method. We consider an exploration trajectory to be *positive* if the n-step discounted-return $v^{n-step}_{target}(s_t)$ estimated from this trajectory starting at state s_t is *higher* than the value $v^{exploit}_{mcts}(s_t)$ the agent approximates for this state s_t with its exploitation policy induced by the exploitatory planning tree:

$$v^{n-step}_{target}(s_t) > v^{exploit}_{mcts}(s_t)$$

Negative exploration trajectory Similarly, we consider an exploration trajectory to be *negative* if the value is *smaller or equal*. The idea is that exploration trajectories that found new useful information *should* modify the agent's exploitation policy, and exploration trajectories that did not *should not* modify the agent's exploitation policy. We will now proceed to describe the different modification to the value and policy targets used to train MuZero, to enable MuZero to learn from exploratory trajectories that can be either positive or negative.

3.5.1 Double-planning in exploration

The first modification proposed in this work to overcome adverse effects of off-policy targets generated from exploratory trajectories is to **execute two separate planning trees at each planning step in exploration episodes**: a *planning-for-exploration* planning tree, and a *regular, exploitatory* planning tree. Exploration episodes can be executed for every episode that is not executed to evaluate the agent's current policy (evaluation episodes), or as specific episodes during the learning of the agents, as will be discussed in section 3.5.3. The actions chosen during exploration follow the policy generated by the planning-for-exploration tree. Two sets of statistics are saved from planning with two trees, instead of one: the exploratory root's statistics, and the regular-root's statistics. This allows us to decouple both the policy targets as well as the value targets generated from this trajectory from the consequences of the policy generated this trajectory in the environment. The policy targets can now be computed based on the visitations to the *regular* planning tree, instead of an exploratory tree, and are thus now *exploitation*-policy targets. The bootstrapped-value used to compute the value at the end of the trajectory of the n-step value target $v_{mcts}(s_{t+n})$, can now come from a regular, *exploitatory* planning trees:

$$v^{n-step}_{target}(s_t) = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n v^{exploit}_{mcts}(s_{t+n})$$

The rewards r_{t+i} used to compute the value target are still the rewards recieved from executing the exploratory policy in the environment, however. While this is not expected to be sufficient to overcome the challenges associated with the n-step value targets, we expect this to already provide a stabilizing effect on the learning process, by generating policy targets that are in-line with the *exploitatory* policy we ultimately intend to learn, as well as value targets that are at least bootstrapped exploitatorily.

3.5.2 Zero-step targets and max targets

We remain with two main challenges: 1) Choosing the type of policy target: the policy targets from double-planning can either follow exploratory planning or exploitatory planning. In *negative*

exploration episodes, the *exploitatory* policy targets are expected to induce stability. However, in *positive* exploration trajectories, the *exploratory* policy targets are expected to induce faster learning. 2) The value targets generated from negative exploration trajectories are still n-step targets and expected to have an adverse effect on the agent's value approximation, as discussed at the beginning of this chapter. We propose the same approach to overcoming both challenges, starting from the value targets.

An alternative to n-step targets is *zero-step* targets. In zero-step targets, the value target v_{target} can be computed as follows:

$$v_{target}^{0-step}(s_t) = v_{mcts}^{exploit}(s_t)$$

As long as the estimate $v_{mcts}(s_t)$ comes from an exploitatory planning tree, these targets are on-policy, even when computed in exploration trajectories. If the estimate of the tree $v_{mcts}(s_t)$ is sufficiently better than the prediction of the value network $v(s_t)$, these zero-step targets can be expected to useful targets. n-step targets are known however for inducing very rapid learning. Therefor, simply replacing them by zero-step targets may have an insufficient beneficial effect to justify their use, or even an overall detrimental effect. To enjoy the best of both worlds - swiftly-learning, reliable n-step targets, and on-exploitatory-policy 0-step targets, we propose an additional step: the *max(0-step, n-step)* target, or simply *max value targets*. For every trajectory following an exploratory policy that exists in the replay buffer, we can compare the n-step value approximation $v_{target}^{n-step}(s_t)$ with the exploitatory 0-step approximation of the agent, $v_{target}^{0-step}(s_t)$. For all exploration trajectories where $v_{target}^{n-step}(s_t) > v_{target}^{0-step}(s_t)$ (positive exploration trajectories), the n-step targets can be used, to anchor the agent with the effects of choosing a better trajectory in the environment. For all other exploration trajectories, the 0-step exploitatory-policy targets should be used, to not confuse the agent with misleading, off-exploitatory-policy value targets.

The same idea can be employed for policy targets in a similar manner: in all positive exploration trajectories, the agent can use as target the *exploratory*-tree policy targets, indicating that the exploration-policy found something promising, and should be followed in exploitation. In negative exploration trajectories, for stability, the *exploitatory*-tree policy targets can be used. We term this target-generation method as *max policy targets*.

3.5.3 Decoupling reevaluation episodes from exploration episodes

In order to further reduce the adverse effect of zero-step value targets, we can introduce exploitatory episodes into the agent's training loop, to allow the agent to reevaluate the value estimates associated with its current exploitation policy. Without executing any reevaluation episodes, the agent will not be able to develop reliable estimates for the values associated with its current exploitation policy, because all the trajectories it can learn from will be exploratory. In turn, this can make any exploratory trajectory resulting in an n-step value target that is *lesser* than some early, unreliable value estimates seem like negative exploration trajectory, despite the value estimate being potentially completely unreliable. Consider for example an environment where the agent receives a reward of -1 for every time step, and the goal is the only real terminal state. All values of all states of all optimal policies for every possible goal in this environment that requires executing more than one step, will be negative. If the agent's original value estimates are non-negative, or simply larger than the values of the optimal policy, the agent will struggle to learn from exploratory trajectories that lead it to the goal. On the other hand, alternating reevaluation episodes every certain number

of exploration episodes will enable the agent to explicitly gather data in order to improve the estimates of the exploitatory policy. We expect alternating the types of learning episodes in this way to induce better ability to contrast and identify actually-positive exploration trajectories, compared to exploitation trajectories. There are many possibilities for alternating between reevaluation and exploration episodes, such as alternating episode by episode, dynamically reducing the portion of exploration episodes, and many others. For simplicity, in this work we chose to alternate between episodes evenly, and switch learning-episode type every 10% of the training-steps' budget, starting from exploitation episodes.

Chapter 4

Experimental Setup

This section describes the methodology applied to evaluating the framework, method and adaptations proposed in this work. We open with a description of engineering efforts that were required to tailor the uncertainty mechanisms to MuZero. We follow with a description of the agents that are used to evaluate the method. We proceed to describe and motivate the tasks the agents were evaluated against. Last, we outline and motivate the evaluation metrics and statistical significance evaluation approach opted for in our work, describe the hyper parameter optimization conducted, and discuss seeding and reproducibility.

4.1 Estimating epistemic uncertainty in MuZero

Our method relies on estimating the epistemic uncertainty associated with different predictions used by MuZero for planning: value, reward, state and representation. The reliability of the estimates of the uncertainty in these predictions is expected to have a make-it-or-break-it effect on the method, which makes evaluating even just the soundness of the approach not immediately trivial. To simplify this problem, we identify the least reliable source of epistemic uncertainty to estimate, and remove it from the approximation of the uncertainty propagation. We discuss this decision, as well as motivation and expected consequences, in section 4.1.1. In order to evaluate the method's soundness, we employ a simple, reliable and badly-scalable uncertainty estimation method in the form of a state-visitation count (see section 2.5.2). The approach used to adapt real-states-counting into MuZero to estimate the uncertainty in two independent predictors, as well as extending it for a continuous state-space environment, is described in section 4.1.2. In order to further evaluate the capacity of our method under a more realistic and scalable uncertainty estimation method, we employ ensembles with prior networks (see section 2.5.2). MuZero however uses a categorical representation for the reward and value predictions [2], which introduces additional challenges with estimating a variance in an ensemble of predictions of scalars. Further details regarding the challenges induced as well as engineering efforts overcoming them are described in section 4.1.3.

4.1.1 Planning without explicit transition uncertainty

The method proposed in this work is composed of several moving parts. First, a series of assumptions and approximations to develop computable uncertainty propagation, that may or may not be

sufficiently exact. Second, uncertainty estimation of local quantities, that may or may not be sufficiently reliable. Last, a new approach for using the epistemic uncertainty in planning with MCTS, the consequences of which are yet to be evaluated. A failure in any one of the parts may prevent the methodology from achieving success altogether. In addition, much tuning and engineering is required for every part. As a result, failure of the methodology may be caused by any one of many possible causes. To overcome these challenges and arrive at a methodology that is simpler, and more reliable even without access to a variety of reliable uncertainty estimates, we take two steps. First, we identify the most challenging source of uncertainty, and second, we make the required approximations in order to adapt the methodology to operating without it.

As discussed in section 4.1, in this work we make use of two standard methods to estimate epistemic uncertainty. One relies on state visitations counting, and the other on variance within the output of an ensemble. Both are not immediately trivial to employ to identify transition uncertainty in Muzero. In the original MuZero, the dynamics model is trained only with second-hand loss. Direct loss is only applied to the reward, value and policy functions. As a result, an ensemble of dynamics models cannot be expected to converge to the same representation necessarily, and thus a variance within an ensemble is not a directly usable uncertainty mechanism. Since MuZero uses state-abstractions that do not aim to directly reflect states in the environment, utilizing state-visitation uncertainty is also not immediately available. In addition, the state-abstraction-representation used is a vector, and its approximated propagation through Jacobian matrices as developed in section 3.1 is likely to require the largest tuning efforts. Following these considerations, we believe the state-transition uncertainty $Cov(\boldsymbol{s}_k; a_{1:k-1}, o) = \Sigma_{\boldsymbol{s}_k; a_{1:k-1}, o}$ (see section 3.2.1) is the most challenging and least reliable source of uncertainty.

While none of these challenges are necessarily insurmountable, as will be discussed in section 7.2, due to time and resources constraints we have decided to evaluate our approach under the simplifying approximation that the state-transition uncertainty, the variance associated with $\Sigma_{\boldsymbol{s}_k; a_{1:k-1}, o} = 0$. This choice does not require any additional adaptation from the methodology or the implementation, and can, in principle, directly be replaced back by a mechanism that estimates a nonzero transition uncertainty. The choice to evaluate the method without transition-dynamics-uncertainty has three specific consequences we would like to note explicitly: First, upon success, this evaluation illustrates that even without access to more complex uncertainty mechanisms, that are able to sufficiently-reliably estimate the co-variance between vectoral representations of state, the method is effective, which we believe even further recommends its usability. Second, we believe that this extends the results and the conclusions following from them, and thus the application of the methodology, to DMBRL algorithms that are provided with access to a transition-dynamics model, as well as those that learn it (for example, algorithms such as AlphaZero, that are trained with access to the rules of a game but without access to a model of the reward dynamics necessarily). Third, the methodology for propagating transition-uncertainty, as well is its effects on the method, are not evaluated.

4.1.2 Estimating epistemic uncertainty with state visitation counting in MuZero

As discussed in section 2.5.2, counting of state-action-pairs visitations' can naturally be used as an epistemic uncertainty estimate, that can even be viewed as directly proportional to the variance in an estimator of the mean. We identify three challenges to incorporating this epistemic uncertainty estimator into MuZero: 1) MuZero is planning with *abstracted* states, while the counter is designed

to work against discrete real states of the system. 2) Our method requires two independent estimates for the uncertainty: one in reward, and one in value, while state visitation counting provides a single source for uncertainty estimation. 3) Estimating uncertainty in continuous state-space environments. In this section, we will describe the approach used to overcome each one of those challenges.

Estimating counting uncertainty in planning

In order to estimate the uncertainty associated with *real* states during planning, we allow MuZero access to a real model of the environment. This of course violates the assumption that MuZero is able to learn entirely from interactions with the environment without access to any prior knowledge. This is only introduced in order to evaluate the soundness of our method, and the real model is only used to estimate the uncertainty with planned actions. The second uncertainty estimation method we use, ensembles, does not violate any such assumptions. The epistemic uncertainty in reward-prediction is estimated as follows:

$$u_{r_k} = \beta \frac{1}{n_{s_k} - \epsilon}$$

Where n_{s_k} denotes the count of visitations to *real* state s_k, the state associated with action trajectory $a_{0:k}$ and observation o in a deterministic environment. β is some constant used to scale the uncertainty, and ϵ is a constant used to guarantee numerical stability when $n_{s_k} = 0$. During the MCTS planning phase, in each expansion step, the agent used the real model to predict the transition associated with the chosen action from the chosen state, and uses this prediction to estimate the reward uncertainty.

Creating separate estimates for reward uncertainty and value uncertainty

In order to use state visitation counting as an independent uncertainty estimate for both the value of a *leaf* planning-tree-node k as well as the reward predicted for a transition k, we employ two ideas: 1) we assume that the future reward uncertainty $u_{r_{k+i}}, \forall i > 0$, can be crudely estimated as equal the local reward uncertainty u_{r_k} without completely debilitating the value uncertainty estimation's reliability. 2) We utilize a similar approach to MC simulations to arrive at an approximation of the value uncertainty u_{v_k} that is expected to be better than that provided by 1).

We combine both ideas to arrive at a final computation for the value-uncertainty estimate for leaf-node k. First, the agent plans from real state s_k forward, using the real model, with some action-selection policy π_σ one trajectory h steps into the future. At each step, the agent evaluates the uncertainty of each transition with the state-counter. Second, upon arriving at step $k + h$, the agent uses the geometric-series formula to approximate the uncertainty of following the same policy to infinity, with the approximation that all uncertainties from state s_{k+h} into the future, following policy π_σ, are constant and equal $u_{r_{k+h}}$:

$$u_{v_k} \approx \sum_{i=0}^{h-1} \gamma^{2i} u_{r_{k+i}} + \gamma^{2h} u_{v_{k+h}} \approx \sum_{i=0}^{h-1} \gamma^{2i} u_{r_{h+i}} + \sum_{i=h}^{\infty} \gamma^{2i} u_{r_{k+h}} = \sum_{i=0}^{h-1} \gamma^{2i} u_{r_{k+i}} + \frac{\gamma^{2h}}{1-\gamma^2} u_{r_{k+h}}$$

The first step approximates u_{v_k} as the discounted sum of reward-uncertainties along the trajectory $k : k + h - 1$, and then with an as yet unknown discounted end-of-trajectory value-uncertainty

estimate $u_{v_{k+h}}$. The second step approximates the end-of-trajectory value-uncertainty estimate $u_{v_{k+h}}$ as the sum of a geometric series with the constant reward uncertainty attained at the end of the trajectory, $u_{r_{k+h}}$. The policy π_σ we chose to follow is "repeat action a_{k-1}". For example, if the action leading to planning node k was "accelerate to the right", π_σ chooses "accelerate to the right" for all actions along the trajectory $k : k+h$. This enables u_{v_k} to propagate information from future decisions that (may) be taken by the algorithm, which should provide rather-independent uncertainty estimation from the local reward uncertainty estimates u_{r_k}.

Extending state- visitations counting to continuous state-space environments

As will be discussed in section 4.3, the state space of one of the two environments used to evaluate our method is continuous. In order to employ visitations- counting in a continuous position-velocity state-space environment, we use discretization of the state-space to a 50 by 50 space of possible position-velocity combinations, which is made possible because the ranges of both the velocity as well as the positions are finite.

4.1.3 Estimating epistemic value and reward uncertainty with an ensemble

Estimating the same quantities with the ensemble is done in a much more straight forward manner. The variance in the predictions of the different ensemble members is computed, and is used as the direct measure of the uncertainty in each function - reward and value. As mentioned in the introduction to this section, MuZero predicts the rewards and values using a categorical representation, rather than a simple regression to scalar. The categorical representation can represent numbers in the range $(-support, support)$, for some hyperparameter $support$ that specifies the size of the output layer of the network. The vector-output of the network is passed through a SoftMax function. The weights of the categorical distribution are multiplied by the values represented by the $(-support, support)$. Finally, the entries are summed to produce the final prediction. This architecture introduces an additional challenge to the variance computation - rather than computing the variance over a set of scalars, now one is presented with a set of distributions over which to compute the variance.

As an additional effect of this architecture, we have observed that in under-trained areas of the input space, the networks have tendency to converge to outputs close to 0. We explain this with the claim that for inputs that for the network are arbitrary, the network is likely to produce outputs that are arbitrary. Due to the central limit theorem, we expect arbitrary output for a categorical representation to, on average, not be concentrated in one extreme side of the representation. If the categorical predictions are arbitrary, they are likely to represent values that "sum" to around-0. This suspected phenomenon has two noteworthy effects: 1) the variance in the scalar-representation of the ensemble prediction reduces to zero in under-trained areas of the input space, which is exactly adverse to the behavior we require. 2) This results in an implicit, if unreliable, optimistic or pessimistic initialization of rewards and value predictions (depending on the reward scheme of the environment). Specifically, in environments where the true values are all negative, this may induce an inherent optimistic-initialization effect to the agent's value and reward estimates, implicitly encouraging the agent to explore the unknown.

In order to mitigate these unintended effects, we have taken two steps. First, we have modified the reward schemes of the environments we have tested against, to only produce positive rewards,

and only in the goal state, as detailed in section 4.3, to disable the effect of any unintended optimistic initialization, which may conflict with the method of this work, and give the vanilla version an unintended advantage. Second, rather than compute the straight-forward variance in an ensemble as the variance over the translated-to-scalar predictions, we compute the variance as the variance between the entries of the different categorical representation-entries, entry by entry, and sum them as the final variance measure:

$$Var[y] \approx \sum_{i=0}^{2support+1} Var[y_i]$$

for $y \in [0,1]^{2support+1}$ denoting the categorical-vector output of the NN. An additional variance computation that was considered but had not shown advantage in our preliminary experiments was computing the average categorical distribution of the ensemble, and then taking the average Jensen-Shannon distance [58] between each ensemble-member's categorical distribution, and the mean categorical distribution. While both approaches cannot be expected to be in the correct scale of the real variance of the scalar reward or value predictions, our experiments show that the entry-by-entry variance, at least, is sufficient to achieve both directed as well as deep exploration (section 5.1).

4.2 Agents

To evaluate the increased capacity of our agent to explore, we compare three variations of MuZero: two exploring variations that follow our methodology, each with a different uncertainty-estimation mechanism (exploratory MuZero(s)), to a vanilla version of MuZero. A detailed description of the agents is provided in the following sections.

4.2.1 Exploratory MuZero

The exploratory MuZero agents employ *planning for exploration* to provide deep and directed exploration. The two variations that are evaluated differ based on the uncertainty mechanism used: to evaluate the soundness of the method, state-visitation counting is used. To evaluate it's resilience, reliability and scalability, ensemble-variance is used. The ensemble-based agent uses an ensemble of size 5. The size of the ensemble was chosen as a middle ground between not-incurring a significant computational cost increase as well as expected to be able to provide sufficiently reliable variance estimation in un-explored areas of the state space. The visitation-count based agent employs the mechanism that counts only state-visitations, as an approximation to the more general mechanism that counts state-action visitations. The details describing how exactly the mechanisms are Incorporated are described in the previous section. The agents evaluated employ all adaptations proposed in section 3.5: max value and policy targets, double planning and alternating episodes.

4.2.2 Vanilla MuZero variations

We compare our modified agents to vanilla MuZero, as the baseline. In order to provide a fair comparison to vanilla MuZero, we compare two hyperparameter configurations that induce two random exploration behaviors, and report the agent that achieved the best results. The hyperparameter configurations differ in the temperatures - the first configuration uses high temperatures

(high random action selection in exploration, see section 4.5.1 for more detail), to give MuZero a chance to explore its environment while sacrificing convergence speed. The temperatures were chosen based on a hyperparameter configuration used in the original implementation. The other agent is evaluated with low temperatures, identical to the temperatures used by the exploratory agents, to verify that the reduced temperature parameter does not provide an unexpected, unfair advantage to the exploratory agent. Due to limited compute and a very large number of possible hyperparameters that can be optimized, in this work we did not evaluate a larger set of temperature parameters.

4.2.3 Ablation study

In section 3.5 we describe several adaptations that are expected to enable MuZero to more stably learn from off-policy, exploratory trajectories. In order to evaluate the individual effects of the adaptations proposed we conduct a detailed ablation study. Due to time and compute constraints, we conduct this study only with the state-visitation counting uncertainty mechanism, which is cheaper computationally and thus in cheaper in training time, as well. We believe the results can be expected to extend to other uncertainty mechanisms that are reliable enough to allow the agent to effectively explore.

4.3 Environments

To evaluate the modified agents' capacity for directed deep exploration, we test them against two environments. The first is a toy environment, specifically tailored to be adversarial to random action-selection exploration, requiring exploration that is directed across multiple time steps, which we dub the Slide environment. The second is a standard Gym [59] environment, which is commonly used to evaluate advanced exploration methods in RL algorithms: the Mountain Car environment [21].

4.3.1 The Slide environment

The Slide environment is designed to be an extension to the chain environment [7] that is even more adversarial to agents that do not utilize deep exploration. Slide is inspired by the idea of attempting to climb up a slide in a playground. The state space of the slide environment is a 1-dimensional array of length N. The agent always starts at the bottom of the slide, first entry in the array. The action space consists of 3 actions: down, stay, up. When climbing down, the agent slides back 10 positions (or the bottom of the slide if the agent can't slide further). When attempting to stay in place, the agent slides back one position. Upon climbing up, the agent advances one position forward. The goal state is at the end of the slide, the N-th position of the slide. The goal state is a terminal state. The different reward schemes used with this environment are described in section 4.3.3.

4.3.2 The Mountain Car environment

Mountain Car places the agent at the bottom of a valley. The action space consists of three actions, allowing the agent to accelerate to the right, left, or not at all. The acceleration is constant. The goal state, which is a terminal state, is placed at the top of the mountain to the right. The state

space is continuous, and is comprised of the agent's position along the x axis, as well as the velocity of the car. The environment is calibrated such that the agent cannot arrive at the goal by simply accelerating to the right directly from the initial state. Rather, the agent must first accelerate in one direction, build up momentum according to a simulation of Newtonian gravity, and then accelerate in the other direction to build additional momentum, until it builds up sufficient momentum to drive to the top of the valley, to the goal state. The Mountain Car environment is strongly adversarial to random action selection - there is only one goal, and arriving at it requires executing a very specific sequence of actions. Any "wrong" action can send the agent tumbling down and losing precious momentum. As an additional challenge, the timeout parameter of the environment is tuned such that the agent can try 200 timesteps each episode before timeout, while the fastest way possible to the goal consists of around 90 timesteps, which leaves a very low margin for mistakes. The agent's starting location is chosen randomly inside a small area the bottom of the valley. Among other effects, this induces that the optimal policy has a score that changes per starting position. Any agent that successfully reaches the goal within the 200 timesteps is generally considered successful, and scores that are under 150 timesteps are considered within the range of optimality. Due to the properties described above, requiring execution of a specific and long sequence of actions in order to arrive at the goal during exploration, Mountain Car is considered as an environment that requires deep exploration in order to solve.

4.3.3 Reward schemes

The standard reward scheme of the Mountain Car environment produces a reward of -1 at each time step. The only escape available to the agent is the goal state, which is terminal. The optimal policy induced by this reward scheme is "arrive at the goal in the smallest number of timesteps possible". As mentioned in section 4.1.3, reward schemes that induce negative values are likely to cause unexpected and averse effects for the purpose of evaluating the modifications to the agent. For this reason, we use an additional reward scheme: a *non-Markovian* reward scheme, that extends a zero-one reward scheme.

The zero-one reward scheme is a standard scheme used in RL. Based on this scheme, the agent receives a reward of 0 for every transition, with the exception of the transition into the goal state, which is rewarded with 1 (or in a more general setting, some positive constant). As the discount factor γ is smaller than 1, the agent is expected to learn to arrive at the goal state in the smallest number of timesteps possible. The disadvantage of this scheme is that it makes it less easy to distinguish between agents that achieved the optimal policy, and agents that just manage to arrive at the goal state within the time limit, and any policy in between, when plotting cumulative return. To make evaluating the optimality of the policy of the agent easier, we extend the zero-one reward scheme to a non-Markovian reward scheme. The non-Markovian reward scheme is similar to the zero-one reward scheme. The main difference is that rather than a constant reward at the transition to the goal state, the agent receives as reward $r_{goal} = T_{timeout} - T_{elapsed}$. $T_{timeout}$ denotes the maximum number of timesteps the environment allows for, before sending a timeout signal and terminating the agent. $T_{elapsed}$ is the number of timesteps elapsed in the environment, up until the agent transitioned into the terminal goal state. While this non-Markovian reward induces a non-Markovian environment, the optimal policy remains the same, and so does the learning process of the agent. In addition, this reward scheme allows us to conveniently plot not only whether the agent's policy is successful, but also its optimality. As a result, the reward scheme we chose to use in the main experiments evaluating our methodology, as well as in the ablation testing, is the

non-Markovian reward scheme.

4.4 Evaluation metrics, statistical significance, seeding and reproducibility

We evaluate the agents based on standard RL metrics: learning stability, learning speed, and expected return of final policy. To evaluate the statistical significance of our results, for convenience of plotting, we use the standard error of the mean (SEM) [60]. While standard deviation is a stronger significance metric, the results presented convey the same message with either metric and are, in our opinion, clearer when presented with SEM. In the main results that evaluate our contribution (section 5.1), as well as when evaluating the agents against the original reward scheme (section 5.2), mean and SEM are presented for 10 seeds for each agent. In the ablation study, due to the large number of experiments, we evaluate the agents with 5 seeds per agent. To ease reproducibility, all experiments were seeded with logged random seeds, sampled between 0 and 100000. This range was chosen arbitrarily, expected to be high enough to induce a sufficient variety of seeds within the total number of experiments ran in the process of this work (low hundreds).

4.5 Hyperparameter optimization

The purpose of the main evaluation presented in this work is to illustrate the effect of planning to explore compared to the vanilla version of the algorithm. Further, as the the two versions of the algorithm are not too different from each other, we expect that the majority of hyperparamer optimization will effect all versions similarly. For this reason, no dedicated tuning of hyperparamers was conducted as part of the experiments conducted in this work. The hyperparameters used were chosen based on existing implementations for other environments in the original code base [23]. The network architecture used for mountaincar was based on another implementation of MuZero [61] that was evaluated against the Mountain Car environment.

Two main hyperparameters are exempt from this statement, however. These are: the temperature parameter T and the exploration coefficient c_σ introduced with our proposed methodology for planning for exploration. Motivation and description of the reasoning behind the optimization process and the process itself are provided in the following sections.

4.5.1 Temperatures

The temperature parameter is used by MuZero to decide the weights with which the action selection in the environment is done by weighted random sampling. Specifically, the temperature parameter T is used as follows:

$$a_t \sim p(\cdot), p(a_i) \propto (c_{a_i})^{\frac{1}{T}}$$

a_t is the action to be sampled at time t, and $A = \{a_1, a_2, ..., a_n\}$ is the action space of the environment. c_{a_i} denotes the count of the number of visitations to the child associated with action a_i in the planning tree planned at step t. $p(a_i)$ denotes the probability of sampling action a_i according to the temperature and the visitation counts. When the temperature $T \to 0$, the probability distribution collapses to greedy action selection according the the maximum number of visitations. When

the temperature $T \to \infty$, the distribution becomes uniform. The temperature induces exploration in the environment through random action selection weighted towards "better" actions from an exploitatory perspective, according to the estimates of the tree. As the modifications proposed in this work are meant to provide much more informed exploration, the temperatures used were lower than the original configuration used by other implementations. The original range was $1 \to 0.25$, and the modified range was $0.25 \to 0.1$. Exact hyperparameters are specified in appendix B.2.1.

4.5.2 Tuning the exploration coefficient c_σ

The exploration coefficient c_σ (see section 3.4) was tuned independently for each uncertainty mechanism used, and again per environment. The tuning aimed to achieve preference by the UCB of un-visited states over everything else, and the goal-reward over anything except un-visited states. The tuning was stopped upon observation that deep exploration was achieved successfully in most seeds. The range of c_σ investigated for the state-visitation-counting method was between 0.1 and 100 and was done using rough and then fine grid-search. The range of c_σ investigated for the ensemble-variance method was between 10^1 and 10^8 and was search with a rough and then fine grid search.

Chapter 5

Results

In this section we present the evaluation of our methodology with different uncertainty mechanisms, against two environments with different reward schemes. The two environments, the toy-environment Slide and the more standard for evaluation advanced exploration methods, the Mountain-Car environment, are described in more detail section 4.3: Last, we present an ablation study for the different training targets modifications proposed in section 3.5.

5.1 Evaluation against non-Markovian reward scheme

Figure 5.1 presents the results of planning-to-explore MuZero with either uncertainty estimation mechanism, compared with vanilla MuZero, against both environments, under the non-Markovian reward scheme which only provides positive reward upon reaching the goal state, and zero rewards otherwise. Both versions of planning-to-explore learn reliably, quickly and stably, which demonstrates the agents' capacity to achieve reliable deep and directed exploration. The vanilla MuZero fails to learn to arrive at the goal state in both environments under the learning-steps budget used.

5.2 Evaluation against negative-rewards scheme

Figure 5.2 presents the results against the original Mountain Car reward scheme: rewards of -1 for each time step, only the goal is a true terminal state. The results include planning-to-explore agents with either reward scheme, as well as vanilla agent. We observe that while all agents struggle to learn against the Slide environment (the optimal policy is a consistent score of -60), the planning-to-explore agents are able to learn to arrive at the goal in the Mountain Car environment.

5.3 Ablations study

In this section we present the results of an ablations study conducted on the different modifications proposed in section 3.5 to allow an agent designed to learn with on-policy targets to learn from dedicated exploration trajectories. We present ablations to investigate the individual effects of each target adaptation on the learning process of the algorithm. Specifically, we present the following

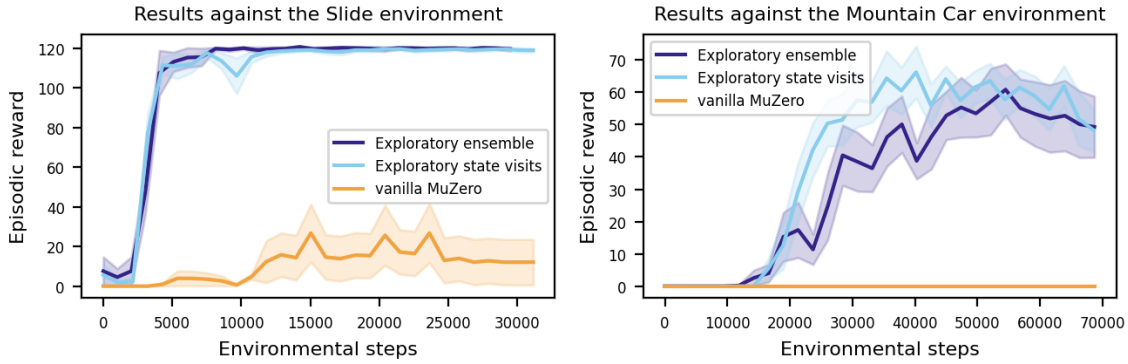

Figure 5.1: Vanilla MuZero vs. planning-to-explore with ensemble-variance uncertainty (exploratory ensemble), vs. planning-to-explore with state-visitation-counting uncertainty (exploratory state visits), against both environments, with the non-Markovian reward scheme. Our method achieves deep and directed exploration with both uncertainty mechanisms.

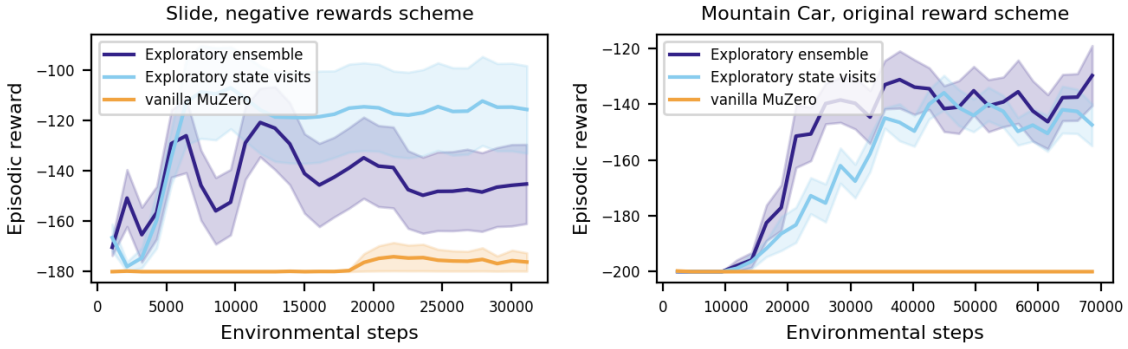

Figure 5.2: Vanilla MuZero vs. planning-to-explore with ensemble-variance uncertainty, vs. planning-to-explore with state-counting uncertainty, against both environments with Mountain Car's original reward scheme: negative reward of -1 at every time step, and only true terminal at the goal. In the Slide environment, only the method using state visitations learns reliably. In Mountain Car however, both versions of our method demonstrate capacity for deep exploration, as well as reliable and stable learning.

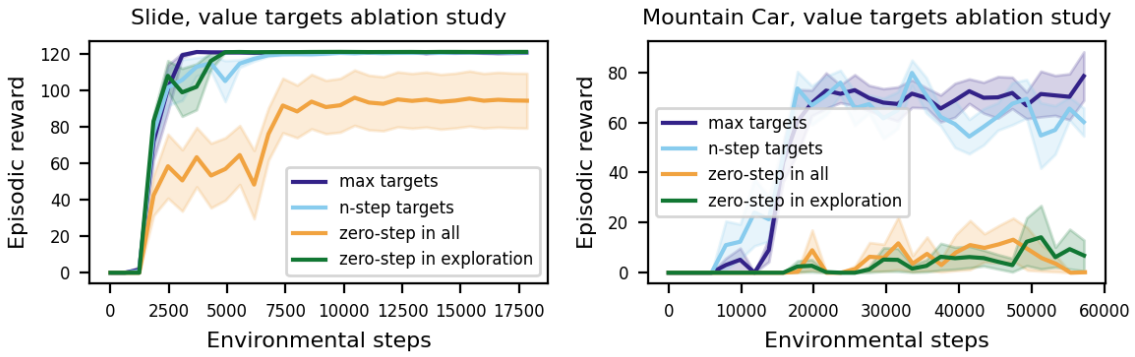

Figure 5.3: Ablations study consisting of different value targets. All agents use max policy targets, alternating episodes and double planning. zero-step targets show significant detrimental effect, while max value targets appear to overcome this effect and achieves the best performance in terms of optimality of final policy and overall stability.

sets of ablations: 1) Value targets ablations, section 5.3.1; Policy targets ablations, section 5.3.2; Double-planning ablations, section 5.3.3; Alternating exploration / reevaluation episodes ablations, section 5.3.4. We conduct each ablation study against both environments. Finally, we discuss the conclusions from the ablation study, in section 5.3.5.

5.3.1 Value targets ablation study

In order to investigate the effect of the different value targets proposed and considered in this work, we compare four ablations. To separate the effect of the value ablations from the effect of the other modifications, we use the following configuration in all value-targets ablation experiments: all agents use alternating episodes, double-planning and max policy targets. The value target ablations investigated are as follows: 1) **zero-step value targets in all types of episodes**, to evaluate the detrimental effect of zero-step targets. 2) **n-step value targets** for all types of episodes, to evaluate the detrimental effect of on-policy learning from off-policy trajectories. 3) **zero-step value targets in exploration** episodes, n-step value targets for exploitation episodes, to evaluate if using zero-step targets only on exploration episodes is sufficient to overcome the general detrimental effect of zero-step targets. 4) **max value targets** for exploration episodes, n-step value targets for exploitation episodes. To evaluate whether choosing on or off policy targets based on their optimality is a sufficient metric to improve the learning of the algorithm, and overcome adverse effects from the presence of zero-step targets. The results are presented in figure 5.3.

In both environments, the zero-step targets in all episodes have a significant detrimental effect. This effect is so severe in the Mountain Car environment, that even when zero-step targets are only used in exploration episodes the agent is unable to learn to arrive at the goal. The effect of using n-step targets from negative-exploratory trajectories (on policy learning from off-policy targets) does not appear significant in the results attained, and appears to influence mostly the stability of the learning. Max value targets appear to have a slight advantage over n-step targets, and mostly in stability. The advantage of max targets over the other two targets in the Mountain Car environment however is substantial.

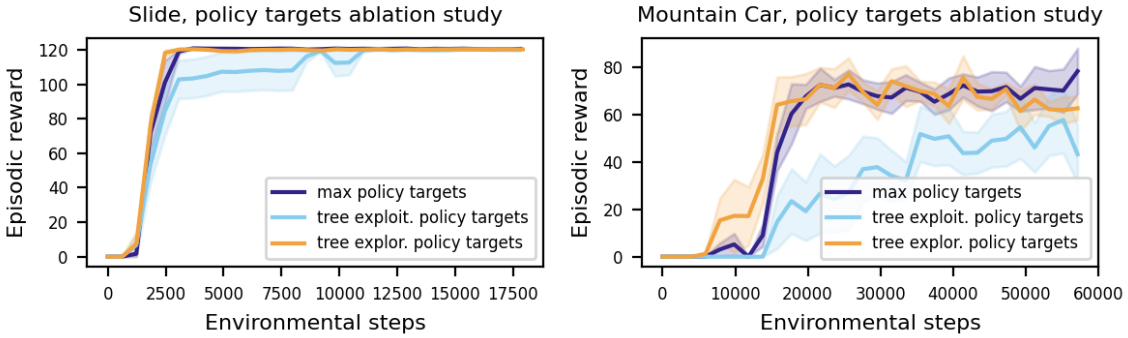

Figure 5.4: Ablations study consisting of different policy targets. All agents use max value targets, alternating episodes and double planning. The effect of using policy targets that follow an exploitatory policy in exploration episodes (tree exploit. policy targets) is the most significant, and the most detrimental. Max policy targets appear to overcome the detrimental effect that the exploitatory policy targets show, as well as induce additional learning stability.

5.3.2 Policy targets ablation study

In order to investigate the effect of the different policy targets proposed and considered in this work, we compare 3 ablations. To separate the effect of the policy ablations from the effect of the other modifications, we use the following configuration in all policy target ablation experiments: all agents use max value targets, alternating episodes and double-planning. The policy target ablations investigated are as follows: 1) **Exploratory-tree policy targets** in exploration episodes, to evaluate the effect of incorporating off-policy policy-targets. In exploitation episodes, the agent always uses the same policy targets, which come from a regular, exploitatory, planning tree. 2) **Exploitatory-tree policy targets** in exploration episodes, to evaluate the effect of policy targets that may conflict with the learned values (see section 3.5.2). 3) **Max policy targets** in exploration episodes, to evaluate whether this middle-ground provides significant gain over the others. The results are presented in figure 5.4.

In the slide environment, all agents appear able to converge to a similarly optimal final policy. Exploitatory policy targets induce significantly slower learning, however. In Mountain Car, the exploratory and max policy targets again achieve similar final policy-optimality, while the exploitatory has a more significant detrimental effect. In both environments however, having access to exploratory trajectories' policy targets (positive, or both) accelerates the learning significantly. We note that under the reward scheme used in this experiment, exploratory policy targets from negative exploration trajectories do not appear to have a significant effect on the learning of the agent. This can be expected, considering that the exploration parameter is tuned such that upon learning to achieve the goal, exploitation should overcome exploration in the decision making of the agent (the modified UCB step in the planning tree).

5.3.3 Double-planning ablation study

In order to investigate the necessity for double-planning, we compare 2 ablations. The double-planning ablations investigated are as follows: 1) **No double planning** max value targets, exploratory policy targets, alternating episodes. 2) **Double planning** max targets for both policy

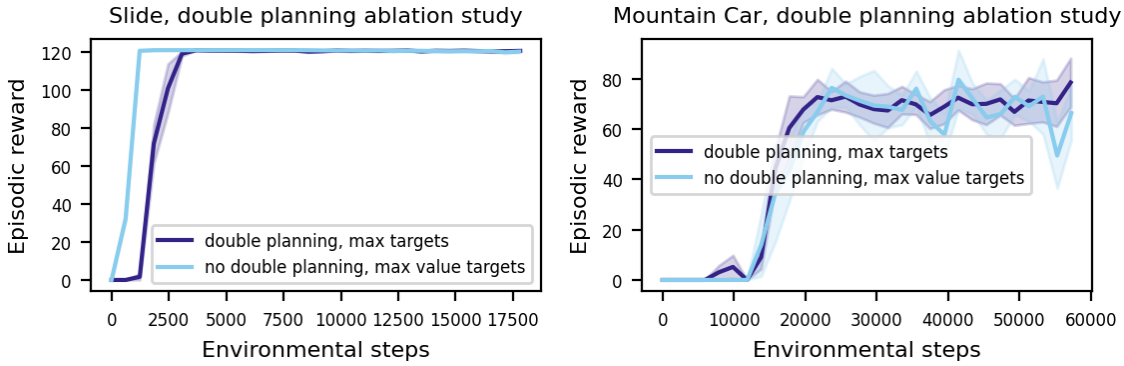

Figure 5.5: Ablations study consisting of configurations of the agents with and without double-planning. All agents use max value targets and alternating episodes. In no double-planning all tree-approximations (such as the policy targets and the value bootstrap) in exploration episodes are from exploratory planning trees. In the slide environment, relying on exploitatory targets from accessible from double-planning appears to do nothing but slow the learning of the agent. In Mountain Car however, double-planning appears to induce stability, and does not show any slowing down of the learning.

and value, alternating episodes. The results are presented in figure 5.5. Against the slide environment, the no-double planning learns faster, if the difference in actual environmental steps is small (about 2000 environmental steps). Against the Mountain Car environment, both methods provide similar learning speeds. The learning stability of the double-planning ablation appears better.

5.3.4 Alternating episodes ablation study

In order to investigate the effect of alternating between exploratory / reevaluation episodes, we consider two different sets of agents, one for each environment. First, over the slide environment, the following agents are compared: 1) **Alternating, max targets** for both value and policy, to evaluate the gain from alternating episodes with the agent that incorporates all ablations. 2) **Non-alternating, max targets** for both value and policy, to compare with the above agent. During learning, a non-alternating agents only performs exploration episodes. 3) **Alternating, zero-step** value targets in exploration episodes, max policy target, to evaluate the expected balancing effect of alternating episodes on the detrimental effect of zero-step targets. 4) **Non-alternating, zero-step** value targets, max policy targets, non-alternating episodes, to compare to the above agent.

In the Mountain Car environment however, in figure 5.3, the severe detrimental effect of zero-step value targets is already explored, both when the agent alternates with reevaluation episodes and uses n-step value targets in them, as well as when the targets in those episodes are zero-step targets. An agent that does not alternate with reevaluation episodes at all, and uses zero-step value targets in all episodes (as with this configuration, all episodes are exploration episodes), can only be expected to perform worse. Thus, instead we compare the following agents, that are expected to provide a more interesting study: 1) **Alternating, max targets** for both value and policy, to evaluate the gain from alternating episodes on the agent that incorporates all ablations. 2) **Non-alternating, max targets** for both value and policy, to compare with the above agent. 3) **Alternating, n-step** value targets in exploration episodes, max policy target, to evaluate the expected balancing effect of alternating episodes on the detrimental effect of n-step targets. 4)

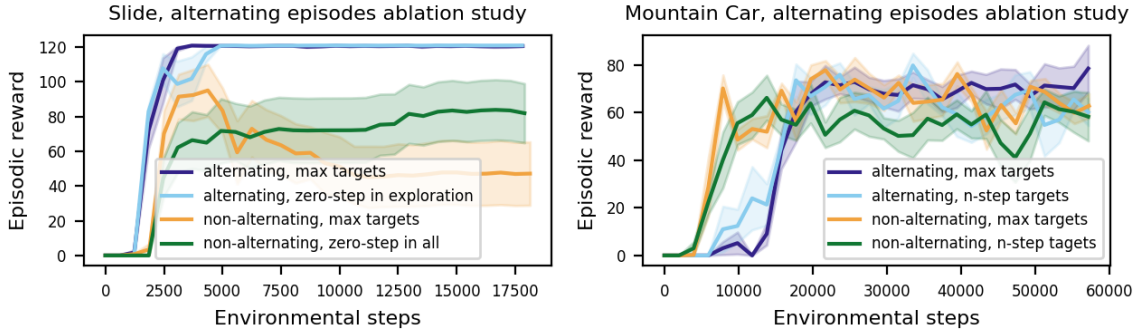

Figure 5.6: Ablations study consisting of different configurations of the agent, with and without alternating between exploration and reevaluation episodes. The improvement compared to agents using zero-step value targets (Slide) is more significant than against those using n-step value targets (Mountain Car). Additionally, in Mountain Car, alternating episodes slows down the learning distinguishably. Still, the agent with alternating episodes as well as max targets appears to achieve the most learning stability as well as most optimal final policy, if with meager statistical significance.

Non-alternating, n-step value targets, max policy targets, non-alternating episodes, to compare to the above agent. The results are presented in figure 5.6.

In the Slide environment, we can see that alternating episodes can be sufficient to overcome the detrimental effect of zero-step targets. This result does not extend to the more complex Mountain Car environment, as illustrated in figure 5.3 presented in the value-ablations section, where both versions of the zero-step targets are unable to learn to arrive at the goal stably. In the Mountain Car environment alternating between exploration and reevaluation episodes results in an identifiable slowdown of the learning of the algorithm (the different time of the learning spike in the first 10000 environmental steps), which is to be expected, as the early reevaluation-episodes are "wasted" as long as the goal state has yet to have been found. When looking at learning stability and stability of final policy however, the dominating agent is the alternating, max-targets agent.

5.3.5 Conclusions from the ablations study

The adaptations for learning from off-policy, dedicated-exploration trajectories introduced demonstrate an overall significantly positive effect on the learning under the conditions investigated. Among the conditions investigated, the effects were the most significant against the more complex Mountain Car environment. It is clear that removing the on-policy n-step value targets and replacing them with zero-step targets is not a sufficient modification in Mountain Car, and further replacing them with max value targets can alleviate the problems arising from the use of zero-step targets, without losing their benefits. While training with value targets that are *exploitatory* is reasonable, training on policy targets that are *exploratory* (i.e. training the agent to follow its exploration policy in exploration episodes) appears preferable to training on policy targets that may be conflicting with the value targets that are learned by the agent from positive exploration trajectories. While the effect of max policy targets does not appear significant, we stipulate that this is due to the environments being of one goal and zero reward in all other states. We would still expect both the max value as well as the max policy targets modifications to provide the same or more stable learning compared to the other targets in most environments, especially environments with

rich reward schemes and multiple possible near-optimal policies. Double planning, which is mostly responsible for providing access to exploitatory-targets, demonstrates an effect on the stability of the final policy. Finally, alternating between exploration and reevaluation episodes can stabilize the learning further, with the price of slowing it down by a distinguishable, if not necessarily significant, margin.

To summarize, based on the results obtained, all the configuration of the different target-adaptation mechanisms we propose recommend themselves when employing dedicated-exploration: *max value targets*, *max policy targets*, *double planning* and *alternating episodes* between exploration and reevaluation.

Chapter 6

Related Work & Discussion

This chapter summarizes related work that tackles similar problems to the ones approached in our work, and discusses expected strengths and possible limitations of our method.

6.1 Related work

In this section we cover several branches of related work that are adjacent or overlap with different aspects of our work. First, we discuss existing approaches to utilizing uncertainty in MBRL (section 6.1.1) including an approach that uses epistemic uncertainty in planning specifically for exploration (section 6.1.2). Next, we summarize existing approaches for utilizing forms of uncertainty in MCTS (section 6.1.3). That is followed by a description of existing approaches for using uncertainty to achieve deep and directed exploration (section 6.1.4). Last, we discuss previous work on on policy training from off-policy trajectories (section 6.1.5).

6.1.1 Model-based reinforcement learning under uncertainty

Probabilistic ensembles with trajectory sampling (PETS) [62] is an approach for uncertainty-aware MBRL. PETS uses the cross entropy method (CEM) [63] for trajectory optimization, by probabilistically sampling trajectories from an ensemble. [64] extends PETS to achieve Bayesian MBRL, by introducing a new framework for model-predictive control (MPC), *variational inference* MPC. This line of work however does not distinguish between aleatoric and epistemic uncertainty, nor is uncertainty utilized specifically to drive exploration. [65] proposes an optimistic MBRL algorithm, dubbed *Hallucinated-UCRL* (H-UCRL), aiming to achieve optimistic exploration with deep-MBRL. H-UCRL relies on Upper-Confidence Reinforcement Learning (UCRL) [66], which optimizes jointly over policies and models inside a set that contains statistically plausible models. H-UCRL combines offline policy optimization with a learned model with online planning with a learned model. The policy optimization step is executed prior to the start of every episode, and aims to provide the agent with a policy that maximizes the *most-optimistic* plausible model, as well as a critic and the optimistic-model itself. Following that step, an episode is executed in the environment. At every step in the episode the agent executes online-planning with respect to the *optimistic* model, learned policy and critic, to induce optimistic exploration in the environment.

Calibrated DMBRL [67] proposes a different approach for using uncertainty estimates in DMBRL. Specifically, Calibrated DMBRL proposes to insert a model-recalibration step into the learning process of the agent, in order to improve the probabilistic model used by the agent for its otherwise unmodified DMBRL approach, with any DL uncertainty quantification method. This work's contribution is essentially orthogonal to the one proposed by our work, with the exception that our work does not require the usage of probabilistic models, but rather, formulates discrete models as probabilistic in order to estimate and utilize epistemic uncertainty. [68] proposes an MBRL approach that is safety aware in systems with uncertainties. The work does not target exploration however, nor are its contributions immediately extendable to other MBRL approaches. [16] is another approach that aims to achieve safe learning through awareness of uncertainty, but does not distinguish between aleatoric and epistemic uncertainty. Another approach to MBRL is to use the model to generate virtual data and train the RL algorithm against it as well as against (or sometimes instead of against) data from the real environment [69].

6.1.2 Planning to explore in DMBRL

Another line of work on DMBRL algorithms that use planning with learned models, parallel to that developed by DeepMind and culminating in MuZero, had been proposed in [52], [70]–[72]. One of the later algorithms proposed in this line of work, Dreamer [71] (and later Dreamer-v2 [73]), are very similar to MuZero in many aspects, in the components of the model they learn and the choice to use it for planning. Two central differences between the two methodologies are that Dreamer is trained with a standard actor-critic approach, and does not employ MCTS in its planning. A recent work in this line has also proposed the idea of planning explicitly for exploration, dubbed *Plan2Explore* [52]. The approach of Plan2Explore, while conceptually and on the surface very similar, differs from our *planning for exploration* approach in several key aspects. First, the only uncertainty that is taken into account by Plan2Explore is from disagreement over *state* representations in the dynamics model, while our method takes additional uncertainties: in value prediction and reward prediction. Second, Plan2Explore does not conduct any explicit forward-propagation of uncertainty, which is encapsulated in the fact that the uncertainties are estimated locally at each imagined state in planning. In other words, the influence of the depth of a planning trajectory is not directly taken into account in the uncertainty computation in Plan2Explore. Our framework however is constructed specifically to evaluate and approximate the propagated uncertainty, both forward (in the approximation of the local uncertainties, sections 3.2.1 to 3.2.3) and back (in the value uncertainty, section 3.2.4), which induces different and potentially more robust uncertainty estimates. Third, Plan2Explore uses the *transition*-uncertainty as imaginary-intrinsic-reward, and propagates the uncertainty back as intrinsic reward, and not explicitly as variance. In comparison, our method propagates the uncertainty explicitly as variance, which induces different back propagation that is more theoretically motivated. Fourth, in Plan2Explore the planning is used explicitly to train an exploration policy, that is used for sample gathering in the environment with which an exploitatory policy will later be trained using offline-RL. Our method does not require training of two policies, and also avoids the unexpected effects associated with learning a policy that is optimized to maximize non-stationary reward-dynamics that are induced by learning intrinsic rewards that are based on decaying novelty of states, by planning to explore *in real time* rather than *learning* to explore. Fifth, the Plan2Explore objective optimizes entirely for novelty of states (or state-transitions). This design choice is made to enable fast adaptation to exploring new tasks with unknown reward schemes. A consequence of this approach is an exploration policy that the extreme

case aims to explore the entire environment's state space, without prioritizing for more promising regions with respect to the return. Exploring the entire state space in environments with large state spaces is, while effective, highly inefficient. In contrast, rather than optimizing for *the most information gain* (most novelty), our objective optimizes for *the most* ***relevant*** *information gain*, by including the value estimate as well as the uncertainty in the planning objective. To summarize, while the underlying idea is very similar between Plan2Explore and our *planning for exploration*, both the approach as well as the details of the methodology differ significantly.

6.1.3 Planning with uncertainty in MCTS

[74] identifies a source of uncertainty that is unique to MCTS, and proposes methodology for utilizing it in order to optimize the trajectory selection process in MCTS. As this approach does not consider environmental epistemic uncertainty, but rather, planning-tree uncertainty, the contribution is orthogonal to the one proposed in our work. *POMCP* [75] is an approach that was proposed in order to be able to perform MCTS in large POMDPs, and even continuous state and action POMDPs. POMCP implicitly introduces uncertainty into MCTS by maintaining a probabilisticly modelled Bayesian belief state over the state of the agent, using a particle filter. This approach was extended in *POMCPOW* [76] and later in *BOMCP* [77]. This line of approach does not explicitly utilize epistemic uncertainty to optimize decision making for exploration, but rather improves the UCT operator used in the trajectory selection step to achieve better exploitatory decisions.

6.1.4 Utilizing epistemic uncertainty for exploration in RL

The work in [78] proposes to use epistemic uncertainty, as well as aleatoric uncertainty, to drive informed exploration, in an on-policy, model-free setting using Thompson sampling [79] to utilize the learned uncertainties for exploratory action selection. As this work proposes a model-free methodology, it does not deal with any explicit propagation of uncertainty, in planning or otherwise. *The uncertainty Bellman-equation* (UBE) [8] presents an analytically-motivated upper bound on the (epistemic) uncertainty associated with Q-values, in the form of its variance. This quantity is then used to drive exploration with Thompson sampling. This work does not involve model-based algorithms however, and thus does not consider any transition-based uncertainty, whether in propagation or otherwise. *Bootstrapped-DQN* [7], later extended with *Bootstrapped-DQN with randomized prior networks* [40] aims to achieve deep and directed exploration by implicit estimation of epistemic uncertainty. Rather than learning one approximator for the agent's Q values, Bootstrapped-DQN proposes to train an ensemble of approximators as a tractable, if crude, approximation for the posterior distribution over possible Q-networks. Bootstrapped-DQN than proposes to epsilon-greedily follow each estimators' Q-value estimations during learning, relying on variety in the ensemble's estimations on un-known states to drive exploration into states that are unvisited, and enjoy a high initial estimate with at least one of the ensemble members. Randomized prior networks were later introduced into this methodology, to increase variety in ensemble predictions in untrained areas of the state space.

The idea of intrinsic-motivation for exploration [32] has been proposed in RL in the form of *intrinsic reward* [46]. When employing intrinsic reward, the agent learns a modified value function that takes into account as a reward bonus a quantity that relates to the (epistemic) uncertainty with this reward, or the transition associated with this reward. In other words, the original value

function:

$$V^{\pi}(s_t) = E[r(s_t, a) + \gamma V^{\pi}(s_{t+1}) | a \sim \pi(s_t), s_{t+1} \sim P(\cdot|s_t, a)]$$

Is modified with an uncertainty-related bonus:

$$V^{\pi}_{intrinsic}(s_t) = E[r(s_t, a) + u(s_t, a) + \gamma V^{\pi}(s_{t+1}) | a \sim \pi(s_t), s_{t+1} \sim P(\cdot|s_t, a)]$$

Where $u(s_t, a)$ denotes some bonus that decays as the epistemic certainty grows. Common uncertainty bonuses are computed based on visitation counts [80], pseudo-counts [46] and others [48]. The intrinsic reward methodology allows for very effective, far-horizon uncertainty propagation. Because the information about the uncertainty is immediately inserted into the value approximation, it propagates in every value-learning step, and only decays with time and training. The introduction of intrinsic rewards into the value learning carries with it unintended consequences, such as teaching the agent to explore rather than execute, at least in the earlier part of the training, or requiring other algorithmic acrobatics in order to decouple the effect of learning the epistemic uncertainty in the value function, with learning an optimally-exploitatory value function.

An inventive exploration approach dubbed *go-explore* [10] is vying for the title of the most farsighted exploration of them all. Go-explore saves attractive-to-explore states explicitly, and executes exploration by reaching them deterministically, and then exploring from the probabilistically. While this method violates many assumptions that have been standard in RL so far, it is able to achieve far above previous state of the art performance in many of the most challenging exploration domains that RL has faced. One of the metrics go-explore uses to evaluate the attractiveness of states for exploration is epistemic uncertainty.

6.1.5 On-policy training with experiences from off-policy trajectories

A recent work by [57] attempted to achieve greater sample efficiency by using information from simulations done inside the MCTS planning tree in MuZero, as a source of additional virtual experiences to train the algorithm, building on work done in [81]. The method proposed for generating targets from these virtual experiences relies on a traversal of the MCTS planning tree that does not necessarily follow the order in which the tree was created, which induces targets that are off-policy. In order to incorporate these off-policy targets to MuZero's training, the method proposed is to modify the loss function by adding auxiliary value and policy losses from the simulated games. This modification is not applicable to the challenge our method is required to overcome, of introducing none-exploitatory trajectories into on-policy training.

6.2 Discussion

Advanced exploration can be expected to achieve the following goals: 1) **Directed and informed** The method should be able to choose an action for exploration based on decision making that transcends random action selection. 2) **Directed over multiple time-steps** If the exploration is only able to choose one action every so often, the agent will be unable to direct itself towards areas that appear attractive to explore, but are more than one local decision away. Therefor, the method should be capable of executing a trajectory of consecutive exploratory actions. 3) **Farsighted** It is very plausible that areas that are attractive to explore are some decisions away from the agent's current position (in the state-action transition space), and local information will not be sufficient

to inform the agent of the existence of these areas. In other words, information that is relevant for exploration is required to *propagate* from the areas that are attractive to explore, all the way back to the agent's current position. Our method is one of a small set of advanced-exploration methods that expect to achieve all three goals, to some degree. In the following sections, we investigate the extent to which our method achieves those targets.

6.2.1 Directed and informed

Our method combines epistemic uncertainty from different sources to choose decisions for exploration by planning in real time. In that regard, the exploration provided by our method is both directed (action selection is not arbitrarily-random) as well as informed (action selection is based on epistemic uncertainty).

6.2.2 Directed over multiple time-steps

There is a large design space for deciding when to start exploring, for how long a period, and when to stop, as investigated in recent work [82]. In the results presented, the two dominating candidates for exploration periods were exploration that is interleaved in-episode for multiple time steps at a time (intra-episodic), and exploration that lasts for an entire episode, every so-many episodes (episodic). This is well in line with the assumption that in many environment, exploring one step at a time is insufficient, and the exploration must be able to direct the agent over multiple time steps towards *far* attractive areas of the state space. In our work, for simplicity of evaluation, we chose to employ our method with episodic exploration, rather than intra-episodic exploration. Regardless, both approaches are naturally usable with our method, and both approaches achieve directness over multiple time steps, which is the target discussed in this section.

6.2.3 Farsighted

Possibly the most challenging goal to achieve out of the three, reliable farsightedness requires the agent to effectively propagate possible-future information back over many steps into the agent's current state. Perhaps the most farsighted among the standard advanced-exploration methods is the standard variation of *intrinsic reward*, which inserts local uncertainty information into the agent's value learning. This choice to propagate the uncertainty through the value function induces that over time, the uncertainty can potentially be propagated over the entire state space. This method, while able to propagate uncertainty through the entire learned-space of the value function given time, nonetheless suffers from several drawbacks arising from the fact that the uncertainty propagates through the values.

Slow propagation The information cannot propagate faster than the values themselves, which depending on the value-targets used, can be as slow as one step into the future at a time when the value targets are the standard $v_t^{target} = r_t + \gamma v(s_{t+1})$. This can cause the method to require many training steps to propagate the information sufficiently far.

Unreliable estimates in under-trained areas Intrinsic reward cannot expect to reliably propagate information through values of states that are not trained on sufficiently often, for two reasons. First, the information potentially has yet to propagate far enough, following the argument above. Second, propagating the uncertainty, which is expected to decay in time with more interactions, induces non-stationarity into the value function which may further reduce the reliability of under-trained value estimates.

On/off-policy challenges Propagating the uncertainty through the value function implicitly trains an explicitly-exploratory policy. Since in RL we mostly wish to train exploitatory-policies, the standard solution involves introducing an additional value function, that is trained side by side but without the uncertainty. This creates the same set of challenges our method faces when incorporated into on policy algorithms that struggle to learn from trajectories that have been executed following an exploratory policy.

Learning non-stationary values The values induced by intrinsic reward that decays with novelty are by definition non-stationary. While the exact effects of learning non-stationary values are not trivial to evaluate, it is generally considered to be a source for additional behavioral artifacts that may cause unexplained and unwanted behavior in the learning and acting of the agent.

In comparison, our method is able to elegantly propagate epistemic uncertainty from future states in real time without any of the effects introduced by learning the uncertainty through the value function, by taking advantage of the planning that is already used by the agent. The choice to propagate the information through the planning also induces the main limitation of our method - farsightedness that is potentially limited in the depth of the plannig tree, which is at best linear $O(n)$ in the planning budget n, and at worst logarithmic $O(\log(n))$. If the MCTS tree plans in the same direction for an entire planning phase, the planning tree will take the form of a chain of length n. This behavior is generally unwanted due to nonexistent exploration in the tree, and is not generally the behavior expected when employing MCTS. On the other extreme, if the planning tree plans evenly in every direction, the number of nodes n in such a planning tree of height h with action space with cardinality $|A|$ is $n \leq |A|^{h+1} - 1$, and therefor $h \leq \log_{|A|}(n+1) - 1$. Thus, in a general worst case, our method is able to propagate information only as far as $O(\log(n))$. The negative effect of this property is expected to decrease as the reliability of the uncertainty in the values of leaves $Var[\boldsymbol{v}_k^{leaf}]$ increases - the more far-seeing and thus reliable the uncertainty estimate already is, the lesser the harm from not planning further. Increasing the reliability of $Var[\boldsymbol{v}_k^{leaf}]$ can be achieved by combining our method with UBE [8] which proposes a method for learning upper bounds on the uncertainty in values.

Chapter 7

Conclusions & Future Work

In this chapter we discuss our conclusions, as well as possible directions for future work that arise from our work.

7.1 Conclusions

Using the framework we propose for modelling epistemic uncertainty in planning with a deterministic model, along with the analytic motivation of the law of total variance, we answer the first research question investigated in this work and develop methodology for approximating uncertainty as it propagates in planning. To answer the second research question, we develop a new exploration method, *planning for exploration*, that uses this framework to achieve deep exploration. The results demonstrate the efficient and effective deep exploration achieved by the method, expressing the gain from incorporating epistemic uncertainty into planning. These results are maintained even in the presence of less-reliable uncertainty estimation mechanisms, such as the variance within an ensemble, illustrating the resilience of the method to unreliability in uncertainty estimation, which is currently proliferate among uncertainty estimation methods. To answer the last research question, we introduce modular approaches for generating on-policy targets from off-policy exploratory trajectories. The ablation study we conduct demonstrates the capacity of these approaches to enable stable on-policy learning from exploratory trajectories. These approaches introduce an exciting opportunity to incorporate advanced exploration into other on-policy algorithms.

We believe that the effective exploration achieved by incorporating epistemic uncertainty into planning suggests that other approaches for more informed decisions making through planning with epistemic uncertainty can be developed. The framework we propose for incorporating the uncertainty provides a simple and convenient foundation that can be used for developing such approaches. We explore part of this space in the following section.

7.2 Future work

We suggest several directions for future work, that naturally arise out of our work:

7.2.1 Evaluation of state-abstraction propagation

Extending this work by introducing mechanisms to estimate the state-abstraction uncertainty $\Sigma_{\boldsymbol{s}_k;a_{1:k-1},o}$, and evaluating their effect on the method through an ablation study, is perhaps the most immediate of all opportunities for future work. We propose a simple approach for estimating the state-abstraction uncertainty in the form of the same ensemble variance already used in this work. In vanilla MuZero, there is no explicit loss for the state abstraction, which introduces a challenge for using ensembles for this purpose. However, [83] proposes an extension to MuZero that introduces an explicit state-abstraction loss that can be used to train such an ensemble.

7.2.2 Reliable planning

Our framework for epistemic uncertainty propagation immediately suggests an additional opportunity for planning-reliability improvements. Specifically, in each step in the planning tree, the total uncertainty associated with a certain estimate can *drop*, if the uncertainty after expansion is smaller than or equal to the uncertainty before expansion:

$$Var(\boldsymbol{v}_{k+h};a_{0:k+h},o) < Var(\boldsymbol{v}_{k+h};a_{0:k+h-1},o) \tag{7.1}$$

Or *rise*, otherwise (or in a limiting case stay the same). If the epistemic uncertainty grows, the framework suggest that planning in this direction actually *reduces* the reliability of the new estimate, compared to the old estimate. As the purpose of planning is to *increase* the reliability of estimates, this immediately suggests that perhaps the tree should cease planning in this direction, both to optimize its planning-resources allocation, as well as increase the quality of the estimates that result from the planning.

7.2.3 Planning for reliable exploitation

Our method *planning for exploration* immediately suggests a method for *planning for reliable exploitation*. Rather than modifying the UCB to incentivize for planning in the direction of high uncertainty, we can *disincentivize* planning in the direction of high uncertainty. In settings such as offline-RL, where executing in untrained areas of the action space can lead to arbitrarily bad performance, such planning can be expected to provide significant reliability to the final exploitation policy.

7.2.4 Planning for exploration and exploitation in the same tree

An additional venue for future research related to planning resource optimization that we believe would be attractive to pursue is in combining the double-planning effort into a single planning tree, thus reducing the planning resources required by exploration episodes to the same planning resources required by regular episodes. We believe this can be achieved by alternating within the same planning tree between expansion-for-exploration and expansion-for-exploitation, and utilizing the fact that the tree now employs two separate search heuristics as additional exploration within the tree. Further separating the visitation counts to *exploratory*-purposed visitations and *exploitatory*-purposed visitations can allow to retain the same action-selection in the environment employed by MuZero, without any further modifications.

7.2.5 Stabilizing through targeted sampling from the replay buffer instead of sampling in the environment

In order to stably learn reliable exploitation policies despite acting exploratorily in the environment, we proposed to alternate between exploration and practice episodes in the environment. As illustrated in the ablation testing, while this can indeed increase learning stability it induces a slightly lower sample efficiency with many "wasted" practice episodes. To further increase sample efficiency, we believe the same problem can be overcome by balanced sampling of practice and exploratory trajectories from the replay buffer, instead of balanced acting in the environment. In fact, the already parrallelized framework used by MuZero can be extended to use samples from MuZero's existing dedicated evaluation thread to keep a constantly updating practice trajectory in the replay buffer, and alternating the sampling from the replay buffer between exploration and practice trajectories, to achieve higher sample (and computational) efficiency as well as stable learning. If the sanctity of evaluation is wished to be kept, a new dedicated exploitation thread can be added rather than using samples from the evaluation episodes.

Acknowledgements

This endeavor would not have been possible without the pertinent and unstinting support of my immediate supervisor, Dr. Wendelin Böhmer. I would like to express my deepest appreciation to Dr. Marco Loog, for consistent availability for creative discussions. In addition, many thanks go to Dr. Matthijs Spaan, for his persistent support. I would like to express my deepest gratitude to Dr. Frank van der Meulen, for his crucial assistance with developing the theoretical foundation on which this work stands. Special thanks to the PhD students of the algorithmics group of Delft University of Technology, Moritz Zanger, Pascal van der Vaart & Joery de Vries for many fruitful discussion as well as excellent feedback. I'm indebted to Itamar Sher, for providing key insights that streamlined our ability to overcome major challenges encountered in the process of this work. Additional thanks goes to Delft University of Technology for granting us access to the INSY cluster, which was crucial to evaluate the contributions proposed in this work. Last, I would like to thank my partner, Irene Leibbrand, for enthusiastically surviving theoretical discussions about things which are far and away out of her realm of interest, as well as me, in the process of this work.

Bibliography

[1] A. Mandhane, A. Zhernov, M. Rauh, C. Gu, M. Wang, F. Xue, W. Shang, D. Pang, R. Claus, C.-H. Chiang *et al.*, "Muzero with self-competition for rate control in vp9 video compression," *arXiv e-prints*, pp. arXiv–2202, 2022.

[2] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[3] Y. Wang, S. Inguva, and B. Adsumilli, "Youtube ugc dataset for video compression research," in *2019 IEEE 21st International Workshop on Multimedia Signal Processing (MMSP)*. IEEE, 2019, pp. 1–5.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[6] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[7] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," *Advances in neural information processing systems*, vol. 29, pp. 4026–4034, 2016.

[8] B. O'Donoghue, I. Osband, R. Munos, and V. Mnih, "The uncertainty bellman equation and exploration," in *International Conference on Machine Learning*, 2018, pp. 3836–3845.

[9] Y. Oren, R. A. Starre, and F. A. Oliehoek, "Comparing exploration approaches in deep reinforcement learning for traffic light control," *BNAIC/BeneLearn 2020*, p. 179, 2020.

[10] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "Go-explore: a new approach for hard-exploration problems," *arXiv preprint arXiv:1901.10995*, 2019.

[11] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods," *Machine Learning*, vol. 110, no. 3, pp. 457–506, 2021.

[12] R. L. Russell and C. Reale, "Multivariate uncertainty in deep learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[13] T. Pearce, F. Leibfried, and A. Brintrup, "Uncertainty in neural networks: Approximately bayesian ensembling," in *International conference on artificial intelligence and statistics*. PMLR, 2020, pp. 234–244.

[14] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

[15] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *International Conference on Machine Learning*. PMLR, 2015, pp. 1613–1622.

[16] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *arXiv preprint arXiv:1702.01182*, 2017.

[17] B. Lütjens, M. Everett, and J. P. How, "Safe reinforcement learning with model uncertainty estimates," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8662–8668.

[18] Y. Xue, S. Cheng, Y. Li, and L. Tian, "Reliable deep-learning-based phase imaging with uncertainty quantification," *Optica*, vol. 6, no. 5, pp. 618–629, 2019.

[19] J. Heo, H. B. Lee, S. Kim, J. Lee, K. J. Kim, E. Yang, and S. J. Hwang, "Uncertainty-aware attention for reliable interpretation and prediction," *Advances in neural information processing systems*, vol. 31, 2018.

[20] Y. Jin, Z. Yang, and Z. Wang, "Is pessimism provably efficient for offline rl?" in *International Conference on Machine Learning*. PMLR, 2021, pp. 5084–5096.

[21] A. W. Moore, "Efficient memory-based learning for robot control," University of Cambridge, Tech. Rep., 1990.

[22] Y. Oren, "muzero-general," *GitHub repository*, 2022, https://github.com/YanivO1123/muzero-general.

[23] W. Duvaud, "muzero-general," *GitHub repository*, 2021, https://github.com/werner-duvaud/muzero-general/tree/23a1f6910e97d78475ccd29576cdd107c5afefd2.

[24] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[25] K. J. Åström, "Optimal control of markov processes with incomplete state information i," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 174–205, 1965.

[26] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[27] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[29] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[30] T. M. Moerland, J. Broekens, and C. M. Jonker, "Model-based reinforcement learning: A survey," *arXiv preprint arXiv:2006.16712*, 2020.

[31] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[32] P.-Y. Oudeyer and F. Kaplan, "What is intrinsic motivation? a typology of computational approaches," *Frontiers in neurorobotics*, vol. 1, p. 6, 2009.

[33] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2, pp. 235–256, 2002.

[34] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*. Springer, 2006, pp. 282–293.

[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[36] T. M. Mitchell, "Version spaces: A candidate elimination approach to rule learning," in *Proceedings of the 5th international joint conference on Artificial intelligence-Volume 1*, 1977, pp. 305–310.

[37] G. Shafer, "A mathematical theory of evidence," in *A mathematical theory of evidence*. Princeton university press, 1976.

[38] N. Meinert and A. Lavin, "Multivariate deep evidential regression," *arXiv preprint arXiv:2104.06135*, 2021.

[39] V. Bengs, E. Hüllermeier, and W. Waegeman, "On the difficulty of epistemic uncertainty quantification in machine learning: The case of direct uncertainty estimation through loss minimisation," *arXiv preprint arXiv:2203.06102*, 2022.

[40] I. Osband, J. Aslanides, and A. Cassirer, "Randomized prior functions for deep reinforcement learning," *arXiv preprint arXiv:1806.03335*, 2018.

[41] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *arXiv preprint arXiv:1612.01474*, 2016.

[42] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," *Advances in neural information processing systems*, vol. 28, pp. 2575–2583, 2015.

[43] M. Teye, H. Azizpour, and K. Smith, "Bayesian uncertainty estimation for batch normalized deep networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4907–4916.

[44] N. Pawlowski, A. Brock, M. C. Lee, M. Rajchl, and B. Glocker, "Implicit weight uncertainty in neural networks," *arXiv preprint arXiv:1711.01297*, 2017.

[45] C. Louizos and M. Welling, "Multiplicative normalizing flows for variational bayesian neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2218–2227.

[46] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," *Advances in neural information processing systems*, vol. 29, 2016.

[47] G. Ostrovski, M. G. Bellemare, A. Oord, and R. Munos, "Count-based exploration with neural density models," in *International conference on machine learning*. PMLR, 2017, pp. 2721–2730.

[48] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[49] N. Tagasovska and D. Lopez-Paz, "Single-model uncertainties for deep learning," *Advances in Neural Information Processing Systems*, vol. 32, pp. 6417–6428, 2019.

[50] M. Sensoy, L. Kaplan, and M. Kandemir, "Evidential deep learning to quantify classification uncertainty," *arXiv preprint arXiv:1806.01768*, 2018.

[51] A. Amini, W. Schwarting, A. Soleimany, and D. Rus, "Deep evidential regression," *arXiv preprint arXiv:1910.02600*, 2019.

[52] R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak, "Planning to explore via self-supervised world models," in *International Conference on Machine Learning*. PMLR, 2020, pp. 8583–8592.

[53] A. Spanos, *Probability theory and statistical inference: Empirical modeling with observational data*. Cambridge University Press, 2019.

[54] N. A. Weiss, "A course in probability. 2005," 2005.

[55] M. Capiński and T. Zastawniak, *Conditional Expectation*. New York, NY: Springer New York, 2001, pp. 183–212. [Online]. Available: https://doi.org/10.1007/978-0-387-21659-1_11

[56] B. Ochoa and S. Belongie, "Covariance propagation for guided matching," in *Proceedings of the Workshop on Statistical Methods in Multi-Image and Video Processing (SMVP)*, vol. 83, 2006.

[57] A. Borges and A. Oliveira, "Combining off and on-policy training in model-based reinforcement learning," *arXiv preprint arXiv:2102.12194*, 2021.

[58] F. Nielsen, "On the jensen-shannon symmetrization of distances relying on abstract means," *Entropy*, vol. 21, no. 5, 2019. [Online]. Available: https://www.mdpi.com/1099-4300/21/5/485

[59] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[60] D. G. Altman and J. M. Bland, "Standard deviations and standard errors," *Bmj*, vol. 331, no. 7521, p. 903, 2005.

[61] J. de Vries, K. Voskuil, and F. Bryce, "muzero," *GitHub repository*, 2021, https://github.com/kaesve/muzero/tree/ee990aa4927a6ec34eef6928da50446b40f3685f.

[62] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *Advances in neural information processing systems*, vol. 31, 2018.

[63] Z. I. Botev, D. P. Kroese, R. Y. Rubinstein, and P. L'Ecuyer, "The cross-entropy method for optimization," in *Handbook of statistics*. Elsevier, 2013, vol. 31, pp. 35–59.

[64] M. Okada and T. Taniguchi, "Variational inference mpc for bayesian model-based reinforcement learning," in *Conference on Robot Learning*. PMLR, 2020, pp. 258–272.

[65] S. Curi, F. Berkenkamp, and A. Krause, "Efficient model-based reinforcement learning through optimistic policy search and planning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 14 156–14 170, 2020.

[66] P. Auer, T. Jaksch, and R. Ortner, "Near-optimal regret bounds for reinforcement learning," *Advances in neural information processing systems*, vol. 21, 2008.

[67] A. Malik, V. Kuleshov, J. Song, D. Nemer, H. Seymour, and S. Ermon, "Calibrated model-based deep reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4314–4323.

[68] S. N. Mahmud, K. Hareland, S. A. Nivison, Z. I. Bell, and R. Kamalapurkar, "A safety aware model-based reinforcement learning framework for systems with uncertainties," in *2021 American Control Conference (ACC)*. IEEE, 2021, pp. 1979–1984.

[69] R. S. Sutton, C. Szepesvári, A. Geramifard, and M. P. Bowling, "Dyna-style planning with linear function approximation and prioritized sweeping," *arXiv preprint arXiv:1206.3285*, 2012.

[70] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *International conference on machine learning*. PMLR, 2019, pp. 2555–2565.

[71] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," *arXiv preprint arXiv:1912.01603*, 2019.

[72] D. Hafner, K.-H. Lee, I. Fischer, and P. Abbeel, "Deep hierarchical planning from pixels," *arXiv preprint arXiv:2206.04114*, 2022.

[73] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, "Mastering atari with discrete world models," *arXiv preprint arXiv:2010.02193*, 2020.

[74] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, "The second type of uncertainty in monte carlo tree search," *arXiv preprint arXiv:2005.09645*, 2020.

[75] D. Silver and J. Veness, "Monte-carlo planning in large pomdps," *Advances in neural information processing systems*, vol. 23, 2010.

[76] Z. N. Sunberg and M. J. Kochenderfer, "Online algorithms for pomdps with continuous state, action, and observation spaces," in *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 2018.

[77] J. Mern, A. Yildiz, Z. Sunberg, T. Mukerji, and M. J. Kochenderfer, "Bayesian optimized monte carlo planning," *arXiv preprint arXiv:2010.03597*, 2020.

[78] T. M. Moerland, J. Broekens, and C. M. Jonker, "Efficient exploration with double uncertain value networks," *arXiv preprint arXiv:1711.10789*, 2017.

[79] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3-4, pp. 285–294, 1933.

[80] T. Rashid, B. Peng, W. Böhmer, and S. Whiteson, "Optimistic exploration even with a pessimistic initialisation," *International Conference on Learning Representations (ICLR)*, 2020.

[81] D. Willemsen, H. Baier, and M. Kaisers, "Value targets in off-policy alphazero: a new greedy backup," *Neural Computing and Applications*, vol. 34, no. 3, pp. 1801–1814, 2022.

[82] M. Pislar, D. Szepesvari, G. Ostrovski, D. Borsa, and T. Schaul, "When should agents explore?" *arXiv preprint arXiv:2108.11811*, 2021.

[83] J. Scholz, C. Weber, M. B. Hafez, and S. Wermter, "Improving model-based reinforcement learning with internal state representations through self-supervision," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[84] A. Guessab, O. Nouisser, and G. Schmeisser, "Multivariate approximation by a combination of modified taylor polynomials," *Journal of Computational and Applied Mathematics*, vol. 196, no. 1, pp. 162–179, 2006.

[85] J. Hansen, "bootstrap_dqn," *GitHub repository*, 2019, https://github.com/johannah/bootstrap_dqn/tree/734fd78a776cc940da40d5ce5f3775b5ce7cac7d.

[86] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60.

Appendix A

Covariance Approximation

In this section we will elaborate on the motivation for the approximation of the covariance of a nonlinear multivariate function f with respect to a random variable $\boldsymbol{s}_{k-1}; a_{0:k-2}, o$ used in section 3.2.1:

$$Cov\Big(f\big((\boldsymbol{s}_{k-1}; a_{0:k-2}, o), a_{k-1}\big)\Big) \approx J_s \Sigma_{\boldsymbol{s}_{k-1}; a_{0:k-2}, o} J_s^T$$

To shorten notation, we will drop the dependence of $\boldsymbol{s}_{k-1}; a_{0:k-2}, o$ on $a_{0:k-2}, o$ from the notation in the rest of this formulation, and use only $\boldsymbol{s}_{k-1}$ instead. Additionally, we will change notation to $Cov(f(\boldsymbol{s}_{k-1}, a_{k-1})) := Cov(f_{a_{k-1}}(\boldsymbol{s}_{k-1}))$, to emphasize that from the perspective of the covariance, the constant a_{k-1} need not be considered an variable input of the function f. The first order Taylor approximation of a general function $g(x)$, multivariate in input as well as output, can be written as $g(x) \approx g(x_0) + J_{g(x_0)}(x - x_0)$ [84], where $J_{g(x_0)}$ denotes the Jacobian matrix of g at point x_0. The Jacobian matrix $J_{g(x)}$ is the matrix of all the first derivatives of the multivariate function g with outputs $[g_1 \dots g_m]^T$, with respect to every variable $x_i \in [x_1, \dots, x_n]^T = x$ organized as follows:

$$J_{g(x)} := [\frac{\partial g}{\partial x_1} \dots \frac{\partial g}{\partial x_2}] = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \dots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_m}{\partial x_1} & \dots & \frac{\partial g_m}{\partial x_n} \end{bmatrix}$$

Substituting g for f and x_0 for $\mu_{s_{k-1}}$ a mean-estimate of $\boldsymbol{s}_{k-1}$ and using the first order Taylor approximation:

$$Cov\Big(f_{a_{k-1}}(\boldsymbol{s}_{k-1})\Big) \approx Cov\Big(f_{a_{k-1}}(\mu_{s_{k-1}}) + J_{f_{a_{k-1}}(\mu_{s_{k-1}})}(\boldsymbol{s}_{k-1} - \mu_{s_{k-1}})\Big)$$

Where the dimensionality $m \times n$ of the Jacobian matrix $J_{f_{a_{k-1}}(\mu_{s_{k-1}})}$ is the length of the input state vector $\mu_{s_{k-1}}$, times the output state-estimate of $f(\mu_{s_{k-1}}) = s_k$, which is of the same dimensionality. In other words, the matrix $J_{f_{a_{k-1}}(\mu_{s_{k-1}})}$ is square and its number of rows and columns is the same as the dimensionality of the state estimates s_k. We will proceed to simplify the notation with $J_{f_{a_{k-1}}(s_{k-1})} := J_s$. Using the property that $Cov(b + A\boldsymbol{x}) = ACov(\boldsymbol{x})A^T$, for a constant vector b, a

constant matrix A and a random vector $\boldsymbol{x}$:

$$Cov\Big(f_{a_{k-1}}(\boldsymbol{s}_{k-1})\Big) \approx Cov\Big(f_{a_{k-1}}(\mu_{s_{k-1}}) + J_s(\boldsymbol{s}_{k-1} - \mu_{s_{k-1}})\Big)$$

$$= Cov\big(J_s\boldsymbol{s}_{k-1} - J_s\mu_{s_{k-1}}\big) = J_s Cov(\boldsymbol{s}_{k-1})J_s^T$$

Finally, using the notation $\Sigma_{\boldsymbol{s}_{k-1};a_{0:k-2},o} := Cov(\boldsymbol{s}_{k-1}; a_{0:k-2}, o) := Cov\big(\boldsymbol{s}_{k-1}\big)$ for the covariance of $\boldsymbol{s}_{k-1}; a_{0:k-2}, o$ as used in section 3.2.1, we can arrive at the final approximation that is used in the section:

$$Cov\Big(f\big((\boldsymbol{s}_{k-1}; a_{0:k-2}, o), a_{k-1}\big)\Big) \approx J_s\Sigma_{\boldsymbol{s}_{k-1};a_{0:k-2},o}J_s^T$$

Appendix B

Implementation

The implementation used to evaluate the agent is accessible in [22]. This implementation was built on the implementation by [23], which in turn is built on the official psuedocode released in the original MuZero paper [2]. The implementation of the ensemble architecture was based on [85], which is an implementation of the bootstrapped-DQN with randomize prior networks proposed in [40]. In the following sections, we specify first the details of the network architecture used, and second the hyperparameters used.

B.1 Network architecture

Two network architectures were used in this work to evaluate the *planning for exploration* methodology. These architecture are divided between agents that used ensemble-variance as an uncertainty mechanism, and the agents that didn't. Both architectures use blocks of feed-forward networks for every estimator used by the agent: 1) the representation function $g(o)$. 2) the transition dynamics $f(s, a)$. 3) the reward function $r(s)$. 4) the value function $v(s)$ and 5) the policy function $\pi(s)$.

Representation function block This feed-forward network consisted of an input layer of size 1 (the dimensionality of the observation space), a hidden layer of size 16, and an output layer of size 4.

Transition dynamics function block This feed-forward network consists of an input layer of size 7 (state-abstraction-encoding size of 4, and action space of 3), two hidden layers of size 16, and an output layer of size 4.

Reward & value function blocks These two feed-forward blocks have identical architecture, consisting of an input layer of size 4, two hidden layer of size 16 and an output layer of size $support \cdot 2+1$, for a categorical representation of real numbers, as discussed in section 4.1.3. The support size used was 15, for a output-layer size of 31.

Policy function block This feed forward block used an input layer of size 4, two hidden layers of size 16, and an output layer of size 3, the size of the action space.

Ensemble architecture The architecture of networks used by the ensemble-using agents formulated the relevant blocks (reward and value blocks) as ensembles rather than individual blocks. This translates to having 5 (the ensemble size used) independent blocks of reward, and 5 of value. The prediction from the block is taken as the average of the individual blocks' predictions.

B.2 Hyperparameters configuration

We divide the hyperparameters into 3 distinct classes: 1) *planning for exploration* target-adaptation parameters, such as whether to use n-step or 0-step targets. These parameters are described in the experimental setup, section 4. 2) Network-architecture details. These parameters are described in a dedicated appendix, B.1. 3) Additional hyperparameters, such as number of training steps, batch size, learning steps decay, etc. These hyperparameters are detailed in this section, in table B.1.

	Slide	Slide, abl.	Mountain Car	Mountain Car, abl.	Comment
Num. of stacked obs.	1	1	1	1	-
Discount parameter γ	0.95	0.95	0.997	0.997	-
Planning nodes budget	30	30	200	200	1
Root dirichlet α	0.25	0.25	0.25	0.25	-
Root exploration fraction	0.25	0.25	0.25	0.25	-
UCB's pb-c-base	19652	19652	19652	19652	2
UCB's pb-c-init	1.25	1.25	1.25	1.25	3
Training steps	70000	45000	120000	100000	4
Traning ratio	2.25	2.25	1.75	1.75	
Batch size	128	128	128	128	-
Value-loss weight	1	1	1	1	5
Training hardware	distributed CPUs	distributed CPUs	distributed CPUs	distributed CPUs	-
Learning rate λ	0.02	0.02	0.02	0.02	6
Rate of decay	0.9	0.9	0.9	0.9	
Decay steps c_{steps}	500	500	2000	2000	
Replay buffer size	500	500	1000	1000	-
Unroll steps in loss	10	10	10	10	-
n-step target's n	50	50	50	50	-
Prioritized replay	0.5	0.5	0.5	0.5	7
Reanalyze	True	True	True	True	8

Table B.1: Hyperparemeters used in the results presented in chapter 5

We provide a list of comments for additional details regarding some of the hyper-parameters:

1. The number of nodes in each MCTS planning tree. Preliminary results in Mountain Car with planning budget of 50, showed the same behavior as in the results presented in section 5.1 but with lower stability.

2. A UCB parameter used by MuZero's MCTS variant. For more details, see [2].

3. A UCB parameter used by MuZero's MCTS variant. For more details, see [2].

4. The implementation maintains a ratio of *Training ratio* between the training steps and the environmental steps. A ratio of x represents x training steps for each environment steps. Additional results attained but not presented in the work experimented with up to 300000 steps. The behavior observed was the same as the one showed in the results presented. Observing that the large number of training steps does not appear to be necessary to demonstrate the capacity of our method, and due to compute and time resource limitations, the experiments with the final hyperparameters, presented in chapter 5, were conducted with a smaller number of timesteps.

5. MuZero enables scaling of the different losses in the loss computation independently.

6. The learning rate decay is computed as follows: $\lambda\rho^{n/c_{steps}}$, for n the current training step count.

7. The sampling-priorities of trajectories in the replay buffer are computed as the absolute value of the difference between the value and the value target, to the power of this hyperparameter.

8. The rudimentary implementation of MuZero-Reanlyze used in this implementation replaces old value estimates from planning trees, with new value estimates from the value function.

B.2.1 Temperatures

Two ranges of temperatures were used in each environment. The regular temperatures were used to evaluate the vanilla agent, while the low temperatures were used to evaluate the exploratory agent. The exact temperature values are specified in table B.2. The regular temperatures' values switched at 0.3, and 0.5 of the total training step budget. The low temperatures' values switched at 0.3, 0.5 and 0.75 of the total training step budget.

	Slide	Mountain Car
Regular	1.0, 0.5, 0.25	1, 0.5, 0.25
Low	0.75, 0.25, 0.175, 0.02	0.5, 0.25, 0.175, 0.1

Table B.2: Temperature ranges used in this work

Appendix C

Training

The training framework used in this implementation, developed based on the implementation by [23], which is in turn based on the framework proposed by Deep Mind in [2], is highly parallelized. It consists of several dedicated threads: a replay-buffer worker, shared-storage worker, reanalyze worker, training worker, self-play workers, as well as a master worker. The replay-buffer worker maintains the replay buffer. The shared storage maintains the weights of the networks, as well as other information such as current environmental steps, trained steps, last reward, etc. The reanlyze worker iterates over the replay buffer, samples games uniformly randomly, and replaces the value target bootstraps' from old MCTS-planning trees' value estimates, to outputs of the current value function. The training worker randomly samples trajectories from the replay buffer according to a priority. In order to prevent over-training on a small amount of data, the trainer maintains a constant ratio, which is a specifiable hyperparameter, between the trained steps and the environmental steps. The self-play workers update the weights of their networks from the shared-storage at the start of each episode, play an episode, and store the relevant information in the replay buffer and shared storage. Finally, the master worker runs a constant evaluation of the agent, by running greedy evaluation episodes, that are not used for training, with the most recent version of the weights of the trained networks, from the shared storage.

We trained the agent on a non-stationary, distributed computation architecture using the TUDelft's INSY computation-cluster operating based on SLURM [86]. The architecture is none-stationary in the sense that different computation nodes with different hardware were provided by the cluster for each experiment independently. All agents were trained on CPUs. Training on GPUs was not dimmed necessary due to the relatively small size of the network-architectures used.