NTNU
Norwegian University of
Science and Technology

Intrinsic Motivation from Distributional Reinforcement Learning

Olav Bjørnstad Markussen

Master of Science in Computer Science
Submission date: September 2018
Supervisor: Massimiliano Ruocco, IDI
Co-supervisor: Humberto Castejon, Telenor Research
Arjun Chandra, Telenor Research

Norwegian University of Science and Technology
Department of Computer Science

I would like to thank my supervisor Massimiliano Ruocco and co-supervisors Arjun Chandra and Humberto Castejon for proof-reading and excellent guidance.

I also want to thank Aleksander Wasaznik for fruitful discussions and valuable insight throughout the year.

I want to thank Gorm Rødder for proof-reading and encouraging words.

Finally, I want to thank my girlfriend Ingeborg Hollekim Bringslid for her unrelenting support and optimism.

Thank you.

- Olav Bjørnstad Markussen

English Abstract

Reinforcement learning is learning to behave optimally with respect to an external observer through interactions with an environment. An agent repeatedly tries to accomplish a goal, each trial yielding some more information about the environment. Recent work by Bellemare et al. (2017) introduce a technique, C51, that extends the point estimate of future reward to a probability distribution. This opens the door for new action-selection schemes and exploration strategies. It is also a possible source for intrinsic motivation, using uncertainty to generate directed exploration. Recent work by Moerland et al. (2018) presents promising results when using distributions to explore in a deterministic MDP setting by way of Thompson sampling. Their results also prove that this way of representing returns are a valid option to guide exploration. This thesis introduce a novel way of computing intrinsic reward based on distributions from the C51 algorithm. The resulting intrinsic reward enables the agent to quickly explore a new environment, resulting in a performance on par with Moerland et al. (2018) in the randomized Chain environment.

Norwegian Abstract

Forsterkningslæring er å lære oppførsel i et miljø gjennom interaksjoner. En agent prøver gjentatte ganger å fullføre et miljø, og får en mer presis mening om konsekvenser for hvert forsøk. Ny forskning gjort av Bellemare et al. (2017) introduserer en ny teknikk, C51, som forbedrer punktestimatet av en observasjon. De går fra å bruke ett enkelt tall(forventningsverdien) til å approksimere den underliggende sannsynlighetsfordelingen. Tilgangen til en sannsynlighetsfordeling åpner døren for nye væremåter og utforskningsstrategier. Ved å utnytte usikkerheten i meningene til agenten kan vi utforske ved hjelp av en indre motivasjon. Dette kan for eksempel være nysjærrighet eller et ønske om mangfold. Moerland et al. (2018) klarte nettopp dette. De brukte sannsynlighetsfordelingene til utforskning i et deterministisk miljø. Resultatene beviser at distribusjonene kan med fordel brukes til utforskning. Denne oppgaven bruker også distribusjonene til utforskning, men på en annen måte. Ved å utnytte sannsynlighetsfordelingene til en C51-basert agent kan vi gi opphav til indre motivasjon. Motivasjonen brukes som utforskningsmiddel, og lar en agent raskt utforske et nytt miljø. Resultatene er på lik linje med Moerland et al. (2018).

Table of Contents

List of Figures

Chapter 1

Introduction

An introduction of the thesis starts the chapter. The project formulation and research question follows. The main contributions are then summarized. An outline of the remaining thesis concludes the chapter.

1.1 Introduction

Reinforcement learning is acquiring knowledge from interactions with the world. The field has seen several major advances the last few years. A computer is now able to compete and supersede humans is a vast number of games, among them the challenging board game Go (Silver et al. (2016)). A number of smaller contributions made this achievement possible, some of them tackling the exploration/exploitation challenge. One of these contributions is the introduction of a value function. It estimates the expected world-given goodness of a given game-state. This technique has in recent work been extended, estimating the underlying probability distribution function rather than only the expectation. The inherent curiosity in humans has been formalized as *intrinsic reward*, and is a natural way of exploring uncharted territory. However, there is no definite way of computing it, leaving a vast open field of research. We set out to investigate if the recently introduced probability distributions have any merit to direct the agent when it feels lost.

1.2 Project formulation

The C51 algorithm Bellemare et al. (2017) takes a new approach to value-functions. Rather than approximating the expected value, C51 models the distribution of returns and their probabilities, introducing distributional RL. However, the algorithm still uses an old method called e-greedy for exploration. Is there room for improvement on exploration in the distributional RL setting, specifically using intrinsic reward?

1.3 Research Question

This thesis studies intrinsic rewards in the distributional reinforcement learning setting. First, an overview of the current research is presented. Allowing for putting this work into context. The main contribution will then be presented, answering the following research question: **Can the information provided by state-action distributions in distributional reinforcement learning give rise to intrinsic motivation? If so, are they beneficial to the agent?**

1.4 Contributions

The first part of the thesis consists of a literature review and background in reinforcement learning. A new method for intrinsic motivation is presented in section 4.1.1. The modifications and specific implementation is motivated by experiments in section 5.2. Our contributions are:

- Results on several gridworld-type of environments, showing how intrinsic and non-intrinsic agents behave in stochastic and deterministic environments.
- An understanding of probability distributions output by the C51 algorithm.
- A new intrinsic reward function based on said probability distributions.

1.5 Structure

A background section providing essential knowledge on reinforcement learning is first presented. Distributional reinforcement learning is also presented in detail. The state of the art in utilizing distributions for exploration are reviewed and set in context with this thesis. Other state of the art methods on exploration are also discussed. Methods and the resulting agent are then discussed in detail, giving a thorough understanding of the new method. Our path to the specific implementation are then told through the experiments section. Results leading up to an answer to the research question is then presented. A discussion of the overall work follows. The conclusion then touches on the main parts of this thesis, summarizing our findings.

Chapter 2

Background

This chapter concerns the reinforcement learning problem and how it is solved. A presentation of the RL setting will be presented, followed by the environments used in this thesis. The agent is then discussed, and the process of learning introduced. Then follows how the agent makes a guess from a guess, by traditional Q-learning and by distributional reinforcement learning. How to compute intrinsic motivation follows. Different kinds of uncertainties and how the agent should act accordingly to different uncertainties are then explained. The chapter concludes with an explanation of the underlying function approximator used in this thesis.

2.1 The Reinforcement Learning Problem

Solutions to many problems are defined only by the desired outcome. In chess, the task is to win. How to win is not given. Walking is characterized by moving forwards (at minimum energy expense). The exact relation between sensory input and muscle movements is not important. Reinforcement learning (*RL*) is the search for algorithms solving this class of problems autonomously.

2.1.1 Environment

In RL, problems are formalized as *environment*s. We define environments to be a *partially observable Markov decision process* (*POMDP*) Thus for our purposes, an environment consists of a set of *state*s S, set of *action*s A, as well as probability distribution for evolved state as a result of each action in each state $P(\cdot|s,a)$. The environment defines an *observation* as a function of state. If the environment is fully observable, that is, the observation uniquely identifies the state, the environment is a *Markov decision process* (*MDP*). A reward may be positive, negative, or even zero. The environment also describes rewards given for each action in each state, $R(s,a) : S \times A \rightarrow \mathbb{R}$. We call each state transition a *step*. Consecutive steps form an *trajectory*. A trajectory is completed when the last state has all actions guaranteed to lead back to itself. Such a state is called a *terminal state*. A trajectory from a start state to a terminal state is called an *episode*. The goal is always to maximize the expected cumulative reward of steps in an episode (*ECR*). Since the goal is to maximize ECR, an episode is concluded when non-zero rewards can no longer be reached. The *return* in an episode of length T is the discounted cumulative rewards, $G_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{\mathrm{T}-1} r_{\mathrm{T}}$.

This thesis mainly considers three different environments, the NChain from OpenAI gym Brockman et al. (2016), a slightly modified version called Chain-n from Osband et al. (2016), and a grid-based environment the author created during this work.

NChain(figure 2.1) is described on OpenAI's webpage like so: "This game presents moves along a linear chain of states, with two actions: Forward, which moves along the chain but returns no reward. Backward, which returns to the beginning and has a small reward. The end of the chain, however, presents a large reward, and by moving 'forward' at the end of the chain this large reward can be repeated. The observed state is the current state in the chain (0 to n-1)". Slight modifications are made to the standard NChain environment, it isn't slippery anymore, and the agent has just enough time to reach the large reward if it always moves 'forward'.

Chain-n (figure 2.2) bare similarities to the above description of NChain. However, one action leads to the next link, and the other terminates the episode with zero reward. The correct action in each state is randomized at environment initialization, removing the possibility to generalize between states. A reward of 1 is issued at the final link, and the episode terminates. Action randomization proved crucial to distinguish between exploration strategies.

The last environment is the Wall environment, seen in figure 2.3. It is heavily inspired from gridworld-like environments in OpenAI gym. Here, the agent observes the current state, and can always walk in the four cardinal directions. Actions leading off the map are no-ops. Every transition yields no reward, except when reaching the goal, yielding a reward of 1. An

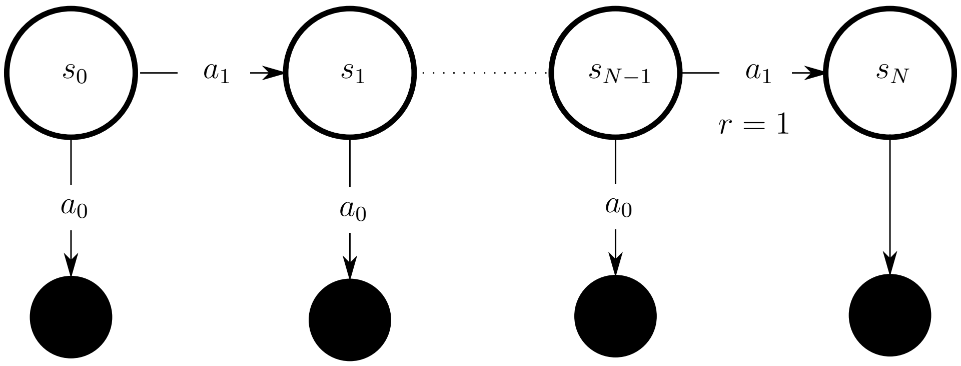


Figure 2.1: The *NChain* environment. Each state has two possible actions. One leading to the next link, and one terminating the episode, depicted by a black circle. Upon reaching the final link in the chain, the agent is rewarded a unit of 1, and the episode terminates.

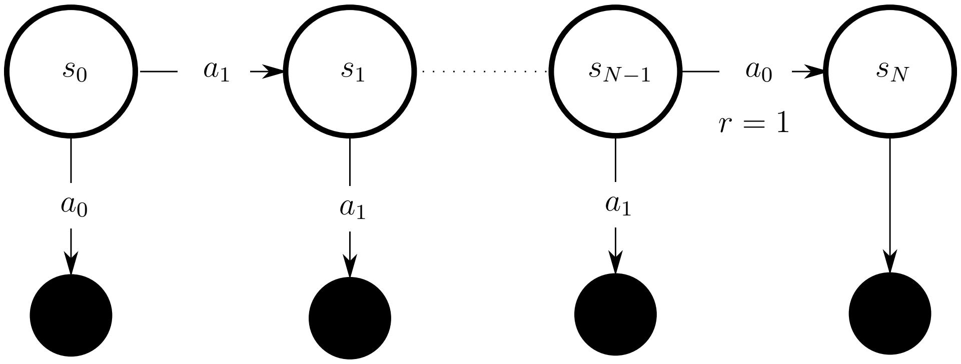


Figure 2.2: The *Chain-n* environment. Each state has two possible actions. One leading to the next link, and one terminating the episode, depicted by a black circle. Upon reaching the final link in the chain, the agent is rewarded a unit of 1, and the episode terminates. All actions are uniformly randomized at environment initialization.

S	F	F	F
F	F	G	F
H	H	H	H
F	F	F	F

Figure 2.3: The *Wall* environment. S is the starting state, F are frozen(walkable) squares, G is the goal and H are holes in the ice that terminates the episode.

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

Figure 2.4: The FrozenLake-v0 environment. S is the starting state, F are frozen(walkable) squares, G is the goal and H are holes in the ice that terminate the episode.

agent can only act in a certain part of the state-space, never experiencing what is on the other side of the wall, illustrated by the line of Hs in figure 2.3.

The FrozenLake-v0 environment from OpenAI gym was used in the experiment phase. Except for the layout, the environment is almost identical to Wall. However, the environment is stochastic. Each time the agent takes an action, there is a $\frac{1}{3}$ chance the agent will slide and execute one of the other three actions.

2.1.2 Agent

An algorithm producing actions is called an *agent*. A chosen environment and agent are allowed to interact. The agent receives observations and rewards and has a *memory* it can update. When a terminal state is reached, the environment is reset to its initial state, and the agent is informed of this by a special observation. An agent with its memory has *solved* an environment when the ECR is maximized (within some margin).

Although an agent may be specialized for the environment, not necessarily improving its performance over interactions, in RL we are interested in algorithms that learn to maximize the ECR. It is also desirable for the algorithm to be as general as possible with regard to choice of environment.

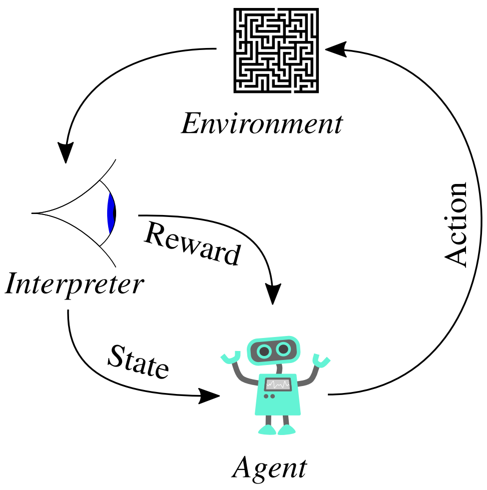


Figure 2.5: The agent-environment interaction.

It is worth noting that the agent and environment are isolated from each other. The environment cannot communicate with or disturb the agent in any way except for emitting observations and rewards. Likewise, the agent cannot influence the environment by any other means than observing and commanding actions. In particular, the environment definition is not in general accessible to the agent. This prevents a general agent from saving or duplicating the environment so as to try different scenarios risk-free. However, an agent may be specialized, possessing a (possibly parameterized) a-priori model of the environment. In this report, we do not give our agents models of the environment.

2.1.3 Structure of an agent

How should an agent choose an action given a particular observation? At the core of an agent is the *policy* π, which selects a stochastic action based on the memory of the agent. A concrete toy example of a policy is random selection of action regardless of memory. In some cases random is the optimal policy, like when playing rock-paper-scisors against an optimal adversary. In more difficult problems, the agent might perform better if it can show a preference in action selection. In a mash-the-button environment, where an agent steps by either idling or pressing a button that gives a positive reward, an agent that has a preference for pressing the button will perform better. A singe action is always best in cases like this. In other environments the most lucrative rewards are not as immediate, but taking a specific action could give access to situations with more rewarding actions later. Here the best choice of action in dependent on the situation.

Suppose the mash-the-button environment is augumented with a lever that changes the reward of the button. No reward is given when the lever is adjusted. The causality between an increased reward, the correct lever position, and the lever adjustment is hidden by many steps. Clearly, the best action to take depends on the *belief* about the lever's position. Belief is simply any inference from memory.

Let us formalize the ECR for a policy that observes the whole state of the environment. Given such a policy, ECR is the expected reward for all possible rollouts from a given state.

An optimal policy will be such that the ECR in every state-action will be equal or higher that the ECR for other policies. Let π^* be some policy that has all ECRs equal or higher to any other policy in that environment. Under any policy acting on states (not observations), every state will have a future ECR. One optimal policy, π^{maxecr}, is the greedy policy applied to ECR. It simply selects the action with the highest ECR.

2.1.4 Q-learning

We are generally interested in environments with large state-action spaces. Computing the exact π^{macecr} is unfeasible in this case, as our agents have space limitations for their memory. Because of this, we have to resort to approximations. π^{maxecr} can be approximated by approximating the ECR. This approach is called Q-learning by the RL-community, and is the core problem to solve. The approximation of $ECR_{\pi^*}(state)$ is called $v(state)$. Remember however that the job of the policy is not to evaluate current state, but to select the best action in current state. The value function is useless alone, as we cannot predict which state an action will lead to. The chosen solution is to instead approximate what we are interested in directly. That is a value for a state-action combination, called the Q-value. Formally, the value that this Q-function should take is max over actions

$$E[r(s, a, s') + v(s') \mid \text{action } a \text{ was taken}]$$

where r is the environment-specific reward-function for doing action a in state s and evolving to state s'. One approximation to the ECR is to do random rollouts from a state until episode termination. Given enough time, this strategy will visit every state-action pair in the current environment.

A combination of random rollouts and greedy policy is called ϵ-greedy. It selects random actions sometimes, but otherwise is like greedy. This policy inherits this guarantee of exploring everything given enough time from the random policy, while being much better at exploring promising actions first due to its greediness.

As the policy is nearing optimal, the q-values can be approximated as simple mean values of rollouts. This can be expressed as an iterative update of the means $\mu_1\ \mu_2 \dots$ of the sequence $x_i,\ x_2 \dots$ as

$$\begin{aligned}
\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j && (2.1)\\
&= \frac{1}{k}\Big(x_k + \sum_{j=1}^{k-1} x_j\Big) && (2.2)\\
&= \frac{1}{k}(x_k + (k-1)\mu_{k-1}) && (2.3)\\
&= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1}) && (2.4)
\end{aligned}$$

However, to begin with the policy is not anything like optimal, so results will drag down the average from the true value. This is alleviated by using exponential moving average instead. This introduces a parameter called the learning rate that determines how quickly an agent distrusts old memories. The best action can be found by iterating over actions, figuring out the values of each state-action pair, and act so as to maximize the ECR. An agent's approximation of ECR will never be exact. be completely sure of state. Instead, we learn Q for the agent's best estimation of state from observation and memory. More formally, we approximate the Q-function as

$$Q(\text{mem}, a) = \mathbb{E}[\text{ECR}(\tilde{s}, a)|\text{mem}]$$

But because we do not want Q to have all the responsibility of interpreting memories, we fist massage the memory into some belief vector.

$$Q(\text{belief(mem)}, a) = \mathbb{E}[\text{ECR}(\tilde{s}, a)|\text{belief(mem)}]$$

We can for simplicity, let belief be the last observation. This is enough for fully observable environments and environments where the optimal behaviour does not depend on non-observable information, such as the previ-

ous state. Both are true for all environments presented in this thesis.
Operators exists that specifies exactly how to update the agent's memory with root in interaction with the environment. One such operator is the Bellman operator. This operator makes updates to the Bellman equations(2.5, 2.6) Bellman (2013), central to describe the value function $Q : S \times A \to \mathbb{R}$.

$$Q^{\pi}(s, a) := \mathbb{E}\, R(s, a) + \gamma \mathop{\mathbb{E}}_{P,\pi} Q^{\pi}(s', a'). \tag{2.5}$$

$$Q^{*}(s, a) := \mathbb{E}\, R(s, a) + \gamma \mathop{\mathbb{E}}_{P} \mathop{max}_{a'} Q^{*}(s', a'). \tag{2.6}$$

The Bellman operators(2.7, 2.8) is used to update the Bellman equations. It also describes the expected behaviour of a Q-learning agent.

$$\mathcal{T}^{\pi} Q(s, a) := \mathbb{E}\, R(s, a) + \gamma \mathop{\mathbb{E}}_{P,\pi} Q(s', a') \tag{2.7}$$

$$\mathcal{T} Q(s, a) := \mathbb{E}\, R(s, a) + \gamma\, \mathbb{E}_P \mathop{max}_{a' \in A} Q(s', a') \tag{2.8}$$

These operators are contraction mappings (Bertsekas and Tsitsiklis (1995)), making the repeated update from an initial Q_0 converge to Q^{π} or Q^{*}, respectively .
We now know how to update the Q-function with information from an environment.

2.1.5 Distributional Reinforcement Learning

We will now discuss how distributional reinforcement learning differs from Q-based, and what possibilities there are. Specifically, we will focus our discussion on the work by Bellemare et al. (2017). The following also applies to other distributional RL work.
Among practitioners of RL, it has been usual to approximate the Q-value with a point estimate(it's expectation). This is very natural, as the agent only needs the expected value of each action to compare the actions and choose the best one. However, estimating the underlying distribution opens up a new range of opportunities to take advantage of. An agent could let uncertainty in the estimate impact action selection. Using distributions has proved useful for risk-sensitive agents by Morimura et al. (2012). The return distribution can also guide exploration, letting the agent take actions that it learns the most from. This is discussed in chapter 3.
An example of a risk-based scenario would be if the agent needed to catch a plane using modes of transport with stochastic travel times. Actions are

choosing which mode of transport of getting to the airport. As the agent doesn't want to miss the plane it must consider this possibility, and take the appropriate action thereafter. This would incentivize the agent to pick a sound mode of transport, minimizing the possibility of missing the plane while still arriving on time. Bellemare et al. (2017) develops a categorical distribution for which to approximate value distributions. Using this approach, an agent has information on the variance and possible multimodal state-action distributions, allowing for complex action selection as seen in Moerland et al. (2018). We can treat the q-value function as a random variable that we so far have only known the expectation of, and of which we now can approximate the full value distribution.
A new update rule is needed to represent the state-action-values as a distribution, and not as a single point-estimate. Bellemare et al. (2017) presented the *distributional bellman update*. This update rule projects updates to state-action values onto a categorical probability distribution, which represents the random return. Regular Q-based agents use the well known Bellman operator $\mathcal{T} : Q \to Q$ for learning. Bellemare et al. defines a policy evaluation operator as $\mathcal{T}^\pi : \mathcal{Z} \to \mathcal{Z}$ where the reward function is a random vector $R \in \mathcal{Z}$, under policy π.

As we did in Q-learning, we need both an optimality operator 2.9 to learn the best policy, and the operator to update under the current policy 2.10. Bellemare et al. (2017) defines the distributional Bellman operator $\mathcal{T}^\pi : \mathcal{Z} \to \mathcal{Z}$ as

$$\mathcal{T}^\pi Z(x,a) \overset{D}{:=} R(x,a) + \gamma P^\pi Z(s,a) \tag{2.9}$$

$$\mathcal{T} Z := \mathcal{T}^\pi Z \text{ for some greedy policy } \pi, \tag{2.10}$$

$\mathcal{T}^\pi$ is also a contraction, converging to Z^π by iterative application from an initial $Z_0 \in \mathcal{Z}$. There are many ways to represent the resulting probability distributions. We focus now on the specific implementation in the paper, namely the categorical representation. It uses a number of supports for the state-action value. These supports z_i are evenly spread within the limits of parameters V_{min} and V_{max}: $\{z_i = V_{\text{MIN}} + i\Delta z : 0 \leq i < N\}, \Delta z := \frac{V_{\text{MAX}} - V_{\text{MIN}}}{N-1}$, effectively being the received returns. A parametric model θ :

$\mathrm{S} \times \mathrm{A} \rightarrow \mathbb{R}$ gives atom probabilities p_i:

$$Z_\theta(s,a) = z_i$$
$$p_i(s,a) := \frac{e^{\theta_i(s,a)}}{\sum_j e^{\theta_j(s,a)}}$$

The updated value function is then projected onto the support after each application of $\mathcal{T}^\pi$. This step crucially reduces the Bellman update from regression to multiclass classification. To update the underlying function approximator(a neural network in this case), a sample loss of categorical cross-entropy is used between the updated projected value distribution, and the old one. Calculating the expected value of $Z(s,a)$ yields $Q(s,a)$, resulting in usage of well known indirect policies, such as the greedy policy.

2.1.6 Off-and on-policy

An agent can learn by interacting with the environment. It can also learn by observing other agents acting in same environment. These ways of learning are called *on-policy* and *off-policy*, respectively. “In off-policy learning we seek to learn a value function for a *target policy* π, given data due to a different *behaviour policy* b” Sutton et al. (1998). The off-policy agent can learn from other agents, or from non-learning processes like a machine or repetitive task done by a human. Q-learning assume that the best action was taken in each step. It is an off-policy algorithm that calculates a guess from the guess about the next state, which is the focus of section 2.1.7. It does not assume that the action took was the best action according to the current policy, which would result in an on-policy algorithm. The same argument goes for distributional learning.
Off-policy learning lets us store all agent-environment interactions in a *replay buffer* in the form of transition-tuples starting at s_t, taking action a_t resulting in reward r_t and next state s_{t+1}, (s_t, a_t, r_t, s_{t+1}). This buffer is then sampled from, usually uniformly, and the agent learns the samples. As we are focused on off-policy learning, we can still use these tuples in a later update even though the current policy may be vastly different than the accountable policy. This results in a dataset the agent repeatedly learns from, and adds to in each training phase. Using an experience replay buffer brings considerable advantages. The same transition tuple might be used several weight updates, improving data efficiency. Sampling uniformly decorrelates the strong intra-episode data correlation. This technique reduces vari-

ance in the update, resulting in a better function approximator Mnih et al. (2013). An experience-replay also reduces parameter divergence and oscillations Tsitsiklis and Van Roy (1997).

2.1.7 Bootstrapping

Bootstrapping is to "learn a guess from a guess" Sutton et al. (1998). The algorithm for Q-learning described above is a prime example, it only accounts for the momentary, or temporal difference from the next guess. Joining the current state's evaluation and the next is done by the update rule

$$\begin{aligned} Q(s,a) &\leftarrow Q(s,a) \\ &\quad + \underbrace{\alpha \left(r(s,a) + \gamma \underset{a'}{max}\, Q(s',a') - Q(s,a) \right)}_{\delta_{\text{TD}}(s,a)}. \end{aligned} \tag{2.11}$$

Which is known temporal-difference zero, or TD(0). The one-step stochastic look-ahead introduce low variance because of a relatively small outcome space. However it inhibits high bias from it's guess in the next state.
Using the immediate next state to approximate the value function is only one possible way of learning in the RL setting. There are options to move away from bootstrapping and use ground-truth instead, namely Monte-Carlo(MC) methods. MC methods learn directly from experience with no bootstrapping involved. This is done by considering the mean return from a state as that state's value:

$$Q(s,a) \leftarrow Q(s,a) \tag{2.12}$$

$$+ \underbrace{\alpha \left[\sum_{t=0}^{\text{T}} \gamma^t r(s_t,a_t) - Q(s,a) \right]}_{\delta_{\text{MC}(s,a)}} \tag{2.13}$$

This technique has the advantage that the estimates was once actually given from the environment, resulting in no bias. However, as the episode get progressively longer, the resulting trajectory becomes decreasingly probable. This makes MC methods have high variance in their estimates. Combining different ways of approximating the value for a state leads to lower approximation error and better learning efficiency (Sutton et al. (1998), Tsitsiklis and Van Roy (1997)). One way is the *mixed Monte-Carlo update* (MMC):

$$Q(s,a) = Q(s,a) + \alpha((1-\beta_{\text{MMC}})\delta_{TD}(s,a) + \beta_{\text{MMC}}\delta_{MC}(s,a)) \quad (2.14)$$

Weighting the one-step update TD(0) and the rollout-based Monte-Carlo by a hyperparameter β_{MMC} this way has been crucial to several RL applications(Ostrovski et al. (2017), Bellemare et al. (2016)). There are many other ways of combining different look-ahead methods, as explained by Sutton et al. (1998) and are out of scope for this thesis.

2.1.8 Intrinsic Reward

So far we have restricted our discussion to extrinsic rewards, that is rewards given to the agent by the environment. Extrinsic reward moves the agent regardless of the agent's own belief. The rewards can be points in a game, failing or succeeding to stack bricks, or some other external metric. *Intrinsic* reward is the polar opposite, arising form the agent's own beliefs about the world. It moves the agent to act so as to maximize enjoyable topics such as diversity and curiosity. There is no set way of computing intrinsic reward, and is an active area of research. However, it is often added to the target update, changing it from 2.15 to 2.16.

$$Q(s_t,a_t) = (1-\alpha)Q(s_t,a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1},a)) \quad (2.15)$$

$$Q(s_t,a_t) = (1-\alpha)Q(s_t,a_t) + \alpha(r_t + r_t^+ + \gamma \max_a Q(s_{t+1},a)) \quad (2.16)$$

One successful way of computing intrinsic reward is the *count-based* intrinsic reward. Count-based intrinsic motivation dates back to Strehl and Littman (2008). The idea is simple, given an environment with a discrete state space, count the visits to each state. The state-count is held in agent memory, usually by the map data structure. Intrinsic reward is then computed directly from the state count:

$$r_t^+ := \frac{\beta}{\sqrt{n(s_t)}}$$

Where $n(s_t)$ is the current number of visits to state s and β is a hyperparameter controlling the intrinsic motivation, usually around 0.01. This type of reward encourages visiting less-visited states.
Given two states with the same reward and next states with predicted re-

ward of 0 by a greedy agent. State one and two has been visited one and two times respectively. The states are reached from two distinct actions, a_1 and a_2. Calculating the update would then result in 2.17 for state one, and 2.18 for state two.

$$r^+ := \frac{\beta}{\sqrt{1}} \tag{2.17}$$

$$r^+ := \frac{\beta}{\sqrt{2}} \tag{2.18}$$

Using the intrinsic rewards to update the Q-function looks like

$$\begin{aligned}
Q(s_0, a_1) &= \alpha(\beta + \gamma \max_a Q(s_1, a)) \\
&= \alpha(\beta) = 0.0001 * 0.01 \\
&= 1 * 10^{-6} \\
Q(s_0, a_2) &= \alpha\big(\frac{\beta}{\sqrt{2}} + \gamma \max_a Q(s_2, a)\big) \\
&= \alpha\big(\frac{\beta}{\sqrt{2}}\big) = 0.0001 * (0.007) \\
&= 7 * 10^{-7}
\end{aligned}$$

resulting in different values for each state-action pair, with a_1 being the highest. As the policy selects actions greedily, it will compare all actions available in s_0 and choose to execute a_1. This is the exact behaviour we want with the count-based IR, visiting less visited states.

2.1.9 Aleatory and Epistemic Uncertainty

As distributional RL discussed in this thesis models uncertainty, it is important to know which uncertainty is modeled, and what it means in this setting.
In reinforcement learning, the agent interacts with an environment, and produces data like state-observations and actions. If the environment is stochastic, it has inherent *aleatory uncertainty*. Aleatory uncertainty, or statistical uncertainty, represents the unknowns that vary each time we run an experiment. We might not know exactly the reward from a given state, because we might either reach the goal, or fall into a hole.
When the agent interacts with the environment, the *epistemic uncertainty*

is decreased for every timestep. Epistemic, or statistical uncertainty, is reduced as more samples are available, and better guesses can be made by the agent. The epistemic uncertainty is highly desirable, because it tells us something about how far we are from the true value distribution. An agent equipped with this knowledge would be able to take actions as to maximize it's information-gathering potential Tang et al. (2016).

2.1.10 Acting on the expectation of distributional RL

The return distribution as approximated in this thesis may be impacted by two sources of uncertainty: a stochastic policy, and a stochastic environment. This point is stressed by Bellemare et al. (2017), saying that "We emphasize that Z^π describes the intrinsic randomness of the agent's interactions with its environment, rather than some measure of uncertainty about the environment itself". Removing uncertainty induced by the environment only leaves room for the agent's policy to introduce uncertainty. As this policy is updated, it can shift the return, in way of maximizing accumulated reward. The epistemic uncertainty is used to compute intrinsic motivation. That is why this thesis will primarily focus on deterministic environments. A stochastic environment would induce additional noise to the return distribution, for which an agent shouldn't act on the expectation.

2.2 Deep Reinforcement Learning

So far we haven't talked about the approximator to $Q(s, a)$. We started off the chapter by saying that large spate-action spaces are lucrative, leading to an approximation of ECR. In smaller environments however, we can in fact compute the exact value, by using a lookup table, where each discrete state is updated separately form each other. This method is unfeasible when it comes to larger state spaces. In that case, we need a function approcimator that can generalize from states. The current well-used approximator is the artificial neural network LeCun et al. (2015). Artificial neural network are trainable and is especially suited to the RL setting, where data is abundant. Agents trained in from-pixel environments like the Arcade Learning Environment Bellemare et al. (2012) often relies on neural networks to distill the actions from states. Artificial neural networks consists of nodes that are non-linear functions of the sum of it's inputs and form a directed weigthed graph. The nodes are stacked in layers, each layer connecting to the next by trainable parameters. An all-to-all connection is common between one layer and the next. There might be several nodes in each layer. Other architectures that stack nodes differently exists, but we only use the *feed-forward* neural network in this thesis. The feed-forward network suits our problem perfectly, as other architectures tackle the problem of image recongnition and temporal data, of which we have none.

2.2.1 Double Q Network

Q-learning use the same function approximator to decide on the best action in a state, as well as approximate the state-action value. A problem arise if a state-action pair is overestimated. If an action's value is overestimated, it is more likely to be chosen as the best action. The action's overestimated value is then used as a target in the next update, inducing bias in the function approximator. The overestimation thought to occur because the expectation of a maximum is greater than or equal to the maximum of an expectation van Hasselt (2013). The idea is to introduce a second set of parameters van Hasselt et al. (2015), used to approximate the state-action value. Given two sets of parameters, θ and θ', for a function approximator, we can rewrite the Q-learning update rule from equation 2.19 to equation 2.20, utilizing the bias-reducing network:

$$Q_\theta(s,a) \leftarrow Q_\theta(s,a) + \alpha(r(s,a) + \gamma \underset{a'}{max}\, Q_\theta(s',a') - Q_\theta(s,a)) \quad (2.19)$$

$$\begin{aligned} Q_\theta(s,a) &\leftarrow Q_\theta(s,a) \\ &\quad + \alpha(r(s,a) + \gamma\, Q_{\theta'}(s', \text{arg}\underset{a'}{max}\, Q_\theta(s',a')) - Q(s,a)) \end{aligned} \quad (2.20)$$

Notation of the Q-function parameters is introduced as Q_θ. The regular update rule as introduced earlier is written here with the parameters θ in equation 2.19. Inclusion of second set of parameters θ' and target network in equation 2.20. The rewriting is motivated by a reduction in bias, expecting that biases of the two networks cancel out(assuming the second approximator don't inhibit the exact same noise as the first), resulting in zero bias. It is the author's belief that this reduction in bias will result in better exploration.

Chapter 3

State of the Art

This chapter discerns the current state of research on distributional reinforcement learning and intrinsic motivation. Approaches that directly use distributions as a source of exploration are reviewed first. Methods that use traditional Q-learning and other intrinsic methods follow.

3.1 Distributional view on reinforcement learning

Using distributional reinforcement learning to guide exploration is a relatively new idea. The distinction between uncertainty originating from limited data(epistemic) and uncertainty from the environment(aleatory) was first discussed and used to guide exploration in Moerland et al. (2017). Epistemic uncertainty is computed by applying dropout masks to a neural network, which approximates the posterior predictive distribution. Aleatory uncertainty is captured by a distributional action-value function that assumes the returns from the environment are Gaussian. This is unlike Bellemare et al. (2017) that use a categorical distribution as output.
The authors create a function approximator that combines both uncertainties, allowing for exploration based on a mix of the two. This works by propagating the return distribution weighted over the parametric uncertainty at the next timestep.

$$Z(s,a) = r + \gamma \int Z_{\phi}(s',a')p(\phi|H)d\phi \tag{3.1}$$

where ϕ are the parameters of the neural network, H is the observed dataset, and $p(\phi|H)$ is the posterior distribution. The study only considers deterministic environments, as the agent shouldn't act on the expectation on the value function in a stochastic environment, explained in chapter 2.

Moerland et al. (2018) has concurrently with this work used the return distributions for exploration in a deterministic MDP. The paper identifies the potential of the return distribution for informed(directed) exploration. A deterministic MDP offers the ability to act optimistically on the return distribution in the face of uncertainty. This is because in such a setting, the modeled uncertainty only originates form the policy itself. Meaning that if some reward is seen once, it is possible to consistently achieve it. This idea is explored for both categorical, Gaussian and Gaussian mixture models. Specifically, their method initialize all PDFs as uniform distributions, and take action by using Thompson sampling or UCB. This combination leads to a lot of exploration in the start of training, and smoothly transitions to exploitation as the PDFs narrow over time. The action-selection scheme is interesting, as it isn't governed by the standard epsilon, needing no hyperparameters or decay schedule. Like Moerland et al. (2017) they train their agents in the Chain environment, described in chapter 5.1. Their new method outperforms the old uncertainty-based method by more than a tenfold, behaving optimally after 6 hundred episodes in stead of 10 thousand. Using the return distribution to compute intrinsic rewards is not explored.

3.2 Intrinsic motivation

Intrinsic motivation has in recent work been generalized beyond count-based exploration and tabular environments. Bellemare et al. (2016) provides a technique allowing for significantly improved exploration in hard from-pixel environments, such as Atari 2600-games. This is achieved by what they call *pseudo-count*s. They first introduce a density model $\rho_n(s) := \rho(s \mid s_{1:n})$ that provides a probability distribution for a state given a trajectory. A CTS model Bellemare et al. (2014) approximates the densities. It is then used in the pseudo-count function:

$$\hat{\mathrm{N}}_n(s) = \hat{n}\rho_n(s)$$

Where $\hat{\text{N}}_n(s)$ is the pseudo-count of state s and $\hat{n}$ is the *pseudo-count total*. The resulting intrinsic motivation bonus takes the form of the scalar value

$$r_n^+(s,a) := \frac{\beta}{\sqrt{(\hat{\text{N}}_n(s) + 0.01)}} \tag{3.2}$$

The hyperparameter β was introduced in chapter 2 and controls the amount of intrinsic motivation. Their agent does not need to learn the environment as a whole to have meaningful intrinsic rewards. Regular count-based-rewards and -exploration via learned hashing(Tang et al. (2016)) also hold this property. Distance-based exploration as proposed in this work relies on the same function to take actions and to provide motivation. There is no obvious advantage for using one or the other, and this is an active area of research.

Ostrovski et al. (2017) extends upon the previously mentioned work, improving accuracy and expressiveness in the density model while preserving the overall ideas. The CTS density model is replaced by a neural density model for images, using neural network approaches just as in chapter 4. Specific training is required by the new approach, and changes is made to how hte pseudo count is computed. However, the intrinsic reward is familiar and expressed as $r_n^+(s,a) := (\hat{\text{N}}_n(s))^{-1/2}$.

Count-based exploration has recently been extended to high-dimensional state spaces. The method was previously restricted to discrete state-action spaces, introduced in section 2. Tang et al. (2016) managed to generalize the state-count by domain-dependent learned hash codes. The hash $\phi : \text{S} \to \mathbb{Z}$ discretizes the state space, and is used to calculate the exploration bonus, defined as

$$r^+(s) = \frac{\beta}{\sqrt{n(\phi(s))}}. \tag{3.3}$$

Note how this is slightly different from the original count-based approach, using $\phi(\cdot)$ to reduce the state space considerably. The authors introduce both a static method and a learned hash, the latter performed best. A complex autoencoder constitutes the learned hash code, providing a latent space of 512 sigmoidal activations. A binarization is then applied to the activations, resulting in a binary representation of the state space. Transforming the state space this way is a lossy compression, and will group states that are close to each other in the same representation. The representations are directly used as the keys in the regular count-based map.

Chapter 4

Methods

Probability distributions produced by the distributional approach is presented. The chapter then focuses on how to compute intrinsic motivation from said distributions. Afterwards, I combine the discussed methods into an agent that learned the Chain-n environment.

4.1 Visualizing the output Distributions

We set out to conduct a series of experiments to address the possibility of using the C51 algorithm for exploration. A presentation of the progress follows in this section. We are looking at the implementation by Bellemare et al. (2017)

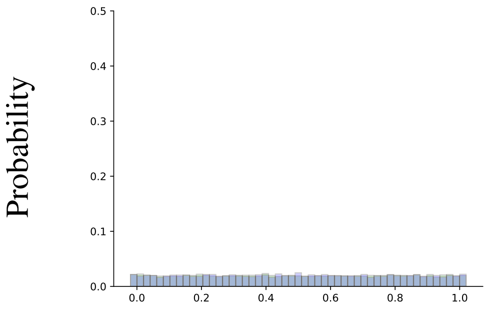
The implementation is based on the approach and algorithm described in Bellemare et al. (2017), i.e the C51 algorithm. The algorithm outputs PDFs for all state-action pairs and is described in chapter 2. The expectation Commentof those PDFs is referred to in literature as the Q-value. Properties of these PDFs could be used in different action selection strategies, be it the variance, kurtosis, or other moments. Experiments were conducted on using such properties as action-selection strategies for guided exploration. However, the experiments turned out negative.

First, we wanted to understand what the distributions themselves looked like under various circumstances, such as in the beginning of training and when an environment is learned. It is also interesting to look at the distribution at various points in between, and with different distance from the goal state. Doing this would help us understand how the C51 algorithm could be used to guide exploration. A categorical distribution allows for multimodal distributions. As interesting as using such distributions to guide actions selection, it has not been in focus of this report.

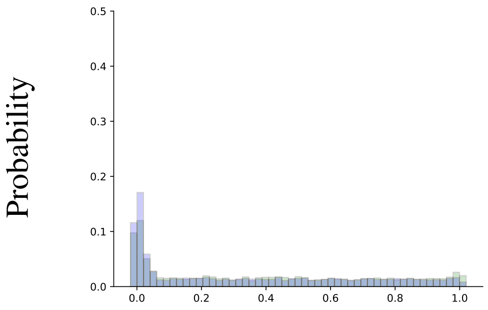
A C51-agent was set out to learn the NChain environment, recording probability distributions during the learning phase. Resulting graph is seen in figure 4.1. The network has no initial bias to the value of a state. It starts out as a quasi-uniform distribution within the boundaries as mentioned in chapter 3. By inspecting our PDFs, we can observe the mass move noisily towards the expected value of the state-action pair as in figure 4.1b. The noise is hypothesized to originate from updates in the neural network and a changing policy. Over time the distribution usually takes the shape of a Gaussian, seen in figure 4.1c. A narrow Gaussian only appears when the agent has experienced consistently the same return for a state-action. This is because acting optimistically on the distribution in a deterministic environment lowers the variance. This crucial property is used for intrinsic motivation.

4.1.1 Intrinsic Motivation from output distributions

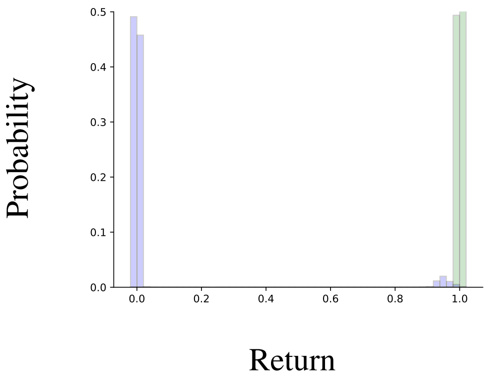
From the exploration of distributions (figure 4.1) we see that the PDFs for a state-action pair change over time. That is, as the agent repeatedly visit a given state and select a given action at that state, the corresponding PDF changes. The group wondered if this fact could be exploited. Specifically, we wondered whether we could exploit the difference between the PDF at times t and $t + 1$ to determine that a state has already been visited and a given action already tried in that state. To achieve this, an agent would need to keep track of the PDF for each action-pair for recent time steps. This is unfeasible for high-dimensional state spaces, unless we applied techniques to encode states, as in the papers reviewed in chapter 4. Instead of comparing the PDF of a state-action pair with itself to determine whether it changes over time, we could also compare it with a reference distribution.
Looking again at the results from exploring of the distributions (figure 4.1), we notice that the PDFs start as a uniform distribution and as time pass by, they move away from the uniform distribution and closer to a Gaussian distribution. Based on this fact, an agent could compare the PDF of a state-action pair with the uniform distribution to determine if it is changing over time. This information could induce intrinsic reward.
From the above remark, the group hypothesized that the PDFs of a less visited state-action pair would be closer in terms of a divergence to the uniform than a frequently visited state. The hypothesis led to an exper-



(a) At the start of learning.



(b) In the middle of training.



(c) Environment is learned.

Figure 4.1: State-action distributions at different stages of learning in the NChain environment for the last state. Each colour represents one distinct action.

iment where the agent could only act and observe a subset of the state-space, using the environment in figure 2.3. Here, Only two thirds of the environment is available for exploration. The last third is hidden behind a wall, never presented to the learner. The agent recorded the Kullback-Leibler divergence(KL-divergence (Kullback and Leibler (1951)) for all state-action distributions in the entire state-action space at certain intervals during training. Meanwhile, the average distance to both observed and unobserved state-actions was measured and recorded. The distance to an unobserved state proved to be higher than to the observed states, both during and after training. This is crucial and identical with the discussion above. It allows for an intrinsic reward with root in the chosen distance function.
Different distance functions were tried. As the C51 algorithm defines a range for it's atoms, we also wanted the distance function to inhibit the same properties, and KL-divergence is unbounded. A distance-function that better suits our needs is the Hellinger distance (equation 4.1) for discrete probability distributions Hellinger (1909), bounded in the range $[0, 1]$:

$$\mathrm{H}(P, Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{k} (\sqrt{p_i} - \sqrt{q_i})^2} \tag{4.1}$$

Where $P = (p_1, p_2, \ldots, p_k)$ and $Q = (q_1, q_2, \ldots, q_k)$ are two discrete probability distributions. Hellinger distance worked comparably to KL-divergence, and is our chosen distance metric between the PDFs.

Results from the Wall-experiment described in chapter 2 lead the group to device an intrinsic reward method based on the distance from a uniform distribution to a state-action distribution.

A distance metric was implemented to quantify the difference between two probability distribution functions. This metric and it's usage is the central part of this report, as it eventually evolved into determining the intrinsic reward in a state. Remember, count-based reward is defined as $r^{+}(s) := \frac{\beta}{\sqrt{n(s)}}$, the distance metric could be used as a proxy for the visit-count($n(s)$). Defining the distance function for one PDF to be

$$d(s, a) = D_{\text{Hellinger}}(Z(s, a)|\text{uniform}), \tag{4.2}$$

we need a sum that takes all actions into account for a given state. Let $d(s) = \sum_{i=0}^{N} d(s, a_i)$ where N is the number of allowed actions, a in s. An exploration bonus $r^+ : \mathcal{S} \to \mathbb{R}$ is added to the reward function, defined as

$$r^+(s) = \frac{\beta}{n(s)} \tag{4.3}$$

where $\beta \in \mathbb{R}_{\geq 0}$ is a hyperparameter controlling the amount of exploration bonus, and $n(s) = e^{d(s)}$ is the count-proxy. The distance $d(s, a)$ starts out close to zero, leading to especially high exploration at the start of interaction with the environment.
The choice of denominator proved crucial as the Hellinger distance for a PDF is maximum of 1. $d(s)$ will therefore have a maximum value equal to the number of possible actions in a state. This is undesirable, as we want our distance function to decay rapidly for a well-visited state. Changing the state-count proxy to $e^{d(s)}$ is needed to obtain the desired behaviour. Our chosen scaling moves away from the proven inverse-square-root dependence Strehl and Littman (2008). However, $\frac{\beta}{\mathrm{N}(s)}$ is also a well used exploration bonus Kolter and Ng (2009).

Mixed Monte-Carlo Backup is used to speed up the learning process, as explained by Ostrovski et al. (2017) and chapter 2. The nature of this backup makes state-action distributions change faster, resulting in better exploration and earlier exploitation. The MMC backup is proven to be crucial when learning difficult exploration games like Montezuma's Revenge (Bellemare et al. (2016), Ostrovski et al. (2017)).

4.1.2 The resulting agent

The above methods are all combined to an agent capable of solving MDP-like environments, described in chapter 2. As the methods are distilled by themselves, we shall now focus on how they are assembled to an agent. It should be noted that this work is an extension of Bellemare et al. (2017), and the algorithms here are extensions of that work. A standard agent-environment loop, described in chapter 2, can be seen in Algorithm 1. At each timestep the agent takes an action, and observes the impact on the environment. This interaction-data is recorded as an $episode$, eventually placed in the agent's replay memory. Notice that the replay memory D consists of episodes, not SARS-tuples that a one-step TD learned would. Every so often, the agent starts training from it's replay memory, pausing the interaction loop for better estimates about the future. As the agent inhibits two architecturally equally function approximators, but parametrically different, the target network is sometimes updated with new learned information from the primary network. The exploration factor ϵ is decayed inverse exponentially, with a cutoff χ, letting it reach 0 faster.

Training is similar to Bellemare et al. (2017), but with three modifications from the original algorithm: 1) the agent now use a MMC backup, 2) intrinsic rewards are added to the target, 3) a double network model is employed.
Given an episode consisting of $(s_t, a_t, r_t, s_{t+1}, \text{done})$-tuples and a greedy policy π w.r.t $\mathbb{E}\, Z_\theta$, the MMC$_{\text{target}}$ and Bellman update is computed $\hat{\mathcal{T}}_{z_j} := r^+(s) + \gamma\text{MMC}_{\text{target}}$ for each atom z_j. The target network's atom probabilities $p'_j(s_{t+1}, \pi(s_{t+1}))$ is distributed to the value network in lines 18 and 19. Intrinsic reward is directly computed from the primary network output as shown in equation 4.2 and added to the target value $\hat{\mathcal{T}} z_j$. The neural network is trained with a cross-entropy loss and the Adam optimizer Kingma and Ba (2014). $[\cdot]_a^b$ bounds the argument between a and b in algorithm 2.

Algorithm 1 The Distance-based agent

```
    Initialize agent with replay memory D and capacity N
    Initialize state-action function $Z$ with quasi-random weights
    Initialize environment
 1: $s_t \leftarrow$ reset state
 2: for $game$ in $games$ do
 3:     $done$ = False
 4:     $episode = [\,]$
 5:     while not $done$ do
 6:         $t \leftarrow 0$
 7:         $action \leftarrow$ agent.get_action($s_t$)
 8:         $s_{t+1}, r_t, done \leftarrow$ environment.step($action$)
 9:         $episode \leftarrow [episode, (s_t, acion, r_t, s_{t+1}, done)]$
10:         $s_t \leftarrow s_{t+1}$
11:         $t \leftarrow t + 1$
12:         if $t \pmod{\text{agent.timestep\_per\_training\_cycle}} \equiv 0$ then
13:             train the agent from replay memory    ▷ Seen in Algorithm 2
14:             if $t \pmod{\text{agent.update\_target\_frequency}} \equiv 0$ then
15:                 update target weights to current weights
16:             end if
17:             if $t \pmod{\text{agent.update\_epsilon\_frequency}} \equiv 0$ then
18:                 agent.epsilon= $\epsilon_0 e^{-\lambda t} - \chi$
19:             end if
20:         end if
21:     end while
22:     D $\leftarrow$ [D, $episode$]
23:     $s_t \leftarrow$ reset state
24: end for
```

Algorithm 2 Categorical algorithm using intrinsic reward, MMC update and a double network architecture.

```
    input An $episode$ sampled from replay memory
 1: Randomly discard the first $n$ tuples in the $episode$
 2: $s_0, a_0, r_0, s_1, \_ \leftarrow episode[0]$
 3: $r^+ \leftarrow \frac{\beta}{e^{d(s_1)}}$
 4: $Q(s_1, a) \leftarrow \mathbb{E}\, Z(s_1, a)$
 5: $a^* \leftarrow \arg\max_a Q(s_1, a)$
 6: $m_i \leftarrow 0, i \in 0, \ldots, N-1$
 7: $\text{MC}_{\text{target}} \leftarrow 0$
 8: for $i$ in range($episode$) do
 9:     $(\_, \_, r, \_, \_) \leftarrow episode[i]$
10:     $\text{MC}_{\text{target}} \leftarrow \text{MC}_{\text{target}} + \gamma^i r$
11: end for
12: for $j \in 0, \ldots, N-1$ do
13:     $\text{TD}_{\text{target}} \leftarrow r_0 + \gamma z_j$
14:     $\text{MMC}_{\text{target}} \leftarrow (1-\beta)\text{TD}_{\text{target}} + \beta\text{MC}_{\text{target}}$
15:     $\hat{\mathcal{T}} z_j \leftarrow \left[r^+ + \text{MMC}_{\text{target}}\right]_{V_{\text{MIN}}}^{V_{\text{MAX}}}$
16:     $b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\text{MIN}})/\Delta z$
17:     $l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$
18:     $m_l \leftarrow m_l + p'_j(s_{t+1}, a^*)(u - b_j)$
19:     $m_u \leftarrow m_u + p'_j(s_{t+1}, a^*)(b_j - l)$
20: end for
    output $-\sum_i m_i \log p_i(s_t, a_t)$                ▷ Cross-entropy loss
```

Chapter 5

Experiments, Results and Discussion

This chapter will expand upon how the group as a whole undertook small steps to arrive at the agent implementation in section 4.1.2. We set out not knowing the characteristics of PDs and their potential for use in exploration. Each propelling experiment will be described in detail, with hypothesis, testing, result and conclusion. From the conclusion we will discuss our findings which propel us to the next experiment.
A discussion on the overall thesis follow. It answers the research question and justifies the research. Lastly, I critically evaluate the study.

5.1 Experiments

The first experiment was simply to understand how variance of PDs behaved in an environment, as we only had an intuition of how PDs behaved at the start of research.
All experiments in this section are easily repeatable. Every experiment behaved the same over repeated trials. Hyperparameters are seen in the Appendix.

5.1.1 Experiment 1: Goal-test

H_0 : The PDs have high variance at start of learning. Lower variance will be observed during and after learning.

Testing: Test an agent on a 20*20 gridworld environment with a goal in the middle and starting position in the top left corner. At certain intervals, record the variance of all actions in all states.

Result: We observe high variance extremely close to the goal, low variance elsewhere.

The high variance is because one action leads to the goal state, while the others don't. This results in a difference in return for the agent, producing variance in the PDs. Figure 5.1 show that variance 'propagates' out from goal to other states, but not too far. The propagation is likely due to the propagating nature of one-step returns, explained in section 2.1.7.

Conclusion: As the values of each state is propagated from the goal, so is the variance. The variance is firstly lowered around the goal state, letting other neighboring states reduce their variance as well.

The observed variance behaviour motivated us know how the PDs behave with respect to the starting distribution(quasi-uniform). The rate and magnitude of change is interesting, as it lets us look into how the agent's belief changes during learning.

5.1.2 Experiment 2: Higher variance on unseen states

The group had been experimenting on gridworlds from OpenAI, but also other self-made gridworlds. It was natural in this stage to rapidly exploit our knowledge and get results.

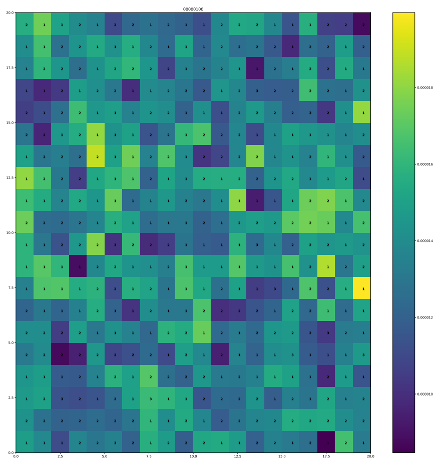
H_0:Given a learned agent, it will have higher distance from a uniform distribution to visited states, than from a uniform to unvisited states. Distance is measured with KL-distance, as explained in section 4.1.1.

Testing: Make a gridworld environment where agent can't explore the complete environment. Measure the average distance for all actions from visited and unvisited states to the uniform at certain intervals. The Wall environment described in section 2.1.1 was created for this purpose.

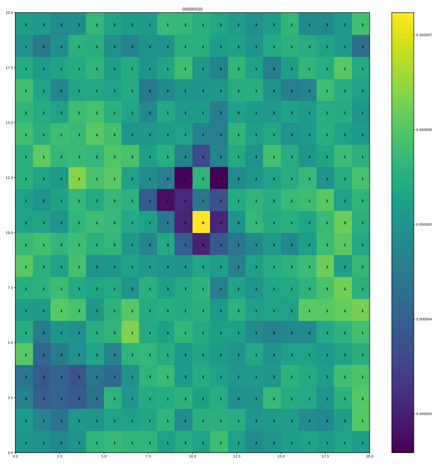
Results: We observe that $D_{\text{KL}}(\text{observed}|\text{uniform}) = 11.046$ and $D_{\text{KL}}(\text{unobseved}|\text{uniform}) = 10.55$ The experiment was also conducted using Hellinger distance, with the same results. The distance to observed and unobserved states was measured during and after training.

Conclusion: Preliminary results on gridworlds show that unvisited states require less amount of bits to code than seen states, given a coding for a uniform probability distribution.

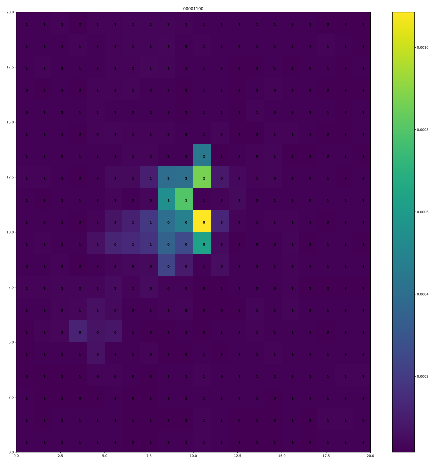
These results were a breakthrough in our research, and allowed the



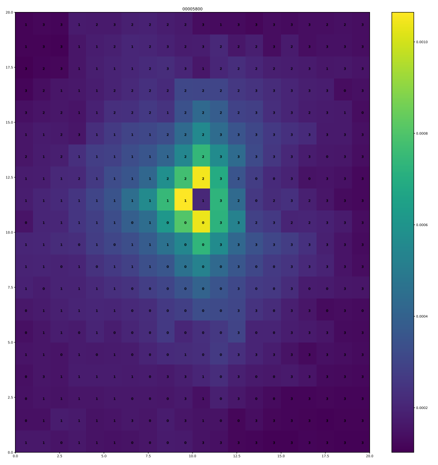
(a) Episode 100



(b) Episode 500

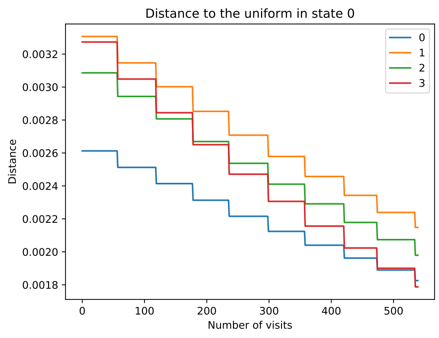


(c) Episode 1100

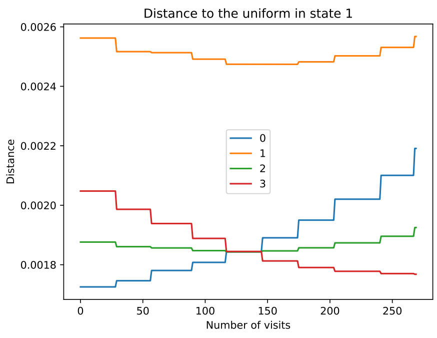


(d) Episode 5800

Figure 5.1: Plots of the variance of output distributions in each state in the 20*20 environment. All squares are walkable, goal in the center, and agent starts in the top left corner. Colourbar is a number line of the variance in each state.



(a) State 0



(b) State 1

Figure 5.2: $D_{\mathrm{KL}}(s|\mathrm{uniform})$ over time in the FrozenLake-v0 environment. Sharp jumps in value are caused by training the function approximator. The four actions are represented as 0 through 4 starting clockwise from 0 being the *North* action. Other states in the same environment look the same.

group to build an intrinsic reward function based on distance. I believe the function approximator is good at generalization. As the PDs start out uniform, the PDs of seen states will rapidly converge to their "correct" values. Updates on seen states will also affect the belief on unseen states because of the mentioned generalization. However, as these states aren't trained on, only the initial uniformness and generalization affect the agent's predictions, resulting in a more uniform distribution on unseen states. Our next question is how much the network changes between two visits to the same state.

5.1.3 Experiment 3: Small updates on consecutive visits to the same state

H_0: Consecutive visits to the same state won't change the distance to uniform much when using $D_{kl}(s|\mathrm{uniform})$ as distance function.

Testing: Create FrozenLake-v0. Register the distance to uniform for all actions in each state visit. Plot distance-history to spot drastic updates in the PDs. The FrozenLake-v0 environment was used in to test the hypothesis.

Results: We observe modest change in the distribution for a state from figure 5.2.

Conclusion: $D_{\mathrm{KL}}(s|\mathrm{uniform})$ does not change much for a single update to a state-action pair in one update. The distance function needs additional scaling to rapidly decrease like the count-based approach by Strehl and Littman (2008)to serve as a count-proxy.

This experiment sparked a discussion to concretize the intrinsic reward. We knew the count-based intrinsic reward decreases rapidly with more visits to a state. We want out intrinsic reward to exhibit the same behaviour. The intrinsic reward as presented in section 4.1.1 was implemented, albeit without the scaling(using e as base and distance as exponent). The updated agent with an intrinsic reward based on distance would be tested of grid-world environments. We thought that using a greedy policy from the start would let the intrinsic motivation-based agents stand out, as their motivation drives them to unfamiliar places.

5.1.4 Experiment 4: Greedy and intrinsic agents

H_0: Doing greedy policy on FrozenLake-v0 is enough to let the intrinsic motivated agent reach the goal substantially faster than the non-intrinsic agent.

Testing: Run different versions of the FrozenLake-v0 environment. Deterministic and stochastic, 4 by 4(the usual) and 8 by 8 versions of the world. Use a greedy policy from the get-go. Record performance at certain intervals.

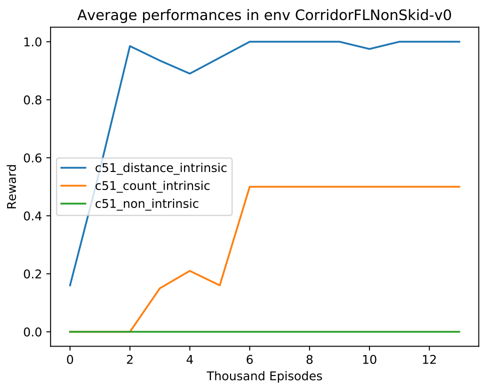
Results: Learning curves are seen in figure 5.3.

Conclusion: There is no real significant distinction between the agents in these environments. This implies the need for a set of environments where we know that the intrinsic reward significantly benefits the agent. The results disprove our hypothesis.

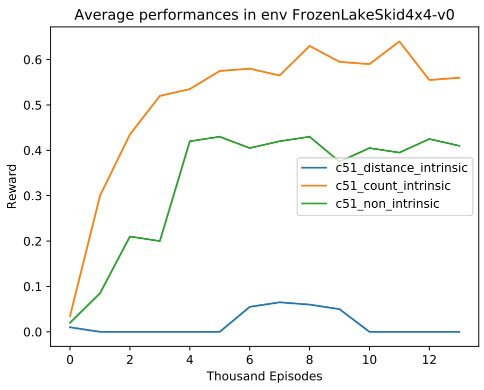
We thought this experiment was enough to set the intrinsic and non-intrinsic agents apart. An environment that highlighted good exploration methods is needed to determine if our intrinsic method has any merit. The work on environments from OpenAI Gym proved nothing, as results were inconclusive. The problem was also discussed by Moerland et al. (2018), Moerland et al. (2017) and Osband et al. (2016). They use the Chain-s environment.

5.1.5 Experiment 5: The Chain-n environment

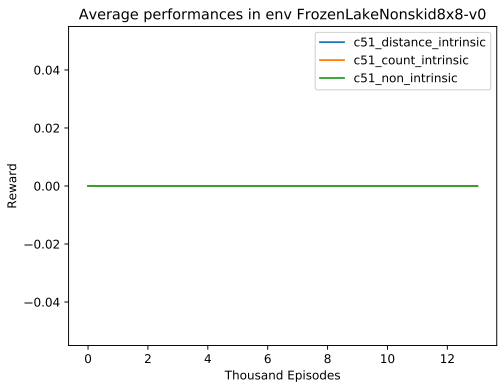
H_0: Intrinsically motivated agents explore the environment faster than a non-intrinsic agent. In turn leading to faster consistent good score in the environment.



(a) Deterministic 4x4



(b) Stochastic 4x4



(c) Deterministic 8x8

Figure 5.3: Learning curves for different gridworld environments. The agents has a one in three chance to slip and do one of the other three actions in a stochastic environment.

Testing: Run all three agents on different lengths of Chain-n. Record performance of all three agents in the different chain lengths.

Results: Both intrinsically motivated agents vastly outperforms the non-intrinsic agent. Learning curves can be seen in figure 5.4.

Conclusion: Through the testing we verified H_0, intrinsically motivated agents do explore the environment faster than non-intrinsic agents in the Chain-n environment.

5.2 Results from Experiment 5

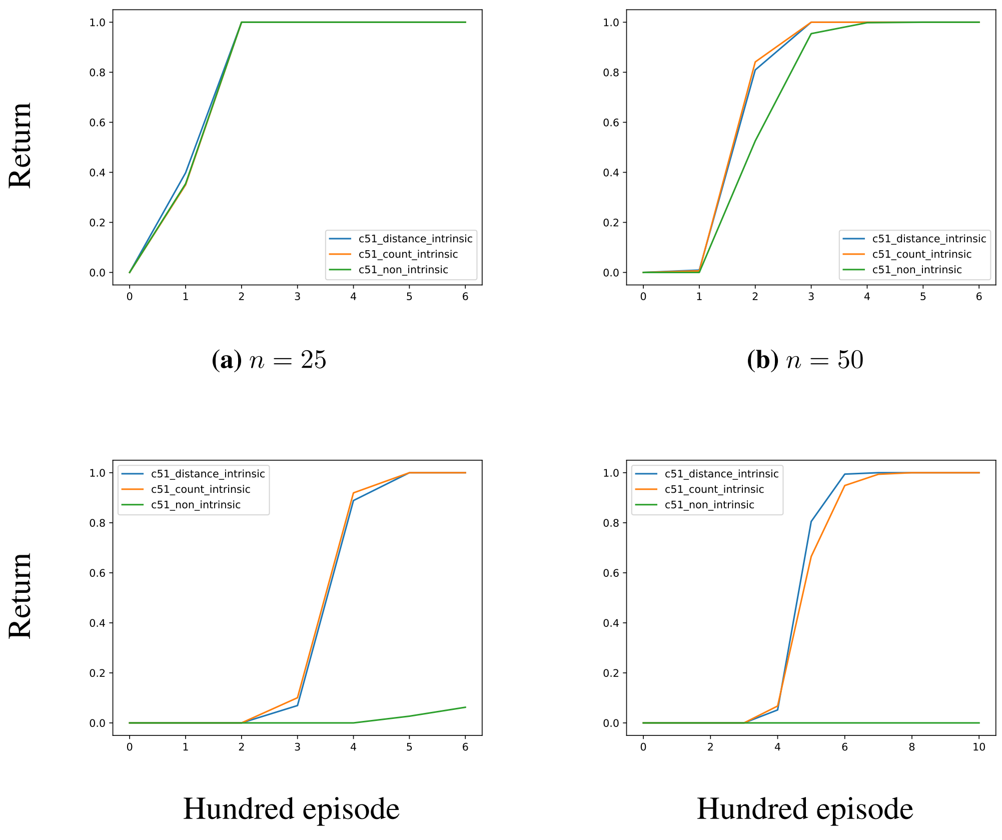
We set out to discover if action-distributions are suited as a source for intrinsic motivation. Experiments were conducted and paved they way for where we should focus our attention. This resulted in an distance metric used to generate intrinsic rewards. The method was tested on the Chain-n environment as described in chapter 4. Three different agents were compared in this environment. Specifically the ϵ-greedy agent, the count-based agent, and the distance-based agent. The *only* difference between the agents is the intrinsic reward function, where the ϵ-greedy agent has no intrinsic reward.

The distance-based agent performs on-par with the count-based agent, outperforming the ϵ-greedy agent by a large margin on longer chains. Performance on Chain domain has been consistently showing the same results during the experiment phase, only with slight but unimportant differences. The reliability of these results is undisputed as the apparent behaviour is exhibited in 16 repetitions for the same agent, with no specific seed set to initialize a random generator.

5.3 Discussion

5.3.1 Behaviour of the intrinsic distance

We will interpret and explain our results, answer the research question and justify our approach in this section. In the end we will critically evaluate our study.



(a) $n = 25$ **(b)** $n = 50$ **(c)** $n = 100$ **(d)** $n = 125$

Figure 5.4: Learning curves for the Chain domain using distance-based intrinsic reward, count-based intrinsic reward and no intrinsic reward. Results are averaged over 16 repetitions.

5.3.2 Interpretation and explanation

The introduction stated that this thesis aims at using PDFs from the C51 algorithm to produce intrinsic reward. Following the logical process one step at a time, we arrived at the intrinsic reward described in chapter 4. Results show that this method performs comparably with count-based intrinsic rewards, and vastly outperforms the non-intrinsic method. Results for the non-intrinsic agent is consistent Moerland et al. (2018), who also had the same outcome. As expected, the intrinsic reward doesn't matter too much in environments where ϵ-greedy is a good enough exploration strategy, as all agents behave equally when $n = 25$, and rather equally when $n = 50$. The big difference is for longer lengths of the chain, forcing the agent to perform 100 or even 125 actions subsequent correct actions. Here, agents using intrinsic rewards both perform comparably as well and outperforming non-intrinsic agents, as is to be expected. Following results from the Wall environment in chapter 4, the agent's state-action distributions will appear more uniformly for states longer down the chain, for which the agent has not(or rarely) visited. Upon reaching such a state, our intrinsic reward function directs the agent into exploring the unknown.

5.3.3 Answer to Research Question

The apparent behaviour is beneficial to the agent, enabling it to explore otherwise near-impossible environments. With this in mind, the author concludes that distributional reinforcement learning can induce intrinsic motivation.

5.3.4 Justification

The current approach emerged from the failure of others. Distance-based intrinsic reward takes the old concept of count-based intrinsic rewards, extending it to the new technique distributional reinforcement learning. Combining the two approaches was showing to be promising, and was stated as a future direction of research in Bellemare et al. (2017).

5.3.5 Critical evaluation of study

As we are focusing our attention on the performance of agents, it is cruical to also have a look at the tested environments. This thesis only considers one environment in the main result, albeit on different difficulty levels.

Training the agents on several environments is a good way of discovering any shortcomings of the proposed method.
No hyperparameter search has been completed, and parameters are primarily gathered from other work and what seemed sensible at the time of implementation. An exhaustive search of parameters for all three agents might provide different results. However, seeing as a non-intrinsic agent didn't manage to learn the environment at hand in Moerland et al. (2018) nor Osband et al. (2016), this is doubtful.
Our first methods based on statistical moments were discarded because of no significant improvement upon the non-intrinsic agent. However, at the time they were tested on both deterministic and stochastic environments. It was a wrong approach, as the nature of C51 can only be used for exploration purposes in deterministic environments, as mentioned in chapter 2 and Moerland et al. (2018). Chain-n is a toy environment. The agent does not need to extract meaning from an image, nor remember temporal information. Those are some of the characteristics of real-world applications. It is yet to be seen if the method is beneficial to an agent set in these types of environments. A new method for computing intrinsic reward was presented in chapter 4. This intrinsic reward was then beneficially applied to a C51 agent, making it capable of exploring in difficult environments. We want the distance based intrinsic distance behave rather similarly to the count-based one. Reason being that count-based is well proven, and is what we want to improve upon with this implementation.

Chapter 6

Conclusion

We show that probability distributions output from a categorical reinforcement learning agent act as a good source of intrinsic motivation. Results in figure 5.4 show that intrinsicly motivated agents outperform non-intrinsic agents. Our method performs on par with the count-based baseline. Surprising and new states to the agent proved to be more uniform in terms of the state-action PD distribution in a small gridworld environment.
As Bellemare et al. (2017) didn't themselves investigate how their technique could induce new exploration methods, this research is important, as it explores a hitherto uncharted territory, bridging the gap between intrinsic motivation and distributional reinforcement learning.

Looking back at my research question, we tackled the research question with a multistep process. First, the author got an overview of current research and methods on intrinsic reward and distributional RL. Note that Moerland et al. (2018) wasn't published at this point. We then shifted our view to utilize from PDs for exploration, with motivation from the field. Experiments and critical evaluation eventually led the author on the path of developing an intrinsic motivation. Further experiments refined and proved it to be working.

Further work can look at how different distributions(a Gaussian or a mixture of Gaussians, Moerland et al. (2018)) lead to contrasting behaviors. There is no evidence of mixed Monte Carlo and double Q-networks having any real impact on performance, and should be investigated further.
Extending the intrinsic reward to from-pixel environments like Atari (Mnih et al. (2013)) is also a natural path for further research.

Bibliography

Bellemare, M., Veness, J., Talvitie, E., 22–24 Jun 2014. Skip context tree switching. In: Xing, E. P., Jebara, T. (Eds.), Proceedings of the 31st International Conference on Machine Learning. Vol. 32 of Proceedings of Machine Learning Research. PMLR, Bejing, China, pp. 1458–1466. URL `http://proceedings.mlr.press/v32/bellemare14.html`

Bellemare, M. G., Dabney, W., Munos, R., 2017. A distributional perspective on reinforcement learning. CoRR abs/1707.06887. URL `http://arxiv.org/abs/1707.06887`

Bellemare, M. G., Naddaf, Y., Veness, J., Bowling, M., 2012. The arcade learning environment: An evaluation platform for general agents. CoRR abs/1207.4708. URL `http://arxiv.org/abs/1207.4708`

Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., Munos, R., 2016. Unifying count-based exploration and intrinsic motivation. CoRR abs/1606.01868. URL `http://arxiv.org/abs/1606.01868`

Bellman, R., 2013. Dynamic programming. Courier Corporation.

Bertsekas, D. P., Tsitsiklis, J. N., 1995. Neuro-dynamic programming: an overview. In: Proceedings of the 34th IEEE Conference on Decision and Control. Vol. 1. IEEE Publ. Piscataway, NJ, pp. 560–564.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W., 2016. Openai gym. CoRR abs/1606.01540. URL `http://arxiv.org/abs/1606.01540`

Hellinger, E., 1909. Neue Begründung der Theorie quadratischer Formen von unendlichvielen Veränderlichen. J. Reine Angew. Math. 136, 210–271.

Kingma, D. P., Ba, J., 2014. Adam: A method for stochastic optimization. CoRR abs/1412.6980.
URL `http://arxiv.org/abs/1412.6980`

Kolter, J. Z., Ng, A. Y., 2009. Near-bayesian exploration in polynomial time. In: Proceedings of the 26th Annual International Conference on Machine Learning. ACM, pp. 513–520.

Kullback, S., Leibler, R. A., 1951. On information and sufficiency. Ann. Math. Statistics 22, 79–86.
URL `https://doi.org/10.1214/aoms/1177729694`

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. nature 521 (7553), 436.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. A., 2013. Playing atari with deep reinforcement learning. CoRR abs/1312.5602.
URL `http://arxiv.org/abs/1312.5602`

Moerland, T. M., Broekens, J., Jonker, C. M., 2017. Efficient exploration with double uncertain value networks. CoRR abs/1711.10789.
URL `http://arxiv.org/abs/1711.10789`

Moerland, T. M., Broekens, J., Jonker, C. M., 2018. The potential of the return distribution for exploration in rl. arXiv preprint arXiv:1806.04242.

Morimura, T., Sugiyama, M., Kashima, H., Hachiya, H., Tanaka, T., 2012. Parametric return density estimation for reinforcement learning. CoRR abs/1203.3497.
URL `http://arxiv.org/abs/1203.3497`

Osband, I., Blundell, C., Pritzel, A., Roy, B. V., 2016. Deep exploration via bootstrapped DQN. CoRR abs/1602.04621.
URL `http://arxiv.org/abs/1602.04621`

Ostrovski, G., Bellemare, M. G., van den Oord, A., Munos, R., 2017. Count-based exploration with neural density models. CoRR

abs/1703.01310.
URL `http://arxiv.org/abs/1703.01310`

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D., 2016. Mastering the game of go with deep neural networks and tree search. Nature 529, 484–503.
URL `http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html`

Strehl, A. L., Littman, M. L., 2008. An analysis of model-based interval estimation for markov decision processes. Journal of Computer and System Sciences 74 (8), 1309–1331.

Sutton, R. S., Barto, A. G., et al., 1998. Reinforcement learning: An introduction. MIT press.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., Turck, F. D., Abbeel, P., 2016. #exploration: A study of count-based exploration for deep reinforcement learning. CoRR abs/1611.04717.
URL `http://arxiv.org/abs/1611.04717`

Tsitsiklis, J. N., Van Roy, B., 1997. Analysis of temporal-diffference learning with function approximation. In: Advances in neural information processing systems. pp. 1075–1081.

van Hasselt, H., 2013. Estimating the maximum expected value: An analysis of (nested) cross validation and the maximum sample average. CoRR abs/1302.7175.

van Hasselt, H., Guez, A., Silver, D., 2015. Deep reinforcement learning with double q-learning. CoRR abs/1509.06461.
URL `http://arxiv.org/abs/1509.06461`

Chapter 7

Appendix

Hyperparameters of all agents ran in experiments chapter is listed here. Sorted after experiment number. All agents run the same code. Behaviour solely is controlled by the parameters listed.

Hyperparameter/Agent	Non-intrinsic
γ	0.99
β_{MC}	0.0
β_{distance}	NA
β_{count}	NA
α	0.001
e_0	0.2
V_{MAX}	2
V_{MIN}	-1
Number of Atoms	51
Batch size	8
Memory size	5000
Update e frequency	1000
Update target frequency	400
Timestep per training loop	100

Figure 7.1: Hyperparameters for experiment one.

Hyperparameter/Agent	Non-intrinsic
γ	0.99
β_{MC}	1.0
β_{distance}	NA
β_{count}	NA
α	0.001
e_0	0.2
V_{MAX}	2
V_{MIN}	-1
Number of Atoms	51
Batch size	8
Memory size	50 000
Update e frequency	400
Update target frequency	400
Timestep per training loop	100

Figure 7.2: Hyperparameters for experiment two.

Hyperparameter/Agent	Distance-based
γ	0.99
β_{MC}	0.0
β_{distance}	0.01
β_{count}	NA
α	0.0005
λ	0.00001
χ	0.0005
e_0	0.2
V_{MAX}	2
V_{MIN}	-1
Number of Atoms	51
Batch size	128
Memory size	50 000
Update e frequency	4
Update target frequency	400
Timestep per training loop	10

Figure 7.3: Hyperparameters for experiment three.

Hyperparameter/Agent	Distance-based	Count-based	Non-intrinsic
γ	0.99	0.99	0.99
β_{MC}	0.0	0.0	0.0
β_{distance}	0.01	0.0	0.0
β_{count}	0.0	0.01	0.0
α	0.0001	0.0001	0.0001
λ	0.001	0.001	0.001
χ	0.0005	0.0005	0.0005
e_0	0.0	0.0	0.0
V_{MAX}	2	2	2
V_{MIN}	-1	-1	-1
Number of Atoms	51	51	51
Batch size	16	16	16
Memory size	500	500	500
Update e frequency	4	4	4
Update target frequency	400	400	400
Timestep per training loop	1	1	1

Figure 7.4: Hyperparameters for experiment four.

Hyperparameter/Agent	Distance-based	Count-based	Non-intrinsic
γ	0.995	0.995	0.995
β_{MC}	0.05	0.05	0.05
β_{distance}	0.01	0.0	0.0
β_{count}	0.0	0.01	0.0
α	0.0001	0.0001	0.0001
λ	0.005	0.005	0.005
χ	0.0005	0.0005	0.0005
e_0	0.2	0.2	0.2
V_{MAX}	2	2	2
V_{MIN}	-1	-1	-1
Number of Atoms	51	51	51
Batch size	32	32	32
Memory size	50 000	50 000	50 000
Update e frequency	4	4	4
Update target frequency	400	400	400
Timestep per training loop	1	1	1

Figure 7.5: Hyperparameters for experiment 5, learning the Chain-n environment.