Bistable Recurrent Cells and Belief Filtering for Q-learning in Partially Observable Markov Decision Processes

Auteur : Lambrechts, Gaspard
Promoteur(s) : Ernst, Damien
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en science des données, à finalité spécialisée
Année académique : 2020-2021
URI/URL : http://hdl.handle.net/2268.2/11474

University of Liège
School of Engineering and Computer Science
MSc in Data Science and Engineering

Master's Thesis

Bistable Recurrent Cells and Belief Filtering for Q-learning in Partially Observable Markov Decision Processes

Master's thesis carried out to obtain the degree of Master of Science in Data Science and Engineering.

Gaspard Lambrechts

supervised by

Damien Ernst

Academic year 2020 - 2021

Summary

In this master's thesis, reinforcement learning (RL) methods are used to learn (near-)optimal policies to act in several Markov decision processes (MDPs) and partially observable Markov decision processes (POMDPs). More precisely, Q-learning and recurrent Q-learning techniques are used. Some of the considered POMDPs require a high-memorisation ability in order to achieve optimal decision making. In POMDPs, RL techniques usually rely on approximating functions that take as input sequences of observations with variable length. Recurrent neural networks (RNNs) are thus a clever choice of such approximators. This work is based on the recently introduced bistable recurrent cells, which have been empirically shown to provide a significantly better long term memory than standard cells, such as the long short-term memory (LSTM) and the gated recurrent unit (GRU). These cells are named the bistable recurrent cell (BRC) and the recurrently neuromodulated BRC (nBRC). First, by importing these cells for the first time in the RL setting, it is empirically shown that they also provide a significant advantage in memory-demanding POMDPs, in comparison to LSTM and GRU. Second, the ability of the RNN to represent a belief distribution over the states of the POMDP is studied. It is achieved by evaluating the mutual information between the hidden states of the RNN and the belief filtered on the successive observations. This analysis is thus strongly anchored in the theory of information and the theory of optimal control for POMDPs. Third, as a complement to this research project, a new target update is proposed for Q-learning algorithms with target networks, for both reactive and recurrent policies. This new update speeds up learning, especially in environments with sparse rewards.

Contents

List of Algorithms

List of Tables

List of Figures

Chapter 1

Introduction

In this master's thesis, the problem of learning near-optimal policies in several Markov decision processes (MDPs) and partially observable Markov decision processes (POMDPs) is addressed by using reinforcement learning (RL). Some of these POMDPs require the RL agent to have a high memorisation ability in order to achieve optimality. MDPs model sequential decision problems where actions have to be taken based on the observation of the current state of this decision process. Each action taken in the MDP updates its state. Optimal control in MDPs consists in computing a policy, such that, when actions are taken according to this policy, the cumulative reward is maximised. Exact dynamic programming has allowed solving the control of simple MDPs. However, this method has failed to deal with more complex environments. Indeed, this approach is inefficient with large discrete state-action spaces and becomes intractable in practice when dealing with continuous state-action spaces. Moreover, it requires the model of the MDP to be known. On the contrary to optimal control and dynamic programming, where the model of the environment allows inferring an optimal policy, RL is concerned with computing a near-optimal policy in a decision process only through interaction with this environment. Moreover, RL was shown very successful in learning optimal sequences of actions, even in complex environments with nonlinear dynamics or large state and action spaces.

However, MDP is a restrictive class of decision processes that is not representative of the full range of real-world problems. Indeed, some of the decision processes for which we want to find optimal policies are better modelled by POMDPs, the latter generalising MDPs to the case where the agent only has access to a partial observation of the state of the process when making its decision. Despite requiring a model of the environment, optimal control has shown itself a powerful tool to solve POMDPs with small and discrete state and action spaces, as for MDPs. Most of these methods consist in maintaining a belief (*i.e.*, a probability distribution over the states) that is updated using the Bayes' theorem each time that a new observation is obtained. Along with this belief filtering, the optimal action for any belief can be computed by solving linear programs (LPs) iteratively. While Bayesian filtering is a well-known and complex problem when many variables are involved, computing the optimal action is not less difficult as it may require the solving of a large number of complex LPs. In addition, when dealing with continuous state, observation or action spaces, both the filtering and the planning steps become intractable in practice. Additionally, in most real-life applications, the model of POMDP is not available to filter the belief and perform planning. The latter limitations are similar to those of dynamic programming for MDPs. Once again, using RL for solving POMDPs allows overcoming some of the limitations of optimal control and has also been extensively studied.

The state-of-the-art RL methods rely on the ability of a recurrent neural network (RNN) to extract and memorise salient information from the past observations in the hidden state, in order to reconstruct the belief distribution over the states (in model-learning RL, also known

as model-based RL), the value function (in value-based RL), or to directly produce a policy (in policy-based RL). Using RNNs in model-free methods have several advantages in comparison to other methods such as utile suffix memory, finite policy graphs or predictive state representations (Spaan, 2012), notably because they theoretically allow to represent infinite time-dependencies while having a finite representation. However, several works showed the inability of current RNN-based RL techniques to discover optimal sequences of actions in simple but memory-demanding environments, where the performance of the policy is directly limited by the memorisation ability of the underlying RNN (Bakker, 2001; Wierstra, Foerster, Peters, & Schmidhuber, 2007; Zhang, McCarthy, Finn, Levine, & Abbeel, 2016). Aside from that, the bistable recurrent cells were introduced recently (Vecoven, Ernst, & Drion, 2020). These cells have been shown to exhibit neural bistability, a mechanism that allows outperforming standard recurrent cells, namely LSTM and GRU, on several supervised learning benchmarks that require a long-lasting memory. This master's thesis is the first work to benchmark these bistable cells in the RL setting.

The first two research questions of this master's thesis can be summarised as follows: Can the bistable recurrent cells provide an advantage in comparison to other cells, when used in model-free RL algorithms for POMDPs? Can the information contained in the hidden states of the RNNs be related to the belief as defined in control theory? Finally, by observing that the convergence of the estimators was very slow on environments with sparse rewards, it was concluded that it came from the low target update rate in the Q-learning algorithms with target networks, such as the algorithms used in this work. It led to the third and last research question of this work that writes as follows: How to understand the target network usage in Q-learning algorithms and how to replace its periodic update with an objective criterion?

The starting point of this research project was to formalise and reimplement the Deep Recurrent Q-Network algorithm (DRQN) as proposed by Hausknecht and Stone (2015) that extends Q-learning to recurrent estimators. Concurrently, the literature on optimal control and model-free RL for POMDPs has been reviewed. In addition, the functions and approximators that are manipulated by these algorithms were formalised in the very early stages of this work. Next, the objective was to reproduce the results from the literature on the different POMDPs that are considered in this work, using the DRQN algorithm and the LSTM network. Afterwards, the three research question were studied. Regarding the performance of the bistable cells, this work studies the ability of these cells to extend the class of recurrent policies that can be learned in RL. More precisely, it is proposed to study empirically the performance of the recurrent policies when BRC or nBRC are used as RNN instead of LSTM or GRU. Specifically, this work focuses on environments that have discrete action spaces and that are moderatly to very memory-demanding. Regarding the information contained in the hidden states, this research question allows studying the internal representations learned in the RNNs, instead of only their performance. Indeed, since the RNNs are responsible for learning the value function of a trajectory — that is defined to be the value function of the belief filtered from this trajectory, it is assumed that the hidden states should contain information about this belief. In practice, the question will be to discover whether there is a high mutual information between the hidden states of the RNN and the belief, when the RNN is trained to learn the Q-values, and whether this can be necessary to reach optimality in some environments. Answering in the affirmative to this question would motivate future works in which particular attention is given to the belief-filtering ability of the RNN architectures for RL. Regarding the target network, the question is raised by the observed inefficiency and instability of the Q-learning algorithms in environments with sparse rewards. Despite this observation, the first two research questions are still answered in the Q-learning setting, without modifications to the algorithm, to better fit in the literature of model-free RL for POMDPs. However, the Q-learning algorithms that we consider (*i.e.*, with target network) are adapted to the Q-iteration setting under the name Online Fitted Q-Iteration (OFQI). This framework, by providing better theoretical foundations, motivates some changes that notably improve the

target update strategy. The Q-learning algorithms, corrected to fit in the Q-iteration algorithms, are shown to provide a greater speed of learning, especially in environments with sparse rewards.

In Chapter 2, RL is described and the state-of-the-art RL techniques for solving POMDPs are reviewed. In Chapter 3, the required background on POMDPs, RNNs and information theory is formally introduced. In Chapter 4, recurrent parametric Q-learning is described. The functions that are approximated, along with their estimators, are thus formally defined and the learning algorithm is detailed. Afterwards, Chapter 5 describes the experimental protocol and the environments. The resulting performances of the different agents are displayed, to answer the first research question. In Chapter 6, the second question is addressed by analysing the outputs and recurrent states of the different estimators, along with their relationship with the belief. Finally, the Q-iteration algorithm proposed as an alternative to the Q-learning algorithms with target networks is described in Chapter 7. Chapter 8 concludes and discusses the constributions and limitations of this work.

Chapter 2

Literature review

In the first section of this chapter, the RL field is described, along with the state-of-the-art methods in model-learning RL, value-based RL and policy-based RL. These state-of-the-art methods nevertheless only tackle MDPs in which the state is fully observable. In the second section, the state-of-the-art methods in RL for POMDPs are presented, along with some of their limitations.

2.1 Reinforcement learning

Nowadays, using reinforcement learning (RL) for solving decision-making problems is a common approach in many domains: robotics (Kober, Bagnell, & Peters, 2013), power network control (Ernst, Glavic, & Wehenkel, 2004; Vlachogiannis & Hatziargyriou, 2004; Ernst, Glavic, Capitanescu, & Wehenkel, 2008), recommendation engines (Zheng et al., 2018), pricing strategies (Kutschinski, Uthmann, & Polani, 2003), resource management (Barrett, Howley, & Duggan, 2013), etc. RL is concerned with the learning of optimal policies from interactions with an unknown environment. The objective is to maximise — in some sense — the rewards collected by the agent in the long run, by selecting a sequence of actions according to this policy (Wiering & Van Otterlo, 2012).

As stated in the introduction (Chapter 1), RL is concerned with computing an optimal policy in a decision process only through interaction with this environment. Note that the interaction samples used to optimize the policy can be generated by the policy to be optimized (on-policy) or by a distinct policy (off-policy) such as an exploration policy or a past version of the policy being optimized. In model-learning RL (also known as model-based RL), a model of the underlying environment is learned. Afterwards, optimal control theory can be used to compute an optimal policy on this approximate model (Moerland, Broekens, & Jonker, 2020). In value-based RL, the objective is to learn the so-called value function without trying to learn the environment model (Sutton & Barto, 2018). Roughly speaking, the value function represents the quality of taking an action in a state of the environment. From this value function, we can thus select the action that is expected optimal. These methods have been widely applied and used to solve many complex tasks such as playing the ATARI games (Mnih et al., 2015). Value-based RL nevertheless suffers from a notable problem, it does not scale well to problems with continuous action spaces. This is not the case with policy-based RL where an optimal policy is directly inferred from the samples collected in the environment. The state-of-the-art policy-based methods usually infer optimal parameters of a parametrized policy by performing gradient ascent, a technique known as policy gradient. These methods have also been widely deployed (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017). Value-based and policy-based approaches, including policy gradient, are grouped under the name model-free. In summary, RL succeeded in learning from experience how to behave in environments with large (or sometimes infinite) state and action spaces without

prior information. However, as mentioned in the introduction, most of these successes boil down to solving MDPs while some of the interesting problems that the RL aims to solve have to be formalised as POMDPs instead of MDPs. POMDPs generalise MDPs to the case where only a partial observation of the state of the process is available.

As for MDPs, RL in POMDPs is aimed at learning an optimal policy without requiring a model of the POMDP. As stated in the introduction (Chapter 1), the computations involved by the theory of optimal control for POMDPs may be intractable when dealing with large problems. In this context, sampling-based techniques such as RL are good candidates for solving these issues. Few RL approaches have been developed until now; they are described in more detail in the following section. They mainly focus on very simple tasks or on tasks where the environment is nearly fully observable. So far, they have also been rather poorly analysed, theoretically, in model-free RL.

2.2 Reinforcement learning in partially observable environments

The problem of optimal control in POMDPs has been extensively studied. This problem consists in acting optimally in an environment assuming its model (dynamics, reward, and observation distributions) is fully known by the agent (Kaelbling, Littman, & Cassandra, 1998). The usual strategy for optimal control is divided into two parts. First, maintaining a belief distribution over the underlying state of the POMDP given the observations. Second, an action maximising the expected cumulative reward that will be gathered is planned, for every possible belief. For POMDPs with discrete state-action spaces and observation spaces, updating the belief is performed by the straightforward application of the Bayes' rule (Bayes' theorem). In addition, computing the optimal action can be achieved by an iterative computation implying solving multiple LPs. These two steps are however known to be intractable in practice. On the other side, the RL strategy gets rid of the possible intractability of these two steps, filtering and planning, by automatically learning from samples.

The RL setting for POMDPs enables an agent to learn from interaction with the environment, having access neither to the underlying model nor the underlying states. First, model-learning approaches were proposed such as (Shankar, Dwivedy, & Guha, 2016) and (Gupta, Davidson, Levine, Sukthankar, & Malik, 2017). Both applied RL to learn, offline, a model of the environment on the one hand and learn a planning strategy based on this model on the other hand. These approaches were nevertheless hybrid in some sense as the state of the environment was assumed to be observable during the learning process. Perhaps the most notable work carried out using model-learning RL in POMDPs was performed by Karkus, Hsu, and Lee (2017). They generalised previous works and introduced the QMDP-Net, a neural network architecture learned in an RL fashion from which actions can be planned without requiring the observation of the underlying state during learning. Indeed, end-to-end training is performed as in model-free RL, except that the underlying computation architecture is constrained to correspond to the belief update, as in the two previous works.

In model-free RL, the idea of combining an RNN (more precisely an LSTM) with temporal difference learning to learn the value functions of non-Markovian environments was made popular by Bakker (2001). In so doing, an approximation of the value function that depended on the complete history of actions and observations was built. This idea was subsequently applied to the off-policy setting by extending the successful Deep Q-Network (DQN) (Mnih et al., 2015) to POMDPs with the Deep Recurrent Q-Network (DRQN) (Hausknecht & Stone, 2015). Using RNN was also applied in policy-based RL as in (Wierstra et al., 2007). In this work, the so-called policy gradients were extended to POMDPs and a stochastic policy, depending on the history of observations and actions, was learned for solving problems requiring long-lasting memory. The latter work is limited to the on-policy setting in opposition to the Recurrent

Deterministic Policy Gradient (RDPG) (Heess, Hunt, Lillicrap, & Silver, 2015) that extends the Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015). In more recent works, particle filtering was combined with policy gradients to explicitly represent the internal belief distribution (Igl, Zintgraf, Le, Wood, & Whiteson, 2018).

As an alternative to RNN-based model-free RL for POMDPs, Zhang et al. (2016) propose to incorporate a memory space that extends the observation space. The agent can then explicitly decide to store or replace an observation in its memory. This choice is called a memory action. As a consequence, the agent is trained to take memory actions in addition to the actions taken in the environment. It can be noted that these methods nonetheless still use a RNN, even if it does not carry the burden of memorisation alone.

However, the algorithms available in the literature only tackle either simple or low-memory-demanding environments. Furthermore, off-policy policy-gradient approaches such as the RDPG introduce a training bias, the importance of which grows with the time-dependencies to be learned. Finally, model-free algorithms do not have strong convergence proofs, as in MDPs (Hutter, 2014; Majeed & Hutter, 2018).

Chapter 3

Background

In this chapter, POMDPs are formally defined, along with the belief and the value function. In the second section, optimal control theory for POMDPs is briefly introduced. Then, the particle filtering algorithm is introduced. Next, RNN approximators are described, and the RNNs of interest are formalised, namely the LSTM, GRU, BRC and nBRC. Finally, the estimation of mutual information between two continuous random variables is reviewed in the last section.

3.1 Partially observable Markov decision process

Partially observable Markov decision processes (POMDPs) model sequential decision making problems under partial observability (Spaan, 2012; Kaelbling et al., 1998). Formally, a POMDP is a 7-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, p_0)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space and $\mathcal{O}$ is the observation space. The probability distribution function p_0 gives to all state $s_0 \in \mathcal{S}$ its probability $p_0(\mathbf{s}_0)$ to be the initial state. A transition probability distribution function T gives the probability $T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ to update the state $\mathbf{s}_t \in \mathcal{S}$ of the process to the state $\mathbf{s}_{t+1} \in \mathcal{S}$ when taking the action $\mathbf{a}_t \in \mathcal{A}$. A reward function R gives the bounded reward $r_t = R(\mathbf{s}_t, \mathbf{a}_t)$ obtained when taking action $\mathbf{a}_t \in \mathcal{A}$ in state $\mathbf{s}_t \in \mathcal{S}$. Finally, the observation probability distribution function O gives the probability $O(\mathbf{o}_t \mid \mathbf{s}_t)$ to observe $\mathbf{o}_t \in \mathcal{O}$ when the process is in state $\mathbf{s}_t \in \mathcal{S}$. The execution of such a decision process is illustrated in Figure 3.1 by considering the sequence of actions chosen *a priori* (*i.e.*, their conditional dependencies are not represented).

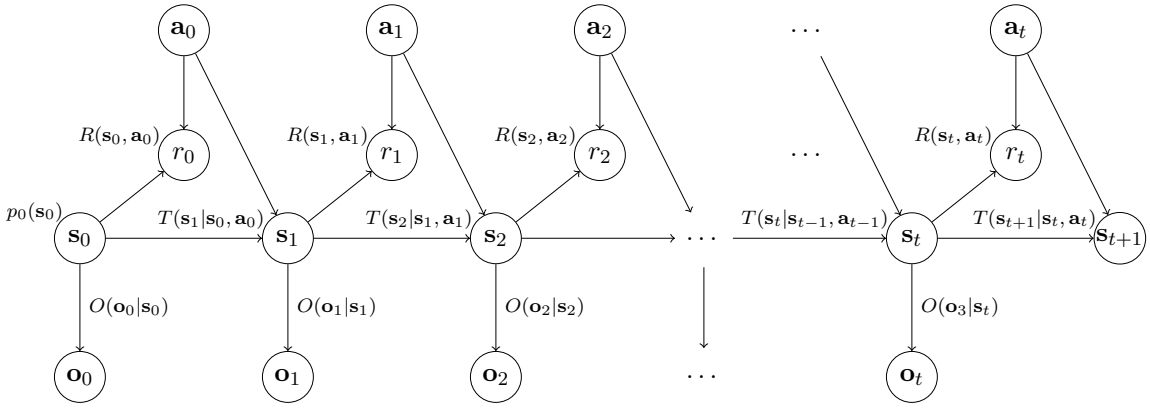


Figure 3.1: Bayesian representation of a POMDP execution

After taking t actions in the POMDP, the sequence of observations and actions is called the trajectory $\tau_{0:t} = (\mathbf{o}_0, \mathbf{a}_0, \ldots, \mathbf{o}_{t-1}, \mathbf{a}_{t-1}, \mathbf{o}_t) \in \mathcal{T}_{0:t}$, where $\mathcal{T}_{0:t}$ is the set of trajectories of length t

that can be generated in the POMDP. The belief b_t at a time t represents the knowledge of the agent about the POMDP state. Formally, $b_t(\mathbf{s}_t \mid \tau_{0:t})$ is the probability to be in state $\mathbf{s}_t$ given the trajectory $\tau_{0:t} \in \mathcal{T}_{0:t}$. The belief $b_t \in \mathfrak{B}$ is thus the probability distribution over the states conditioned on the trajectory $\tau_{0:t}$, and $\mathfrak{B} = \{b_t(\cdot \mid \tau_{0:t}) \mid \tau_{0:t} \in \mathcal{T}_{0:t}, t \in \mathbb{N}\}$ is the set of all possible beliefs (*i.e.*, the beliefs that can be derived from any valid trajectory from the POMDP). Using Bayes' rule, the initial belief writes:

$$b_0(\mathbf{s}_0 \mid \tau_{0:0}) = \Pr(\mathbf{s}_0 \mid \mathbf{o}_0) = \frac{p_0(\mathbf{s}_0)O(\mathbf{o}_0 \mid \mathbf{s}_0)}{\int_{\mathcal{S}} p_0(\mathbf{s}'_0)O(\mathbf{o}_0 \mid \mathbf{s}'_0)d\mathbf{s}'_0} \ . \tag{3.1}$$

and for $t > 0$, the belief is given by:

$$b_t(\mathbf{s}_t \mid \tau_{0:t}) = \frac{O(\mathbf{o}_t \mid \mathbf{s}_t)\Pr(\mathbf{s}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1})}{\Pr(\mathbf{o}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1})} \tag{3.2}$$

$$\Pr(\mathbf{s}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1}) = \int_{\mathcal{S}} T(\mathbf{s}_t \mid \mathbf{s}_{t-1}, \mathbf{a}_{t-1}) b_{t-1}(\mathbf{s}_{t-1} \mid \tau_{0:t-1}) d\mathbf{s}_{t-1} \tag{3.3}$$

$$\Pr(\mathbf{o}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1}) = \int_{\mathcal{S}} O(\mathbf{o}_t \mid \mathbf{s}_t) \left(\int_{\mathcal{S}} T(\mathbf{s}_t \mid \mathbf{s}_{t-1}, \mathbf{a}_{t-1}) b_{t-1}(\mathbf{s}_{t-1} \mid \tau_{0:t-1}) d\mathbf{s}_{t-1} \right) d\mathbf{s}_t \ . \tag{3.4}$$

Indeed, for $t > 0$, from the Bayes' rule conditioned on $(\mathbf{a}_{t-1}, \tau_{0,t-1})$, we have:

$$\begin{aligned}
b_t(\mathbf{s}_t \mid \tau_{0:t}) &= \Pr(\mathbf{s}_t \mid \tau_{0:t}) && (3.5)\\
&= \Pr(\mathbf{s}_t \mid \mathbf{o}_t, \mathbf{a}_{t-1}, \tau_{0:t-1}) && (3.6)\\
&= \frac{\Pr(\mathbf{o}_t \mid \mathbf{s}_t, \mathbf{a}_{t-1}, \tau_{0:t-1})\Pr(\mathbf{s}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1})}{\Pr(\mathbf{o}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1})} && (3.7)\\
&= \frac{\Pr(\mathbf{o}_t \mid \mathbf{s}_t)\Pr(\mathbf{s}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1})}{\Pr(\mathbf{o}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1})} && (3.8)\\
&= \frac{O(\mathbf{o}_t \mid \mathbf{s}_t)\Pr(\mathbf{s}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1})}{\Pr(\mathbf{o}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1})} && (3.9)
\end{aligned}$$

The probability at the numerator writes:

$$\begin{aligned}
\Pr(\mathbf{s}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1}) &= \int_{\mathcal{S}} \Pr(\mathbf{s}_t, \mathbf{s}_{t-1} \mid \mathbf{a}_{t-1}, \tau_{0:t-1}) d\mathbf{s}_{t-1} && (3.10)\\
&= \int_{\mathcal{S}} \Pr(\mathbf{s}_t \mid \mathbf{s}_{t-1}, \mathbf{a}_{t-1}, \tau_{0:t-1})\Pr(\mathbf{s}_{t-1} \mid \mathbf{a}_{t-1}, \tau_{0:t-1}) d\mathbf{s}_{t-1} && (3.11)\\
&= \int_{\mathcal{S}} \Pr(\mathbf{s}_t \mid \mathbf{s}_{t-1}, \mathbf{a}_{t-1})\Pr(\mathbf{s}_{t-1} \mid \tau_{0:t-1}) d\mathbf{s}_{t-1} && (3.12)\\
&= \int_{\mathcal{S}} T(\mathbf{s}_t \mid \mathbf{s}_{t-1}, \mathbf{a}_{t-1}) b_{t-1}(\mathbf{s}_{t-1} \mid \tau_{0:t-1}) d\mathbf{s}_{t-1} && (3.13)
\end{aligned}$$

where the second probability of (3.12) is obtained from (3.11) by considering that the action taken at time $t-1$ is conditionally independent to the state given the observed trajectory until time $t-1$. This hypothesis is reasonable since the policy should only have access to the observed trajectory $\tau_{0:t-1}$. The probability at the denominator writes:

$$\begin{aligned}
\Pr(\mathbf{o}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1}) &= \int_{\mathcal{S}} \Pr(\mathbf{o}_t, \mathbf{s}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1}) d\mathbf{s}_t && (3.14)\\
&= \int_{\mathcal{S}} \Pr(\mathbf{o}_t \mid \mathbf{s}_t, \mathbf{a}_{t-1}, \tau_{0:t-1})\Pr(\mathbf{s}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1}) d\mathbf{s}_t && (3.15)\\
&= \int_{\mathcal{S}} \Pr(\mathbf{o}_t \mid \mathbf{s}_t)\Pr(\mathbf{s}_t \mid \mathbf{a}_{t-1}, \tau_{0:t-1}) d\mathbf{s}_t && (3.16)\\
&= \int_{\mathcal{S}} O(\mathbf{o}_t \mid \mathbf{s}_t) \left(\int_{\mathcal{S}} T(\mathbf{s}_t \mid \mathbf{s}_{t-1}, \mathbf{a}_{t-1}) b_{t-1}(\mathbf{s}_{t-1} \mid \tau_{0:t-1}) d\mathbf{s}_{t-1} \right) d\mathbf{s}_t \ . && (3.17)
\end{aligned}$$

As a consequence, equations (3.2), (3.3) and (3.4) provide a way to update iteratively the belief b_t once observing new information $(\mathbf{a}_t, \mathbf{o}_{t+1})$ through a forward pass such that:

$$b_{t+1} = \text{forward}(b_t; \mathbf{a}_t, \mathbf{o}_{t+1}) \ . \tag{3.18}$$

and this, provided that T, O, p_0, and thus $\mathcal{S}, \mathcal{A}, \mathcal{O}$, are known.

A deterministic stationary policy $\pi \in \Pi$, where Π is the set of all such policies, is a function mapping each belief $b_t \in \mathfrak{B}$ to an action $\mathbf{a}_t \in \mathcal{A}$. Note that, in the following, b_t is used to denote $b_t(\cdot \mid \tau_{0:t})$ without distinction. For a given discount factor $\gamma \in [0, 1[$, a policy $\pi \in \Pi$ is characterised by its value function V^π computed as follows:

$$V^\pi(b_t) = \lim_{T \to \infty} \sum_{k=t}^{T} \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim b_t(\cdot|\tau_{0:t}) \\ \mathbf{s}_{k+1} \sim T(\cdot|\mathbf{s}_k, \mathbf{a}_k) \\ \mathbf{o}_{k+1} \sim O(\cdot|\mathbf{s}_{k+1})}} \left\{ \gamma^{k-t} r_k \right\}, \quad \forall b_t \in \mathfrak{B} \tag{3.19}$$

where $\mathbf{a}_k = \pi(b_k)$ is the action selected by the policy π at time k and $r_k = R(\mathbf{s}_k, \mathbf{a}_k)$ is the reward computed at time k. In addition, the belief is updated as $b_{k+1} = \text{forward}(b_k; \mathbf{a}_k, \mathbf{o}_{k+1})$.

An optimal policy $\pi^* \in \Pi$ is a policy whose value function is maximal given any belief $b_t \in \mathfrak{B}$ at any time t:

$$\pi^* \in \arg\max_{\pi \in \Pi} V^\pi(b_t), \quad \forall b_t \in \mathfrak{B} \ . \tag{3.20}$$

The Q-function of a POMDP is defined as the expected cumulative reward observed when playing an action $\mathbf{a}_t$ given a belief b_t, followed by an optimal policy:

$$Q(b_t, \mathbf{a}_t) = \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim b_t(\cdot|\tau_{0:t}) \\ \mathbf{s}_{t+1} \sim T(\cdot|\mathbf{s}_t, \mathbf{a}_t) \\ \mathbf{o}_{t+1} \sim O(\cdot|\mathbf{s}_{t+1})}} \left\{ r_t + \gamma V^{\pi^*}(b_{t+1}) \right\}, \quad \forall b_t \in \mathfrak{B}, \forall \mathbf{a}_t \in \mathcal{A} \ . \tag{3.21}$$

where $b_{t+1} = \text{forward}(b_t; \mathbf{a}_t, \mathbf{o}_{t+1})$. Finally, given the definition (3.19) and (3.21), we have that a policy outputting the action resulting in the maximal Q-value in each belief is optimal:

$$V(b_t) = V^{\pi^*}(b_t) = \max_{\mathbf{a}_t \in \mathcal{A}} Q(b_t, \mathbf{a}_t), \quad \forall b_t \in \mathfrak{B} \ . \tag{3.22}$$

Let v be a real-valued function. This function respects the Bellman equation if it respects the following equality:

$$v(b_t) = \max_{\mathbf{a}_t \in \mathcal{A}} \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim b_t(\cdot|\tau_{0:t}) \\ \mathbf{s}_{t+1} \sim T(\cdot|\mathbf{s}_t, \mathbf{a}_t) \\ \mathbf{o}_{t+1} \sim O(\cdot|\mathbf{s}_{t+1})}} \{ r_t + \gamma v(\text{forward}(b_t; \mathbf{a}_t, \mathbf{o}_{t+1})) \}, \ \forall b_t \in \mathfrak{B} \ . \tag{3.23}$$

In addition, the Bellman operator H is defined as follows:

$$(Hv)(b_t) = \max_{\mathbf{a}_t \in \mathcal{A}} \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim b_t(\cdot|\tau_{0:t}) \\ \mathbf{s}_{t+1} \sim T(\cdot|\mathbf{s}_t, \mathbf{a}_t) \\ \mathbf{o}_{t+1} \sim O(\cdot|\mathbf{s}_{t+1})}} \{ r_t + \gamma v(\text{forward}(b_t; \mathbf{a}_t, \mathbf{o}_{t+1})) \} \ . \tag{3.24}$$

The operator H can be proved to be a contraction mapping such that the Bellman equation $v = Hv$ has a unique solution, the fixed-point of this contraction mapping. Combining (3.21) and (3.22), it can be seen that this unique solution of the Bellman equation is the optimal value function $V = V^{\pi^*}$.

This recursive definition has lead to the development of the value iteration algorithm. This algorithm consists in computing successive functions $V_1, \dots, V_n$ such that $V_i(b_t)$ is the optimal cumulative reward obtained in belief b_t with a i-step horizon (*i.e.*, only i actions and reward are left to take). The computation of V_i is performed by applying the Bellman operator to V_{i-1}. The value iteration algorithm is proved to converge in norm to the unique fixed point of the Bellman equation, that is the optimal value function. The computations of the successive functions V_i is discussed in the next section, as well as the method for computing an optimal action given V_n.

3.2 Optimal control in partially observable Markov decision processes

The problem of control in POMDP consists in finding a policy mapping any belief to an optimal action. This section focuses on POMDPs with discrete observation spaces. The traditional approach of optimal control is to maintain the belief as dictated by (3.18) and to derive a policy from the value function. Let us first consider POMDPs with discrete state spaces. In this case, the belief $b_t = b_t(\cdot \mid \tau_{0:t})$ can be represented by a vector giving the conditional probability $\Pr(\mathbf{s}_t \mid \tau_{0:t})$ for each state $\mathbf{s}_t \in \mathcal{S}$ of the POMDP. The dimension of the vector is thus $|\mathcal{S}|$.

It has been shown in (Smallwood & Sondik, 1973) that the value function for the n-step horizon, called V_n, is a piecewise linear convex function (PWLC) on the belief space. This allows representing V_n by a finite number of vectors, called α-vectors, representing the different linear pieces. V_n is then modelled by the maximum over these linear functions, making it a PWLC function. Any PWLC function can indeed be represented in this way. We have:

$$V_n(b_t) = \max_{k \in \{1,\dots,K_n\}} \alpha_{n,k} \cdot b_t \tag{3.25}$$

where b_t is represented by a vector, $\alpha_{n,k} = \begin{pmatrix} \alpha_{n,k}^{(1)} & \dots & \alpha_{n,k}^{(|\mathcal{S}|)} \end{pmatrix}$ is an α-vector and $x \cdot y$ represent the dot product between two vectors x and y. In addition, K_n denotes the number of linear pieces necessary to represent V_n. In order to approximate V, we can thus iteratively compute $V_1, \dots, V_n$ with n large, by computing a set of α-vectors for each n. Conveniently, for every n, each of these K_n α-vectors are associated with a certain action. Given a belief, the policy will thus be simply derived from the value function by choosing the action associated with the α-vector maximising the value at this belief.

The iterative computation of $V_1, \dots, V_n$ goes as follows: The horizon $n = 1$ value is given by $V_1(b_t) = \max_{\mathbf{a}_t \in \mathcal{A}} Q_1(b_t, \mathbf{a}_t)$. And for all $\mathbf{a}_t \in \mathcal{A}$, we have $Q_1(b_t, \mathbf{a}_t) = R(b_t, \mathbf{a}_t) = \mathop{\mathbb{E}}_{\mathbf{s}_t \sim b_t} \{R(\mathbf{s}_t, \mathbf{a}_t)\} = \sum_{\mathbf{s}_t \in \mathcal{S}} b_t(\mathbf{s}_t) R(\mathbf{s}_t, \mathbf{a}_t)$. It can be verified that V_1 has the form of equation (3.25) with $K_1 = |\mathcal{A}|$. Precisely, V_1 is defined by the following set of α-vector:

$$\alpha_{1,k} = \begin{pmatrix} R(\mathbf{s}_t^{(1)}, \mathbf{a}^{(k)}) & \dots & R(\mathbf{s}_t^{(|\mathcal{S}|)}, \mathbf{a}^{(k)}) \end{pmatrix}, \; k = 1, \dots, K_1 \; . \tag{3.26}$$

where $\mathbf{a}^{(k)}$ denotes the k^{th} action and $\mathbf{s}^{(i)}$ the i^{th} state. It can be noted that some of these α-vectors could be dominated on the whole belief space (in the case where some actions are never worth playing for maximising its one-step return). In this case, they can be pruned from the representation of V_1. The same can be done in the following iterations. For $n > 1$, V_n is computed from V_{n-1} by:

$$V_n(b_t) = \max_{\mathbf{a}_t \in \mathcal{A}} \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim b_t \\ \mathbf{s}_{t+1} \sim T(\cdot \mid \mathbf{s}_t, \mathbf{a}_t) \\ \mathbf{o}_{t+1} \sim O(\cdot \mid \mathbf{s}_t)}} \{r_t + \gamma V_{n-1}(b_{t+1})\} \tag{3.27}$$

$$= \max_{\mathbf{a}_t \in \mathcal{A}} R(b_t, \mathbf{a}_t) + \gamma \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim b_t \\ \mathbf{s}_{t+1} \sim T(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)}} \left\{ \sum_{\mathbf{o}_{t+1} \in \mathcal{O}} O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1}) V_{n-1}(b_{t+1}) \right\} \tag{3.28}$$

$$= \max_{\mathbf{a}_t \in \mathcal{A}} R(b_t, \mathbf{a}_t) + \gamma \sum_{\mathbf{o}_{t+1} \in \mathcal{O}} \Pr(\mathbf{o}_{t+1} \mid b_t, \mathbf{a}_t) V_{n-1}(b_{t+1}) \tag{3.29}$$

where $b_{t+1} = \text{forward}(b_t; \mathbf{a}_t, \mathbf{o}_{t+1})$ and

$$\Pr(\mathbf{o}_{t+1} \mid b_t, \mathbf{a}_t) = \sum_{\mathbf{s}_t \in \mathcal{S}} \sum_{\mathbf{s}_{t+1} \in \mathcal{S}} b_t(\mathbf{s}_t) T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1}) \; . \tag{3.30}$$

The proof of the piecewise linearity and convexity of V_n can be found in Appendix B. In practice, computing V_n given V_{n-1} thus consists in creating a new set of α-vectors by adding, for all $\mathbf{a}_t$, for all $\mathbf{o}_t$, all the transformed α-vector from V_{n-1}. This construction is however known to have exponential growth in the number of α-vectors, and despite pruning the dominated vectors is possible, it requires solving a linear program for every possible vector.

The complexity limitation of exact value iteration in POMDP has been addressed by point-based value iteration methods (Pineau, Gordon, Thrun, et al., 2003). The idea is to maintain a finite set of belief points $b^{(0)}, \ldots, b^{(q)}$ and to maintain α-vectors at these points. The exponential growth in the number of α-vectors is completely alleviated by the fact that the new vectors do not need to be kept in a set. Only the maximum vector at each belief point b_t is kept. In other words, we refuse to create more pieces in our PWLC value function, but we only keep the vectors that are dominant at the chosen belief points $b^{(0)}, \ldots, b^{(q)}$. The only remaining pruning step is to remove α-vectors that would be shared between several belief points. This technique has an additional benefit because we do not need to learn the value function of beliefs that are not likely to be reached. Indeed, the set of belief points can be carefully chosen.

As far as the continuous state space POMDPs are concerned, the belief can no longer be represented by vectors but should be represented by functions over the state space. Consequently, the belief space is continuous and infinite-dimensional. In (Porta, Spaan, & Vlassis, 2004), it is shown that the optimal finite-horizon value function is also PWLC, analogously to the discrete case, over this infinite-dimensional belief state. The value function is also supported by α-functions that are analogous to the α-vectors in the discrete case. The dot product is replaced by an integral over the state space of the product of the belief function with the α-function. In addition, generalisations of the point-based value iteration algorithms exist for POMDPs with continuous domains.

Both in the discrete and the continuous case, the n-step optimal policy is easily defined from the α-represented value function V_n. Indeed, each α-vector, or α-function, is associated with an action $\mathbf{a}_t \in \mathcal{A}$. And the policy consists in mapping the belief b_t to the action $\mathbf{a}_t$ associated with the dominant α-vector, or α-function, for this belief b_t. This whole approach of optimal control in POMDPs can only be conducted when a model of the POMDP is available. In this work, the opposite case is addressed. The value function is learned from samples, without knowing any of the dynamics of the environment.

3.3 Particle Filtering

As explained in the introduction (Chapter 1), Bayes filtering is a complex problem that becomes intractable in certain POMDPs. In particular, POMDPs with continuous state space require to integrate over the state space. Furthermore, in these environments, the belief should be represented by a function over a continuous domain instead of a finite-dimensional vector. Such arbitrary beliefs cannot be represented in a digital computer.

To overcome these two difficulties, the particle filtering algorithm proposes to represent an approximation of the belief by a finite set of samples from the belief distribution. In other words,

we represent $b_t \colon \mathcal{S} \to \mathbb{R}$ by the following set of N samples:

$$S_t = \{\mathbf{s}_t^n\}_{n=1}^N \tag{3.31}$$

with $\mathbf{s}_t^n \in \mathcal{S}$, $n = 1, \ldots, N$ being realisations of the probability density function $b_t(\cdot \mid \tau_{0:t})$.

Particle filtering is a procedure that allows to maintain a set of samples such that they follow the successive distributions $b_0, \ldots, b_T$. The set is thus updated each time that a new action $\mathbf{a}_{t-1}$ is taken and a new observation $\mathbf{o}_t$ is observed. Although this procedure does not require to evaluate expression (3.18), it is necessary to be able to sample from the initial probability distribution $p_0(\cdot)$ and from the conditional transition probability distribution $T(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)$, and to be able to evaluate the conditional observation probability $O(\mathbf{o}_t \mid \mathbf{s}_t)$. This process, illustrated in Algorithm 1, guarantees that the successive sets $S_0, \ldots, \mathcal{S}_T$ have (weighted) samples following the probability distribution $b_0, \ldots, b_T$ defined by (3.18).

Algorithm 1: Particle filtering

Input: A POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, p_0)$
Input: A trajectory $\tau_{0:T} = (\mathbf{o}_0, \mathbf{a}_0, , \ldots, \mathbf{o}_{T-1}, \mathbf{a}_{T-1}, \mathbf{o}_T)$
Input: The number of samples N
Sample $\mathbf{s}_0^0, \ldots, \mathbf{s}_0^N \sim p_0$
$\eta \leftarrow 0$
for $n = 1, \ldots, N$ **do**
    $w_0^n \leftarrow O(\mathbf{o}_0 \mid \mathbf{s}_0^n)$
    $\eta \leftarrow \eta + w_0^n$
for $n = 1, \ldots, N$ **do**
    $w_0^n \leftarrow w_0^n / \eta$
$S_0 = \{(\mathbf{s}_0^n, w_0^n)\}_{n=1}^N$
for $t = 1, \ldots, T$ **do**
    $\eta \leftarrow 0$
    for $n = 1, \ldots, N$ **do**
        Sample l from discrete distribution w_{t-1}
        Sample $\mathbf{s}_t^n \sim T(\cdot \mid \mathbf{s}_{t-1}^l, \mathbf{a}_{t-1})$
        $w_t^n \leftarrow O(\mathbf{o}_t \mid \mathbf{s}_t^n)$
        $\eta \leftarrow \eta + w_t^n$
    for $n = 1, \ldots, N$ **do**
        $w_t^n \leftarrow w_t^n / \eta$
    $S_t = \{(\mathbf{s}_t^n, w_t^n)\}_{n=1}^N$

Algorithm 1 starts from N samples from the initial distribution p_0. These samples are initially weighted by their likelihood $O(\mathbf{o}_0 \mid \mathbf{s}_0^n)$. Then, we have three steps that are repeated at each time step. First, the samples are resampled according to their weights. Then, given the action, the samples are updated by sampling from $T(\cdot \mid \mathbf{s}_t^n, \mathbf{a}_t)$. Finally, these new samples are weighted by their likelihood $O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1}^n)$ given the new observation $\mathbf{o}_{t+1}$, as for the initial samples. As stated above, this method ensures that the (weighted) samples follow the distribution of the successive beliefs.

3.4 Recurrent neural networks

In this work, we are interested in approximating the Q-function, as defined in Section 3.1. The Q-function are functions of $b_t \in \mathfrak{B}$ and $\mathbf{a}_t \in \mathcal{A}$. However, $b_t = b_t(\cdot \mid \tau_{0:t})$ is a function of the

complete observed trajectory $\tau_{0:t} \in \mathcal{T}_{0:t}$. As will be explained in Section 4.2, computing the belief explicitly is avoided, such that the approximation of Q will directly take $\tau_{0:t}$ as input. This input being of variable length, it makes RNN a good choice of function approximator for Q.

Neural networks denote a very general class of universal parametric function approximator whose parameters are typically learned by gradient descent to minimise a certain loss. Generally, neural networks are used to approximate a function $f\colon \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ is a p-dimensional space and $\mathcal{Y}$ is a q-dimensional space. However, it happens that the input, the output or both are sequential. Given a set $\mathcal{X}$, let $\mathcal{S}(\mathcal{X})$ denote the set of all sequences of elements from $\mathcal{X}$, that is $\mathcal{S}(\mathcal{X}) = \cup_{t=0}^{\infty} \mathcal{X}^{t+1}$. In this work, we are interested in modelling $f\colon \mathcal{S}(\mathcal{X}) \to \mathcal{Y}$. And the sequences $\mathbf{x}_{0:T} \in \mathcal{S}(\mathcal{X})$ that are considered have variable lengths $T+1$, which appeals for using recurrent neural network as function approximator. Indeed, these allow mapping a variable-length sequential input to an output, by maintaining a hidden state (or recurrent state). In comparison to other neural networks that consume sequences, they have the advantage of updating their output efficiently when the sequence grows and not being limited to fixed-length sequences. Consequently, they can model arbitrarily long time-dependencies, or at least, as long as their architecture allows remembering.

In the context of approximating the Q-functions using RNN approximators, we have $\mathcal{X} = (\{\mathbf{0}\} \cup \mathcal{A}) \times \mathcal{O}$ and $\mathcal{Y} = \mathbb{R}^{|\mathcal{A}|}$. More precisely, $\mathbf{x}_0 = (\mathbf{0}, \mathbf{o}_0)$, and $\mathbf{x}_t = (\mathbf{a}_{t-1}, \mathbf{o}_t),\ t > 0$. The approximator is thus fed with a trajectory as input and outputs approximations of the Q-value for each of the $|\mathcal{A}|$ actions.

Formally, a recurrent neural network is defined by its hidden state update and its output function:

$$\mathbf{h}_t = \phi_\theta(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{3.32}$$

$$\mathbf{y}_t = \psi_\theta(\mathbf{h}_t) \tag{3.33}$$

where the update function ϕ_θ and the output function ψ_θ are parametrised functions. In other words, ϕ and ψ define the architecture of the recurrent cell, and are functions of $\theta \in \Theta$, where Θ is the domain of θ, in addition to $\mathbf{x}_t, \mathbf{h}_{t-1}$ and $\mathbf{h}_t$. The parameter space Θ is usually chosen to be $\Theta = \mathbb{R}^{d_\theta}$, where d_θ denotes the number of parameters. Together with ψ_θ and ϕ_θ, it defines the hypothesis space. The parameters θ are be optimised such that the cell returns the best outputs, with respect to certain a loss. This optimisation is typically achieved by gradient descent. As a consequence, ϕ_θ and ψ_θ are required to be differentiable with respect to θ, for any fixed input.

In this work, where we are interested in modelling $f\colon \mathcal{S}(\mathcal{X}) \to \mathcal{Y}$, the output of the neural network is $\mathbf{y}_T$ for $\mathbf{x}_{0:T} \in \mathcal{S}(\mathcal{X})$. In other words, $f\colon \mathbf{x}_{0:T} \mapsto \mathbf{y}_T$ is recursively defined by (3.32) and (3.33). In the following, only the recurrent part ϕ_θ of the architecture is detailed, since ψ_θ is identical for all recurrent architectures and is chosen to be:

$$\mathbf{y}_t = \mathbf{h}_t\ . \tag{3.34}$$

The most straightforward recurrent neural network is the Elman network that writes:

$$\mathbf{h}_t = \sigma(W_{hx}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1}) \tag{3.35}$$

with $\theta = (W_{hx}, W_{hh})$ and σ denoting the sigmoid activation function. However, such a simple architecture has several drawbacks. Indeed, if the activation function and the input are neglected, the successive hidden states $h_1, \ldots, h_T$ will be successively multiplied by the same matrix W_{hh}, which would lead the hidden state to converge towards an eigenvector of the matrix W_{hh} after a few time steps. But without normalisation, or with such a squishing activation function, this hidden state will also probably explode, or vanish, very quickly. In addition, the gradient could also vanish or explode in the backward pass, which will prevent to learn large time dependencies and will introduce instability. A solution to this problem is to use gating.

Gating consists in introducing pass-through or additive paths between the successive hidden states (Goodfellow, Bengio, Courville, & Bengio, 2016). This way, the successive hidden states are not always squished or stretched by the same matrix, but some components of the hidden state can be partially taken from the previous hidden state, or can even be reset to zero. One of the most simple and efficient gated RNN is the gated recurrent unit (GRU) introduced by Chung, Gulcehre, Cho, and Bengio (2014). Its update writes as follows:

$$\mathbf{z}_t = \sigma(W_{zx}\mathbf{x}_t + W_{zh}\mathbf{h}_{t-1}) \tag{3.36}$$
$$\mathbf{r}_t = \sigma(W_{rx}\mathbf{x}_t + W_{rh}\mathbf{h}_{t-1}) \tag{3.37}$$
$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tanh(W_{hx}\mathbf{x}_t + \mathbf{r}_t \odot W_{hh}\mathbf{h}_{t-1}) \tag{3.38}$$

We can see that the forget gate, that outputs $\mathbf{z}_t$, allows taking a mixture of the updated and the old hidden state. Since $\mathbf{z}_t$ is a vector, the mixture proportion can be different for each element of the hidden state. Similarly, we have the reset gate $\mathbf{r}_t$ that is able to reset to zero the legacy from the previous hidden state, or to only take a portion of it. The values $\mathbf{z}_t$ and $\mathbf{r}_t$ are computed from the last hidden state $\mathbf{h}_{t-1}$ and the current input $\mathbf{x}_t$ using new parameters W_{zx}, W_{zh}, W_{rx} and W_{rh}.

Historically, the GRU was introduced to simplify the long short term memory (LSTM) that was introduced by Hochreiter and Schmidhuber (1997). It has an additional gate, and has two recurrent states: the hidden state $\mathbf{h}_t$ and the cell state $\mathbf{c}_t$. Its update writes:

$$\mathbf{f}_t = \sigma(W_{fx}\mathbf{x}_t + W_{fh}\mathbf{h}_{t-1}) \tag{3.39}$$
$$\mathbf{i}_t = \sigma(W_{ix}\mathbf{x}_t + W_{ih}\mathbf{h}_{t-1}) \tag{3.40}$$
$$\mathbf{o}_t = \sigma(W_{ox}\mathbf{x}_t + W_{oh}\mathbf{h}_{t-1}) \tag{3.41}$$
$$\tilde{\mathbf{c}}_t = \sigma(W_{cx}\mathbf{x}_t + W_{ch}\mathbf{h}_{t-1}) \tag{3.42}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \tag{3.43}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \sigma(\mathbf{c}_t) \tag{3.44}$$

More recently, a new update was proposed by Vecoven et al. (2020) under the name bistable recurrent cell (BRC). This update has been carefully chosen such that certain choices of parameters allow neurons to exhibit neural bistability, a mechanism the neurons of the human brain use for storing information at the cellular level on the long term. This property allows the BRC and nBRC to outperform the state-of-the-art on different supervised learning problems requiring high-memorisation ability. By introducing the possibility of having natural equilibrium around certain values, the recurrent cell are able to keep information over a longer period of time. According to the parameters, the input and the last hidden state, bistability can be present or not for each neuron of the cell, and when it is present, two equilibrium points are present, which explains the name *bistability*. The update writes:

$$\mathbf{c}_t = \sigma(W_{cx}\mathbf{x}_t + w_c \odot \mathbf{h}_{t-1}) \tag{3.45}$$
$$\mathbf{a}_t = 1 + \tanh(W_{ax}\mathbf{x}_t + w_a \odot \mathbf{h}_{t-1}) \tag{3.46}$$
$$\mathbf{h}_t = \mathbf{c}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{c}_t) \odot \tanh(W_{hx} + \mathbf{a}_t \odot \mathbf{h}_{t-1}) \tag{3.47}$$

In addition, the cellular memory constraint can be relaxed by allowing the hidden state of a neuron to influence the bistability of another neuron. This new cell was called the recurrently neuromodulated BRC (nBRC) and its update is:

$$\mathbf{c}_t = \sigma(W_{cx}\mathbf{x}_t + W_{ch}\mathbf{h}_{t-1}) \tag{3.48}$$
$$\mathbf{a}_t = 1 + \tanh(W_{ax}\mathbf{x}_t + W_{ah}\mathbf{h}_{t-1}) \tag{3.49}$$
$$\mathbf{h}_t = \mathbf{c}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{c}_t) \odot \tanh(W_{hx} + \mathbf{a}_t \odot \mathbf{h}_{t-1}) \tag{3.50}$$

It can be noted that this update is very similar to that of the GRU.

The biases have been omitted in all updates for readability. But, to gain a little representation ability, we can add biases as parameters inside all the activation functions. Finally, as is sometimes done with RNNs, it is chosen to make $\mathbf{h}_{-1}$ (and $\mathbf{c}_{-1}$ for LSTM), the initial hidden state, a trainable parameter too. Note that the initial input $\mathbf{x}_0$ has been chosen to be indexed at $t = 0$ such that the initial hidden states are indexed at $t = -1$. It was done to harmonise the indexing convention of the RNN literature with the POMDP literature in which the sequences of observations start at $t = 0$. Indeed, as mentioned above, $\mathbf{x}_0 = (\mathbf{0}, \mathbf{o}_0)$ and $\mathbf{x}_t = (\mathbf{a}_{t-1}, \mathbf{o}_t)$ for $t > 0$.

As for standard neural networks, where the backpropagation is very natural, we can make backpropagation in RNNs very straightforward by unrolling the computation graph. Indeed, we can consider the pass in the cell at each time step to be successive layers of a deep neural network. The only difference with deep neural networks is that the weights are shared between each unrolled time step. This is handled by summing the gradients obtained for each parameter on all time steps.

3.5 Mutual information estimation

On the one hand, as suggested by the theory of optimal control and the belief definition of Section 3.1, the optimal strategy for evaluating the Q-function is to maintain an estimation of the belief, the latter being updated each time that a new action is taken and a new observation is received, according to (3.18). The belief b_t indeed forms a sufficient statistic from the complete trajectory $\tau_{0:t}$ to correctly estimate the value of the different actions. On the other hand, as explained in Section 3.4, the successive observations and actions taken in the POMDP will be fed to the RNN recurrently, such that it outputs the Q-value of this trajectory for all possible actions. As a consequence, it will be interesting to see if these hidden states reconstruct the belief in some way. More precisely, the research question of this work is to study the mutual information between the hidden state of a trained RNN having experienced a certain trajectory, and the belief filtered from this same trajectory. In this case, we have a random variable $\tau_{0:t} \in \mathcal{T}_{0:t}$, and we have two deterministic functions: $f\colon \tau_{0:t} \mapsto \mathbf{h}_t$ that represents the hidden states generated by the RNN, and $g\colon \tau_{0:t} \mapsto b_t$ that represents the belief resulting from the trajectory. We are thus interested in the mutual information between the random variable $X = \mathbf{h}_t$ and the random variable $Y = b_t$. For now, it is assumed that the state space of the POMDP is discrete, such that b_t can be represented by a vector.

In contrast to other dependence testing estimators, the mutual information estimator has the advantage of being strongly rooted in the information theory. In addition, the mutual information is theoretically able to measure any kind of dependency between the variables. In particular, the mutual information is invariant to smooth and uniquely invertible mappings of the random variables (Kraskov, Stögbauer, & Grassberger, 2004). The mutual information between two jointly continuous variable X and Y writes:

$$I(X;Y) = \int_{\mathcal{X}} \int_{\mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p_X(x)p_Y(y)} dxdy \tag{3.51}$$

where $\mathcal{X}$ and $\mathcal{Y}$ are the support of the random variables X and Y respectively, p is the joint probability density function of X and Y, and p_X and p_Y are the marginal probability density function of X and Y respectively. The mutual information has the following properties:

$$I(X;Y) = h(X) - h(X \mid Y) \tag{3.52}$$

$$= h(Y) - h(Y \mid X) \tag{3.53}$$

$$= h(X) + h(Y) - h(X, Y) \tag{3.54}$$
$$= I(Y; X) \tag{3.55}$$

where h denotes the differential entropy in the case of continuous variables. The differential entropy is defined as follows:

$$h(X) = -\int_{\mathcal{X}} f(x) \log f(x) dx \ . \tag{3.56}$$

These properties imply that the mutual information between two independent random variables is zero.

Estimating the mutual information between the random variable X and Y thus consists in estimating $I(X;Y)$ from a dataset $\{(x_i, y_i)\}_{i=1}^N$ where the samples (x_i, y_i) are drawn from the joint probability distribution p. The most straightforward approach for estimating the mutual information between two continuous random variables X and Y consists in partitioning the supports $\mathcal{X}$ and $\mathcal{Y}$ such that the variables are discretised. Then, the mutual information is estimated by:

$$I_{\text{binned}}(X;Y) = \sum_i \sum_j f(i,j) \log \frac{f(i,j)}{f_X(i) f_Y(j)} \tag{3.57}$$

where $f(i,j) = n(i,j)/N$, $f_X(i) = n_X(i)/N$, $f_Y(j) = n_Y(j)/N$ and $n_X(i)$ is the number of samples of X falling into the i^{th} bin of X, $n_Y(j)$ is the number of samples of Y falling to the j^{th} bin of Y, and $n(i,j)$ is the number of points in the intersection of these bins. In practice, this method has been improved by considering adaptive bin sizes. Nevertheless, this simple method still suffers from systematic errors coming from binning and from the errors in the estimation of the log probability ratios from the log frequency ratios.

In (Kraskov et al., 2004), the authors propose a new approach for estimating the mutual information based on k-nearest neighbour (k-NN) statistics. At the time, there already exists an extensive literature for estimating the entropy using such statistics (Kozachenko & Leonenko, 1987). Instead of estimating $p(x,y)$, $p(x)$ and $p(y)$ from $f(i,j)$, $f(i)$, $f(j)$, using predefined bins, these estimators estimate $p(x,y)$, $p(x)$ and $p(y)$ by constructing dynamic boxes for each point. These dynamic boxes are chosen to be the smallest boxes centred around the considered point such that there is exactly k nearest neighbours in it. Starting from k-NN estimations of $h(X)$, $h(Y)$, $h(X,Y)$ we can estimate the mutual information by (3.54). However, the errors made in these three entropy estimates would presumably not cancel and the authors propose an alternative. By defining a certain joint norm in the $X \times Y$ space, they succeed in making the biases in the estimations $\hat{h}(X)$, $\hat{h}(Y)$ and $\hat{h}(X,Y)$ cancel out approximately thanks to the shared length scales in the three estimates. This estimator is chosen for the estimation of the mutual information in this work and is referred to as the KSG estimator. However, as shown in (Gao, Ver Steeg, & Galstyan, 2015), the KSG estimator requires a number of samples that scales exponentially with the true mutual information. Alternative methods have been proposed in (Gao, Ver Steeg, & Galstyan, 2015; Gao, Steeg, & Galstyan, 2015) to refine the estimator and provide better estimations in the case of strongly correlated variables. Nonetheless, the resulting estimators relied on an arbitrarily tuned threshold parameter and had no theoretical performance guarantees (McAllester & Stratos, 2020). Since then, it has been proved by McAllester and Stratos (2020) that the exponential sample complexity is inherent to the measurement of mutual information no matter what estimator is used.

In view of the difficulty of estimating the mutual information, especially between variables having a high mutual information, it is decided to develop two simple but interpretable methods that allow to controlling the results obtained by the KSG estimator. Both these methods use a metric to evaluate the quality of the mapping that can be constructed between X and Y. The first

method uses a multilayer perceptron (MLP) approximator trained to map X to Y. Since the target Y is the space of belief on the discrete states, the cross-entropy loss is used for this problem, trying to reconstruct the belief $Y = b_t$ from the hidden state $X = \mathbf{h}_t$. The second one uses a classification random forest (RF) approximator. For this estimator, the accuracy when trying to predict the most likely state from the belief is reported. Indeed, this metric will be particularly interpretable when dealing with environments in which the belief is indicating a single state with probability 1, such as in the Deterministic T-Maze (see Section A.1).

As far as continuous-state POMDPs are concerned, there is an additional difficulty of mutual information estimation that arises from the problem of representing the belief. In this case, as explained in Section 3.3, the belief can no longer be represented by a finite-dimensional vector but should be represented by a function defined over the state space. This function, in the general case, can be arbitrarily complex, and it is thus impossible to represent this function exactly in a digital computer. However, we can select a hypothesis space of functions that are parametrised by a finite-dimensional real-valued vector $\omega \in \Omega$, where Ω denotes the parameter space. Since parametrised functions have been chosen, this parameter space Ω defines the hypothesis space as follows: $\{b_t(\cdot;\omega) \mid \omega \in \Omega\}$ where $b_t(\cdot;\omega)$ is the parametrised belief. Such a hypothesis space allows to identify the belief with a finite-dimensional vector ω, despite $b_t(\cdot;\omega)$ still represents a function defined over the entire continuous state space.

As a result of this representation of b_t for continuous state POMDPs, instead of the mutual information between $\mathbf{h}_t$ and b_t, the mutual information between $\mathbf{h}_t$ and the parameter ω that parametrises the approximation $b_t(\cdot;\omega)$ of $b_t(\cdot)$ is used. In this work, the hypothesis space is simply chosen to be the set of piecewise constant functions on a uniform discretisation of the domain. As a consequence, the parameter ω boil down to the set of constant values of the approximation for every bin of the discretisation. Now, ω still has to be estimated. For POMDPs with continuous state spaces, the belief b_t is estimated with the particle filtering algorithm, as explained in Section 3.3. It is thus represented by a set of N samples from the distribution $b_t(\cdot)$. The approximation $b_t(\cdot;\omega)$ is then built using the frequencies of these samples in each bin of the discretisation.

Chapter 4

Recurrent parametric Q-learning

In this chapter, the problem statement is formalised. The latter consists in selecting a parametrised policy, from a hypothesis space, that maximises the γ-discounted rewards in trajectories sampled from the POMDP. To do so, the policies are derived from a parametric approximation $\mathcal{Q}_\theta$ of the $\mathcal{Q}$-function. This recurrent approximation is learned by recurrent parametric Q-learning as explained in the second section. Finally, the detailed DRQN learning algorithm is introduced in the third section.

4.1 Problem statement

Let us give the POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, p_0)$ as defined in Section 3.1 with discrete action space $\mathcal{A}$. Let us also give the discount factor $\gamma \in [0, 1[$. We consider the problem of approximating the optimal policy $\pi^* \in \Pi$ by a policy $\pi_\theta \in \Pi$ parametrised by the real vector $\theta \in \Theta \subseteq \mathbb{R}^{d_\Theta}$, where d_Θ is the dimension of the vector. The expected return sampled from the initial state, is given by:

$$J(\pi_\theta) = \lim_{T\to\infty} \sum_{t=0}^{T} \mathop{\mathbb{E}}_{\substack{\mathbf{s}_0\sim p_0(\cdot)\\ \mathbf{o}_t\sim O(\cdot|\mathbf{s}_t)\\ \mathbf{s}_{t+1}\sim T(\cdot|\mathbf{s}_t,\mathbf{a}_t),}} \left\{\gamma^t r_t\right\} \tag{4.1}$$

with $\mathbf{a}_t = \pi(b_t(\cdot \mid \tau_{0:t}))$ and $r_t = R(\mathbf{s}_t, \mathbf{a}_t)$. From the definition of V^{π_θ} (3.19), we have:

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{\substack{\mathbf{s}_0\sim p_0(\cdot)\\ \mathbf{o}_0\sim O(\cdot|\mathbf{s}_0)}} \left\{V^{\pi_\theta}(b_0(\cdot \mid \tau_{0:0}))\right\} \tag{4.2}$$

$$= \mathop{\mathbb{E}}_{\substack{\mathbf{s}_0\sim p_0(\cdot)\\ \mathbf{o}_0\sim O(\cdot|\mathbf{s}_0)}} \left\{V^{\pi_\theta}(b_0(\cdot \mid \mathbf{o}_0))\right\} \tag{4.3}$$

Formally, we want to find $\theta^* \in \Theta$ such that:

$$\theta^* \in \arg\max_{\theta\in\Theta} J(\pi_\theta) \tag{4.4}$$

$$\Leftrightarrow \theta^* \in \arg\max_{\theta\in\Theta} \mathop{\mathbb{E}}_{\substack{\mathbf{s}_0\sim p_0(\cdot)\\ \mathbf{o}_0\sim O(\cdot|\mathbf{s}_0)}} \left\{V^{\pi_\theta}(b_0(\cdot \mid \mathbf{o}_0))\right\} \tag{4.5}$$

4.2 Parametric Q-learning in POMDPs

In this work, deterministic policies are derived from Q-functions as suggested by (3.22), this work is thus anchored in the value-based RL. The Q-function are estimated from samples in the process through Q-learning, which has the advantage of being off-policy. In the context of RL, because

of the unavailability of the model, it is not possible to estimate the belief. We thus propose to define a variant of the Q-function defined in (3.21), which are referred to as $\mathcal{Q}$-function, defined over the space of trajectories rather than over the space of beliefs. In addition, we introduce the same variant for the value function $\mathcal{V}\colon \cup_{t=0}^{\infty} \mathcal{T}_{0:t} \to \mathbb{R}$ instead of $V\colon \mathfrak{B} \to \mathbb{R}$. This value function is defined as the optimal discounted return that can be obtained given that we experienced a trajectory $\tau_{0:t}$. Since this value will be completely determined by the probability of being in each state given $\tau_{0:t}$, and that this probability is given by the belief $b_t(\cdot \mid \tau_{0:t})$, it is straightforward that:

$$\mathcal{V}(\tau_{0:t}) = \mathcal{V}^{\pi^*}(\tau_{0:t}) = V^{\pi^*}(b_t(\cdot \mid \tau_{0:t})) = V(b_t(\cdot \mid \tau_{0:t}))\ . \tag{4.6}$$

The $\mathcal{Q}$-function can then be defined by:

$$\mathcal{Q}(\tau_{0:t}, \mathbf{a}_t) = \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim b_t(\cdot|\tau_{0:t}) \\ \mathbf{s}_{t+1} \sim T(\cdot|\mathbf{s}_t, \mathbf{a}_t) \\ \mathbf{o}_{t+1} \sim O(\cdot|\mathbf{s}_t)}} \{r_t + \gamma \mathcal{V}(\tau_{0:t+1})\}, \quad \forall \tau_{0:t} \in \mathcal{T}_{0:t}, \forall \mathbf{a}_t \in \mathcal{A}\ . \tag{4.7}$$

where $\tau_{0:t+1} = \tau_{0:t} \cup (\mathbf{a}_t, \mathbf{o}_{t+1})$. In addition, it is easy to see that:

$$\mathcal{V}(\tau_{0:t}) = \max_{\mathbf{a}_t \in \mathcal{A}} \mathcal{Q}(\tau_{0:t}, \mathbf{a}_t)\ . \tag{4.8}$$

The $\mathcal{Q}$-function is thus the composition of equation (3.18) applied over the whole trajectory $\tau_{0:t}$ and equations (3.21) and (3.22). From the above definitions, it can also be written:

$$\mathcal{Q}(\tau_{0:t}, \mathbf{a}_t) = Q(b_t(\cdot \mid \tau_{0:t}), \mathbf{a}_t), \forall \tau_{0:t} \in \mathcal{T}_{0:t}, \forall \mathbf{a}_t \in \mathcal{A}\ . \tag{4.9}$$

4.3 Learning algorithm

Based on a set of trajectories $\mathcal{B}$, the Q-learning algorithm can be adapted to $\mathcal{Q}$-functions, as recursively defined by equations (4.7) and (4.8). Let us give a hypothesis space of real parameters $\Theta \subseteq \mathbb{R}^{d_\Theta}$, where d_Θ is the dimension of the space. Let us give $\mathcal{Q}$-functions $\mathcal{Q}_\theta$ parametrised by $\theta \in \Theta$. The parameters θ can be updated from the reward r_t and the partial trajectory $\tau_{0:t+1} \in \mathcal{T}_{0:t+1}$ of $t+1$ transitions, where $\tau_{0:t+1} = \tau_{0:t} \cup (\mathbf{a}_t, \mathbf{o}_{t+1})$, as follows:

$$\theta \leftarrow \theta + \alpha \delta(r_t, \tau_{0:t+1}) \frac{d\mathcal{Q}_\theta}{d\theta}(\tau_{0:t}, \mathbf{a}_t) \tag{4.10}$$

$$\delta(r_t, \tau_{0:t+1}) = r_t + \gamma \max_{\mathbf{a} \in \mathcal{A}} \{\mathcal{Q}_\theta(\tau_{0:t+1}, \mathbf{a})\} - \mathcal{Q}_\theta(\tau_{0:t}, \mathbf{a}_t) \tag{4.11}$$

where α is the learning rate and $\delta(r_t, \tau_{0:t+1})$ is the temporal difference. The partial trajectory $\tau_{0:t+1}$ is obtained by truncating a trajectory $\tau_{0:T} \in \mathcal{B}$ after time step t such that $0 \leq t < T$. Note that this update corresponds to performing one gradient step on the loss $L\left(Q_\theta(\tau_{0:t}, \mathbf{a}_t), y_t\right) = (Q_\theta(\tau_{0:t}, \mathbf{a}_t) - y_t)^2$ with respect to θ, where:

$$y_t = r_t + \gamma \max_{\mathbf{a} \in \mathcal{A}} \{\mathcal{Q}_\theta(\tau_{0:t+1}, \mathbf{a})\} \tag{4.12}$$

is the temporal difference target, considered constant with respect to θ.

In practice, the Q-learning algorithm with parametric functions has been adapted in (Mnih et al., 2015) to show a better convergence rate, in the case of reactive policies. These adaptations mainly consist in sampling the trajectories online while applying the update for learning a policy. A replay buffer of trajectories is used and the gradient is evaluated on a batch of trajectories simultaneously at each iteration. In addition, the function approximator is differentiated through auto-differentiation and the parameters θ are updated with the Adam algorithm (Kingma & Ba, 2014). Finally, the use of a target Q-function $Q_{\theta'}$ to compute the temporal difference target — and only updating the target every C episodes by $\theta' \leftarrow \theta$ — eases the convergence of Q-learning.

These ideas have been adapted to $\mathcal{Q}$-function in (Hausknecht & Stone, 2015) with the Deep Recurrent Q Network (DRQN) using backpropagation through time.

In the DRQN algorithm, the weights θ of the parametric $\mathcal{Q}$-function $\mathcal{Q}_\theta$ are initialised randomly, and the replay buffer is filled with totally random trajectories. Then, E episodes of Q-learning are performed using mini-batches of B transitions drawn uniformly from the replay buffer $\mathcal{B}$. In addition, a new trajectory is generated at each episode and replaces the oldest trajectory in the replay buffer. A time horizon T is used to prevent generating infinite trajectories. These trajectory are generated using the ε-greedy policy that choose an action according to the exploration policy with probability ε, and $\arg\max_{\mathbf{a}} \{\mathcal{Q}_\theta(\tau_{0:t}, a)\}$ otherwise. The exploration policy $\mathcal{E}(\mathcal{A})$ depends on the environment. Finally, a target $\mathcal{Q}$-function $\mathcal{Q}_{\theta'}$ is updated by $\theta' \leftarrow \theta$ every C episodes only. Using the target network $\mathcal{Q}_{\theta'}$ allows the targets of the loss to be more stable, easing the convergence of the algorithm, as for the DQN. A pseudocode of the training procedure is provided in Algorithm 2. It can be noted that the parametric estimator $\mathcal{Q}_\theta$ of the $\mathcal{Q}$-function is fed with the trajectory $\tau_{0:t} = (\mathbf{o}_0, \mathbf{a}_0, \mathbf{o}_1, \mathbf{a}_1, \dots, \mathbf{o}_t)$ instead of the sequence of observations $(\mathbf{o}_0, \mathbf{o}_1, \dots, \mathbf{o}_t)$ as is done in (Hausknecht & Stone, 2015). Indeed, for the parametric $\mathcal{Q}$-function to correctly estimate the belief-action Q-function, this estimator has to receive the actions and observations, as suggested by the belief filtering (3.18). Otherwise, the policy would be suboptimal in the general case. This adjustment was initially proposed by Zhu, Li, Poupart, and Miao (2017).

It is noteworthy that the parametric $\mathcal{Q}$-functions $\mathcal{Q}_\theta$ must be able to process inputs of variable length $\tau_{0:t} \in \mathcal{T}_{0:t}, \forall t \in \mathbb{N}$. As explained in Section 3.4, RNN is a class of neural networks that consume the input sequentially by producing a recurrent state that is passed recurrently at each time step. Doing so, we have a network unit, called a cell, that exhibits memory through the recurrent state. These neural networks are thus ideal candidates for defining parametric functions which may depend on sequences of an indefinite length. In the following, these are thus used as function approximators for the $\mathcal{Q}$-functions as proposed in (Hausknecht & Stone, 2015). The advantage of using RNNs, in comparison to other methods, is that they allow, in theory, to model dependencies between inputs and outputs that are separated by arbitrary long time intervals. However, in practice, we are limited by the memorisation ability of the RNN architecture that conditions its representation power. As explained in the introduction, the representation power and memorisation ability are compared for different RNN architecture in the following chapter.

The first architectures of interest are the Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) and the Gated Recurrent Units (GRU) (Chung et al., 2014). Both are standard networks that have been applied to many supervised learning problems. We compare these two RNN with the recently introduced Bistable Recurrent Cell (BRC) and recurrently neuromodulated BRC (nBRC) (Vecoven et al., 2020). The full architecture of the recurrent parametric estimator is represented in Figure 4.1. The output of the first RNN is passed as input to the second. Then, the output of the second RNN is passed to two linear layers separated by a ReLU activation. Finally, it can be noted the reward is passed as input the neural network, which is just an implementation choice.

Algorithm 2: DRQN - Epsilon greedy training

Initialise weights θ randomly
Fill the replay buffer $\mathcal{B}$ of capacity N with transitions generated from the exploration policy $\mathcal{E}(\mathcal{A})$
for $e = 0, \ldots, E-1$ **do**
    `// Evaluate policy`
    if $e \equiv 0 \pmod{M}$ **then**
        Evaluate expected discounted reward from R trajectories with the fully greedy policy
    `// Update target network`
    if $e \equiv 0 \pmod{C}$ **then**
        $\theta' \leftarrow \theta$
    `// Generate new trajectory`
    Reset environment to $\mathbf{s}_0$ and observe $\mathbf{o}_0$
    Set $\tau_{0:0} \leftarrow (\mathbf{0}, 0, \mathbf{o}_0)$
    for $t = 0, \ldots, T-1$ **do**
        Select $\mathbf{a}_t \sim \mathcal{E}(\mathcal{A})$ with probability ε, otherwise select $\mathbf{a}_t \leftarrow \arg\max_{\mathbf{a} \in \mathcal{A}} \{\mathcal{Q}_\theta(\tau_{0:t}, \mathbf{a})\}$
        Take action a_t and observe r_t and $\mathbf{o}_{t+1}$
        Set $\tau_{0:t+1} \leftarrow \tau_{0:t} \cup (\mathbf{a}_t, r_t, \mathbf{o}_{t+1})$
        Replace oldest transition in replay buffer $\mathcal{B}$ by $(\tau_{0:t}, \mathbf{a}_t, r_t, \mathbf{o}_{t+1}, \tau_{0:t+1})$
        if $\mathbf{o}_{t+1}$ *is terminal* **then**
            break
    `// Optimize recurrent Q-networks`
    Sample B transitions $(\tau^b_{0:t}, \mathbf{a}^b_t, r^b_t, \mathbf{o}^b_{t+1}, \tau^b_{0:t+1})$ uniformly from the replay buffer $\mathcal{B}$
    Compute targets $y^b = \begin{cases} r^b_t + \gamma \max_{\mathbf{a} \in \mathcal{A}} \left\{\mathcal{Q}_{\theta'}(\tau^b_{0:t+1}, \mathbf{a})\right\} & \text{if } \mathbf{o}^b_{t+1} \text{ is not terminal} \\ r^b_t & \text{otherwise} \end{cases}$
    Compute loss $L = \sum_{b=0}^{B-1} \left(y^b - \mathcal{Q}_\theta(\tau^b_{0:t}, \mathbf{a}^b_t)\right)^2$
    Optimize θ to minimise L using Adam optimizer with learning rate α

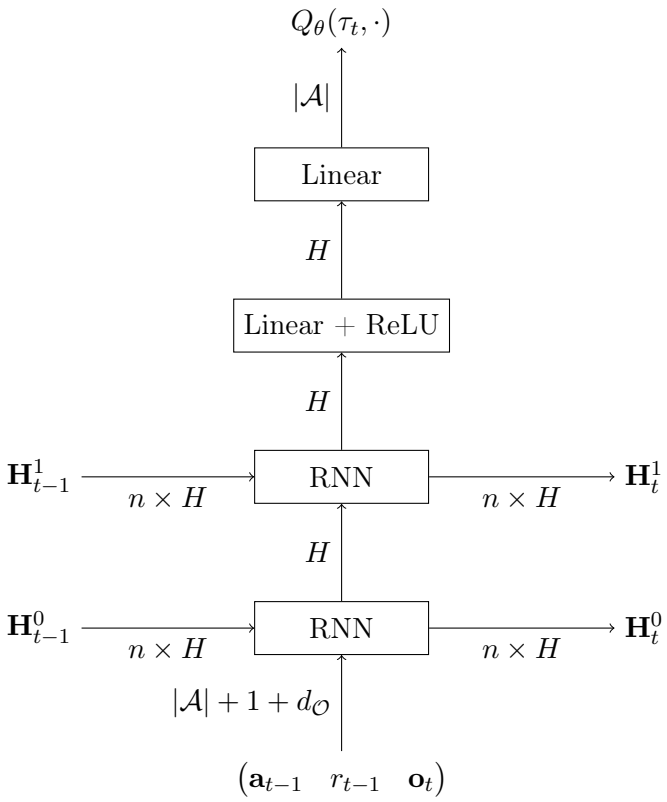


Figure 4.1: Architecture of the recurrent parametric estimator of the $\mathcal{Q}$-function - $d_\mathcal{O}$ is the dimensionality of the observation space $\mathcal{O}$, H is the size of the recurrent states and hidden layers, n denotes the number of recurrent states, being 1 for all RNNs except the LSTM having $n = 2$. $\mathbf{H}^l_t = \left(\mathbf{h}^l_t, \mathbf{c}^l_t\right)$ for the LSTM and $\mathbf{H}^l_t = \left(\mathbf{h}^l_t\right)$ for the other RNNs.

Chapter 5

Experiments

In this chapter, the first research question of this master's thesis is addressed. The experimental protocol is defined, introducing the performance metrics that are used and the class of environments that are considered, namely environments with terminal states. Afterwards, the considered POMDPs and MDPs are briefly described and the performances are reported when using the different RNNs as approximator of the $\mathcal{Q}$-function.

5.1 Experimental Protocol

The DRQN training (Algorithm 2) is applied on the different POMDPs using each RNN architecture. The performance of the DRQN agent are assessed by the cumulative discounted reward of the fully greedy policy. The latter, in the context of value-based RL, denotes the policy derived from the learned $\mathcal{Q}$-function as follows:

$$\pi_\theta(\tau_{0:t}) = \arg\max_{\mathbf{a}_t \in \mathcal{A}} \mathcal{Q}_\theta(\tau_{0:t}, \mathbf{a}_t) \ . \tag{5.1}$$

If $\mathcal{Q}_\theta$ perfectly approximates $\mathcal{Q}$, then this policy is optimal, according to the Bellman optimality equation (4.8). Nevertheless, we do not need $\mathcal{Q}_\theta$ to be perfect to get an optimal policy.

5.1.1 Metrics

The performances of the policies are measured every $M = 10$ episodes by computing the average discounted cumulative reward over $R = 50$ Monte Carlo rollouts. For every POMDP, and every RNN, 3 training sessions will be performed for the T-Maze environments, 4 for the other environments. The learning curves of these different training sessions are averaged. In addition, they are smoothed by an exponential moving average with a very short half-life, to improve readability without losing eventual brief overperformances and underperformances that arise over the course of learning.

For the environments where the optimal discounted reward or the optimal policy is known (*i.e.*, the Deterministic and Stochastic T-Mazes), the following quantities are studied: the speed of convergence towards the optimal policy (if the latter was reached), the stability of the learning process, and the highest performance reached throughout training, relatively to optimality.

5.1.2 Class of environments

In the experiments, POMDPs with terminal states are be considered. A state $\mathbf{s} \in \mathcal{S}$ is terminal if, and only if:

$$\begin{cases} T(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{if } \mathbf{s}' = \mathbf{s} \\ 0 & \text{otherwise} \end{cases} & (5.2) \\ R(\mathbf{s}, \mathbf{a}) = 0 & (5.3) \end{cases}$$

for all $\mathbf{s}' \in \mathcal{S}$ and for all $\mathbf{a} \in \mathcal{A}$. It is obvious that the Q-value of a trajectory that reaches a terminal states is zero for any action. We restrict the class of POMDPs to those where we can observe if the state is terminal. In other words, we only use environments where it can be checked whether $\mathbf{s}_t$ is terminal, from the observation $\mathbf{o}_t$, as is often silently assumed in the RL field. As a consequence, the agent only has to learn the Q-values of trajectories that do not have reached a terminal state yet. Indeed, as can be seen in Algorithm 2, the computation of the target depends on whether o_t is terminal.

In practice, several POMDPs are studied. First, the Deterministic T-Maze environments that define a POMDP given the length L of the maze. This parameter allows playing on the size of the time-dependencies between a certain optimal action and a past observation. Second, a stochastic version of these environments is introduced, with a new parameter $\lambda \in [0, 1[$ that allows playing on the stochasticity of the transitions. Next, the agents are compared on the Mountain Hike POMDP, where the agent should learn to reach the top of the mountain while staying on the crests. This POMDP has a continuous observation space and the observations bring relatively less information about the state than it the case of the T-Maze. An even more difficult version of this POMDP, the Varying Mountain Hike is introduced. Here, the agent starts its trajectory randomly in one out of four different mountainous terrains and does not receive any additional information about its state. Only through its partial observations, it has to recognise the terrain and act accordingly. Finally, a Markov Decision Process is studied. It is known that using recurrent policies can provide an advantage, even in fully observable environments (Hausknecht & Stone, 2015). The results will show if one of the RNN can provide a greater benefit in comparison to the others.

5.2 Deterministic T-Maze

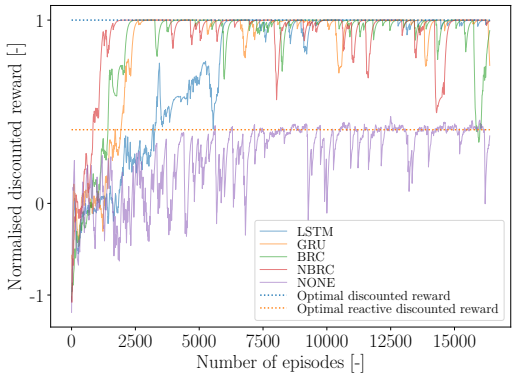
The T-Maze is a POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, p_0)$ with discrete state, observation and action spaces. The state-space is tabular and constituted of two T-shaped mazes, as can be seen in Figure A.1 in Appendix A.1. The T-Mazes are rotated to lie on their right side, such that the T-junction is at the right. The initial state is drawn randomly from the two initial cells of the two mazes. The observation in this initial state indicates in which maze the agent lies. The two mazes differ by the position of the goal: in the first, the goal is located above, while in the second, the goal is located below. The agent can move around in the T-Maze and it should learn to navigate towards the T-junction, and then to take the correct direction in order to find the goal. It should be done by remembering the initial observation that indicates which maze it was in. It receives a reward of 4 when finding the goal, and -0.1 when taking the wrong direction at the T-junction. In both cases, it reaches a terminal state. In addition, when the agent bounces onto a wall, it stands still and receives a reward of -0.1. In all other cases, the reward is 0. This environment is designed to assess the ability of the different RNNs to bridge long-term dependencies, without introducing any other difficulties to confound the results (Bakker, 2001). The Deterministic T-Maze is formally described as a particular case of the Stochastic T-Maze (see Section 5.3) in appendix A.1. It is noteworthy that any trajectory $\tau_{0:t} \in \mathcal{T}_{0:t}$ allows to determine the exact state of the process $\mathbf{s}_t$ for this environment. In other words, the belief b_t is always zero for all states except one.

For this environment, we decided to use an exploration policy $\mathcal{E}(A)$ that is tailored to this problem as the main point is not to study the exploration-exploitation problem. The exploration policy is such that $\mathcal{E}(\text{right}) = 0.5$ and $\mathcal{E}(\text{other}) = (1-0.5)/3$ where other $\in \{\text{up}, \text{left}, \text{down}\}$. This exploration policy is followed with probability ε according to the ε-greedy training described in Algorithm 2. The time horizon T has been chosen as the smallest integer such that a random walk for T time steps in an infinite one-dimensional corridor according to the exploration policy ends beyond $(L, 0)$ with a probability of at least 0.5.

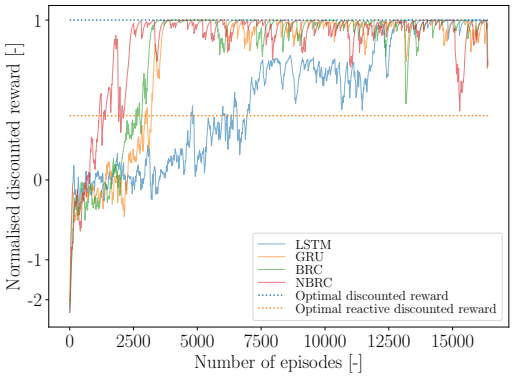
Figure 5.1 shows the learning curves (*i.e.*, the estimated expected cumulative reward of the fully greedy policy over the course of training) for the Deterministic T-Maze environments, averaged over the 3 training sessions. This figure also shows the theoretical optimal discounted reward and the optimal discounted reward that can be obtained by a stationary policy. The learning curves are displayed for all four RNNs that are considered in this work. It can be observed, that the policies are suboptimal during a certain amount of episodes, and then they sometimes converge towards the optimal policy, depending on the RNN and the length of the T-Maze. During the early stages of learning, it is common to observe that the policy corresponds to the optimal reactive policy. Indeed, the optimality of the policy will heavily depends on the ability of the RNN to learn that the value function at the T-junction depends on the action and on the initial observation ($\mathcal{Q}_\theta(\tau_{0:t}, \mathbf{a}_{\text{up}}) > \mathcal{Q}_\theta(\tau_{0:t}, \mathbf{a}_{\text{down}})$ when $\mathbf{o}_0 = \mathbf{o}_{\text{up}}$ and $\mathbf{o}_t = \mathbf{o}_{\text{junction}}$ and vice versa by exchanging up and down). In general, we observe that the bistable cells reach the optimality sooner and that their performance stays higher than those of the other RNNs. This effect is even exacerbated for longer T-Mazes, where only the bistable cells seem to achieve acceptable performances. For $L = 240$, only the nBRC succeed in outperforming the optimal reactive policy. The only exception to this trend is the length $L = 90$ for which the GRU is clearly performing as well as the BRC, and the nBRC is struggling in terms of stability of learning.

In Figure 5.2a, we can see the number of episodes after which the optimal policy was reached, averaged over all 3 training sessions, for each RNN and each length. Note that, for all subfigures of Figure 5.2, the confidence intervals show the maximum and minimum observed values over the 3 training sessions. We can clearly see that the two bistable cells allow learning the optimal policy way faster than the other standard recurrent cells. Furthermore, this figure neglects the fact that many of the training sessions have failed to achieve optimality for the LSTM and the GRU. Next, Figure 5.2b shows the proportion of evaluations for which the policy was evaluated optimal over the course of training. The proportion of optimal evaluations is a measure that reflects both the rapidity (and ability) to reach the optimal policy and the stability of learning. Again, we can observe that the BRC and nBRC are outperforming all other RNNs in terms of optimality proportion, and by a large amount as the length increases. Finally, Figure 5.2c shows the best mean discounted reward obtained over the course of training relative to the optimal policy. Once again, the BRC and the nBRC are clearly outperforming all other RNNs. The fact that bistable cells learn faster and more optimal policies should motivate their use in memory-demanding applications, especially when episodes of interactions are costly to produce.

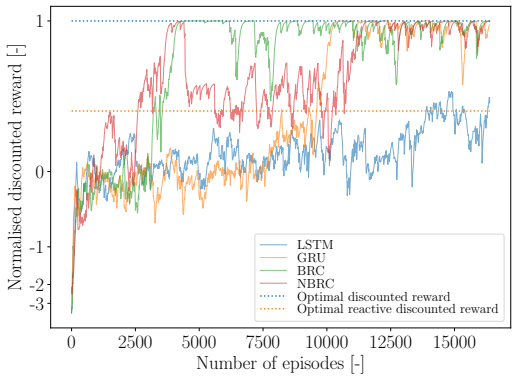
The better speed of convergence of the bistable cells towards the optimal policy could be explained by the bistability mechanisms introduced in (Vecoven et al., 2020). Indeed, information about the initial observation will likely hold out until the agents reach the junction, thanks to the neural bistability that allows some neurons to keep a stable value. The dependency between this information present in the hidden state and the Q-value at the junction will be exploited by the gradient updates to output the correct value. This hypothesis would imply that this bistability mechanism favours the discovery of long time dependencies. Afterwards, backpropagation through time would allow reinforcing the presence of this information until reaching the junction. The fact that the other recurrent architectures tend to vanish information about the initial observation will make the learning of the dependency at the junction harder. Concerning the better stability, it could once again be explained by the bistability mechanism that probably



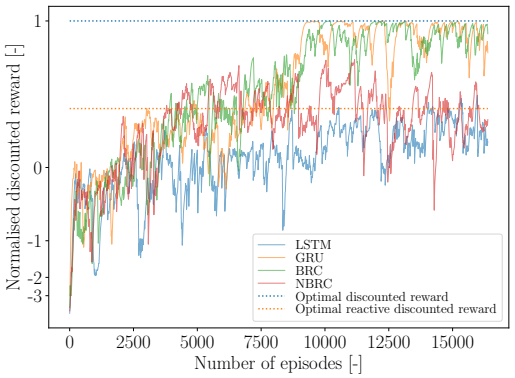
(a) T-Maze of length $L = 20$



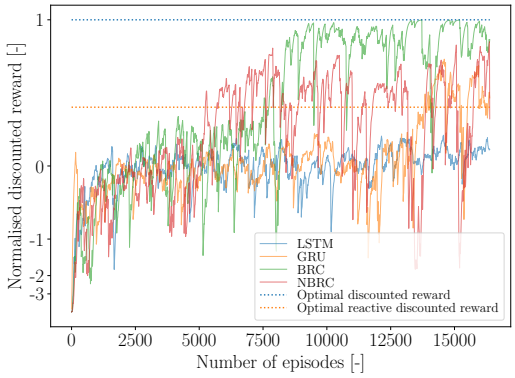
(b) T-Maze of length $L = 40$



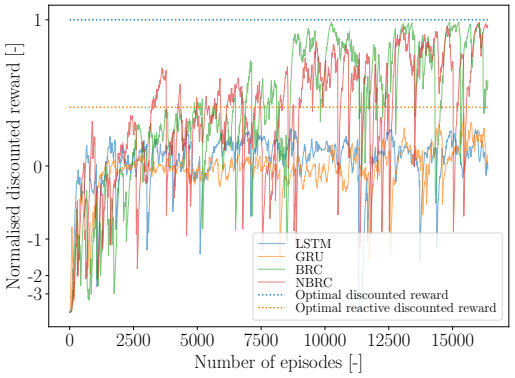
(c) T-Maze of length $L = 60$



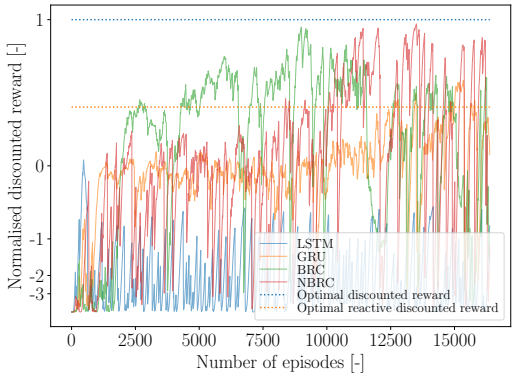
(d) T-Maze of length $L = 90$



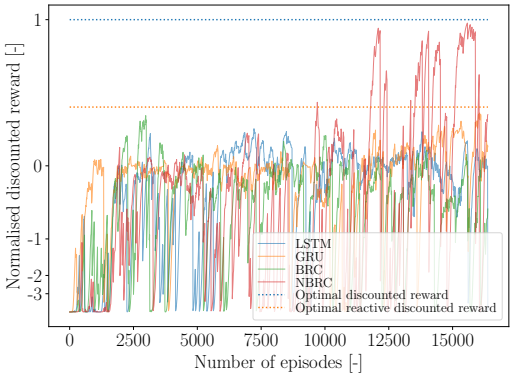
(e) T-Maze of length $L = 120$



(f) T-Maze of length $L = 160$

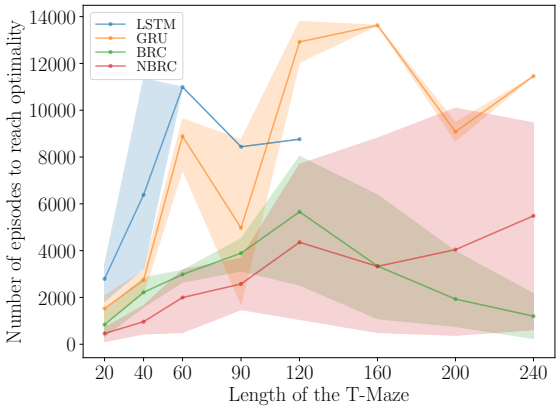


(g) T-Maze of length $L = 200$

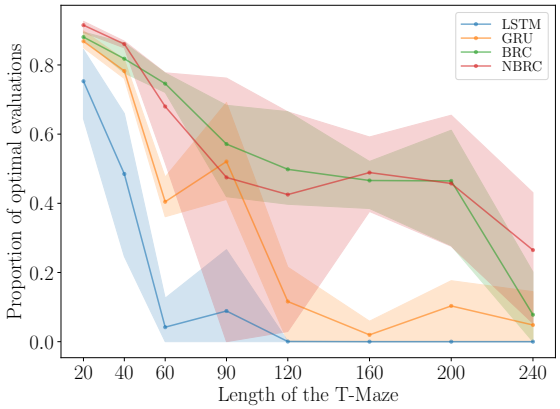


(h) T-Maze of length $L = 240$

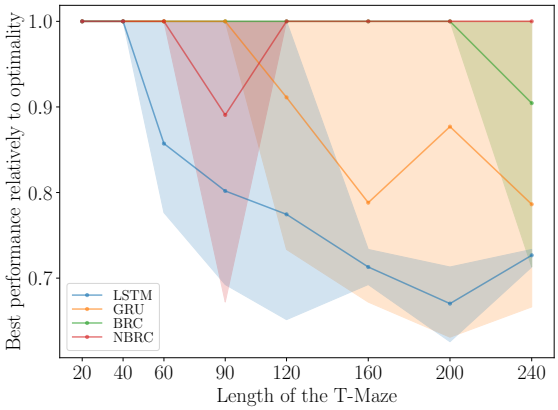
Figure 5.1: Learning curves for the Deterministic T-Maze



(a) Average number of episodes to reach optimality when the optimal policy was reached



(b) Proportion of evaluations with optimal policy



(c) Best performances obtained during training relatively to optimality

Figure 5.2: Additional analysis of the learning curves for the Deterministic T-Maze

allows keeping a stable value in the hidden states, maybe even after a small weight update (for the neurons that stay bistable). The recurrent cells would thus heavily rely on this stable value to output its prediction at the junction. For less stable cells, the footprint of the initial observation in the hidden state at the junction would presumably change more easily as the network weights are updated, causing the Q-values to degrade more easily. Indeed, the dependency between the last hidden states and the output will have to be learned again to a greater extent than with bistable cells since the distribution of the hidden states would change more easily at these distant time steps.

5.3 Stochastic T-Maze

The Deterministic T-Maze is generalised to present stochastic transitions, with a stochasticity parameter $\lambda \in [0, 1[$ that gives the probability to move in a random direction. When $\lambda = 0$, it is equivalent to the Deterministic T-Maze and the T-Maze of Bakker (2001). The stochastic environment is formally described in appendix A.1. In the following experiments, $\lambda = 0.4$ has been considered, which corresponds to ignoring the action of the agent and simulating a random action with probability 0.4.

From the learning curves of Figure 5.3, the same learning behaviour as for the Deterministic

T-Maze can be observed. However, for a fixed T-Maze length, the optimal policies are reached later for all RNNs. The RNNs also start failing sooner, at $L = 40$ for the LSTM, $L = 60$ for the GRU. However, as for the Deterministic T-Maze, the GRU unexpectedly succeed in reaching optimality for the length $L = 90$. Finally, for $L = 120$, all the RNNs fail except the nBRC that still achieves optimality. Although it only happened at $L = 240$ for the Deterministic T-Maze, it should be noted that the Stochastic T-Maze will lengthen the trajectories by $\frac{1}{1-\lambda}$ on average, which correspond to 1.67 here. This, and the fact that the trajectories fed to the RNNs are noisier, could explain why they fail twice earlier. In addition, it can be observed that some discounted rewards exceed the theoretical optimal discounted reward. This is due to the stochasticity and the fact that the performance is evaluated using $R = 50$ Monte Carlo rollouts. The last observation that can be made is that the performances of the policies are less stable over the training, compared to the Deterministic T-Maze.

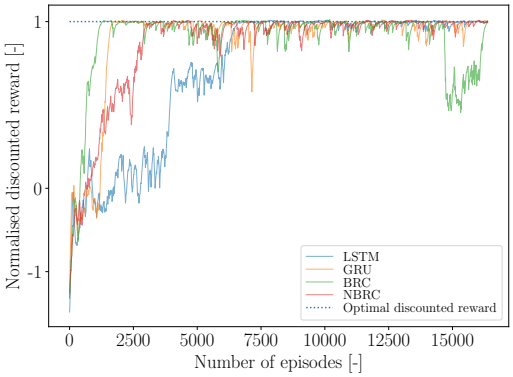
We can also see in Figures 5.4a and 5.4b that the bistable cells still surpass the other cells by a large amount in terms of rapidity and stability of learning. It means that they are both faster to learn and more optimal on average over the training. Interestingly, it can be seen from Figure 5.4c that the best performance of the GRU is very close to those of the two bistable RNNs, despite the GRU takes longer to reach optimality when it does and produces an optimal policy less often. It probably means that, although the GRU is able to remember the initial observation, it is less efficient in learning quickly the dependence between this initial observation and the Q-value at the junction.

To conclude, both on the Deterministic T-Maze and the Stochastic T-Maze, the bistable cells were found to be able to discover and learn long time-dependencies faster than the standard cells. In addition, these cells are more stable in their learning. Indeed, they have a far higher ratio of optimal evaluation over their course of training. Finally, they allow achieving better policies on harder environments, where the two standard cells completely fail in outperforming the optimal stationary policy. It already answers in the affirmative to the first research question of this work: the bistable cells allow extending the class of recurrent policies that we can represent in recurrent RL.

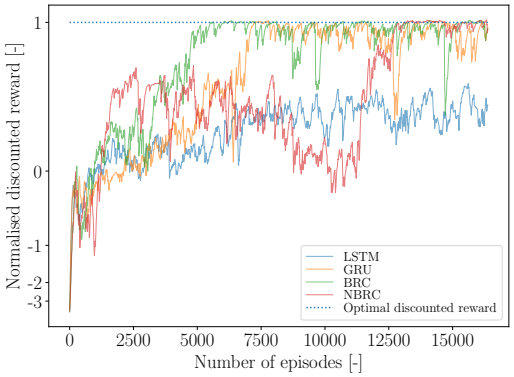
5.4 Mountain Hike

The Mountain Hike environment is a POMDP with continuous state space. The agent can move in 4 directions on a 2-dimensional map, with Gaussian noise on the transitions. The only observation that it makes is a noisy measure of its altitude. The latter is always negative. The reward is also equal to its altitude plus some noise. Around the highest point of the map, the states are terminal. The optimal policy consists in going as fast as possible towards these terminal states while staying on the crests in order to get less negative rewards than in the valleys. The environment is represented in Figure A.2 in Appendix A.2. This environment is inspired by the Mountain Hike environments described in (Igl et al., 2018). On the contrary to this environment where a noisy observation of the position was available, only the altitude is observed in this work, and with some noise too.

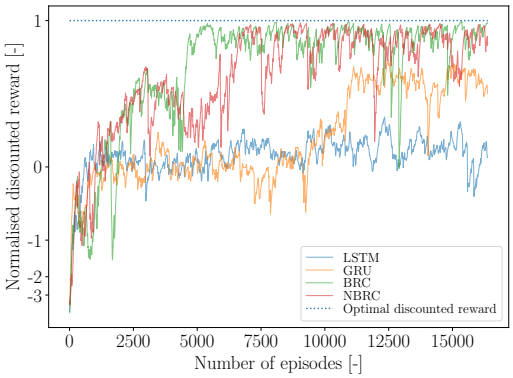
As Figure 5.5a illustrates, all RNNs seem to converge towards the same performance. It can be seen that the GRU learn faster, followed by the nBRC and BRC. The LSTM, however, is significantly slower to learn satisfactory policies. As mentioned in Section 5.1.1, Figure 5.5a displays the smoothed averaged performances from 4 training sessions. Sample trajectories from the first training session can be visualised in Figure 5.5b. From this figure, we can conclude that all RNNs are quite good at constantly filtering their position from the noisy altitude observation, in order to correct their trajectory towards the terminal states.



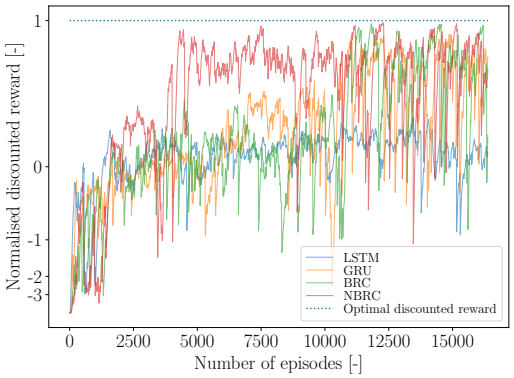
(a) T-Maze of length $L = 20$



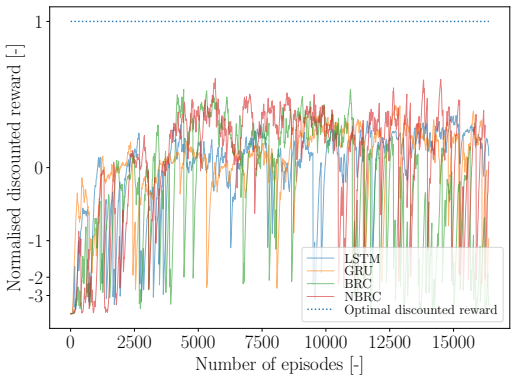
(b) T-Maze of length $L = 40$



(c) T-Maze of length $L = 60$

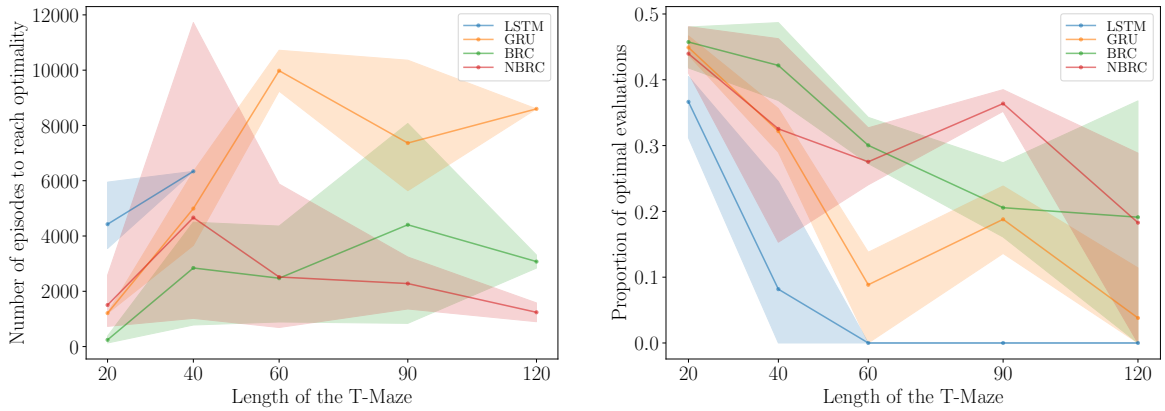


(d) T-Maze of length $L = 90$



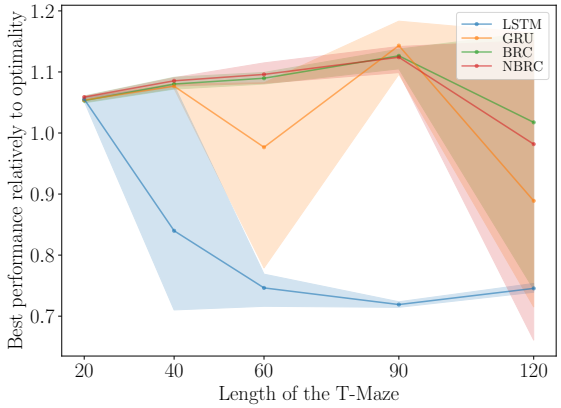
(e) T-Maze of length $L = 120$

Figure 5.3: Learning curves for the Stochastic T-Maze with $\lambda = 0.4$



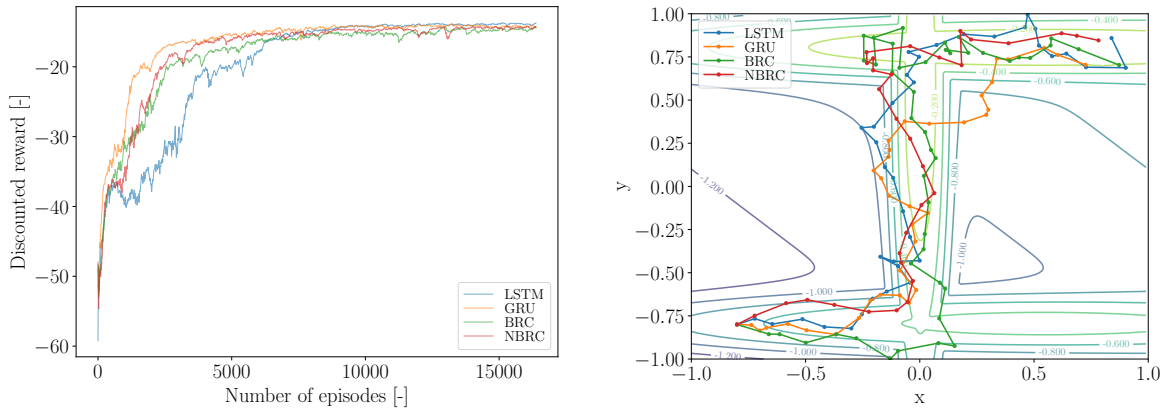
(a) Average number of episodes to reach optimality when the optimal policy was reached

(b) Proportion of evaluations with optimal policy



(c) Best performances obtained during training relatively to optimality

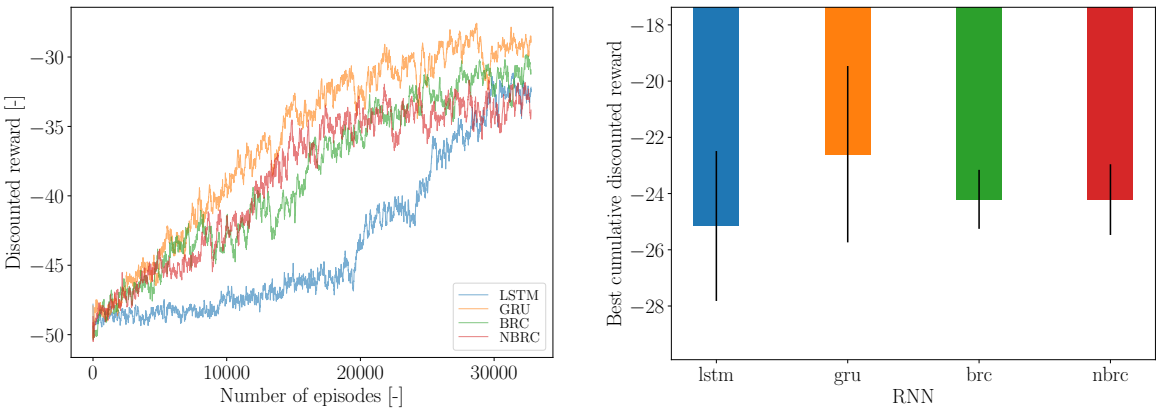
Figure 5.4: Additional analysis of the learning curves for the Stochastic T-Maze with $\lambda = 0.4$



(a) Learning curves

(b) Policies derived from the learned Q-values

Figure 5.5: Learning curves and illustration of the final policies for the Mountain Hike



(a) Learning curves

(b) Best performances obtained during training

Figure 5.6: Learning curves and best performances for the Varying Mountain Hike

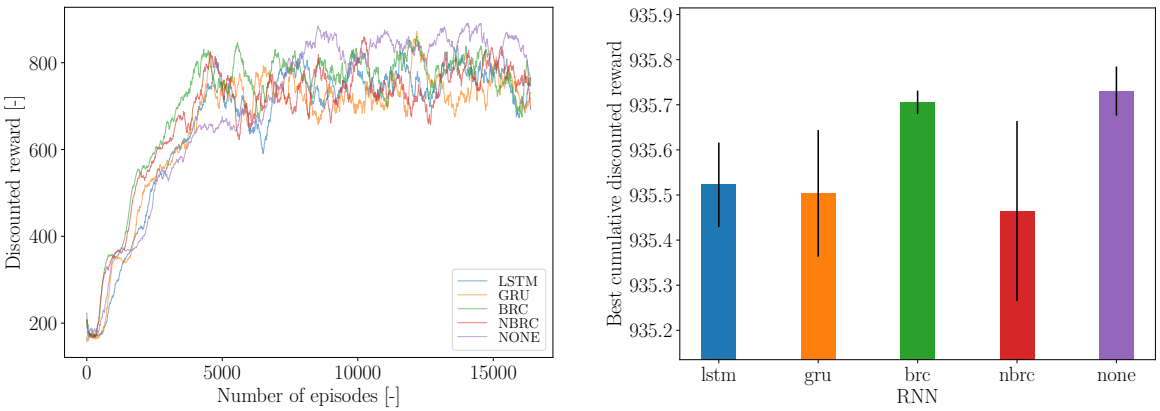
5.5 Varying Mountain Hike

The Varying Mountain Hike environment is a simple generalisation from the previous environment. Instead of having a single mountainous terrain, the terrain is drawn randomly from one out of four different variations at the start of each episode. The agent does not receive any new observation in addition to the altitude. These variations consist of the four rotations by 90° of the original terrain. It makes this POMDP a complicated environment in which the agent should succeed in recognising the terrain from very partial observations and then using the appropriate directions, while also constantly compensating the noisy transitions.

In this environment, it is clear that the GRU outperforms all other cells by a considerable amount, including the two bistable ones. This result is thus interesting. As often, the LSTM underperforms. Despite the (Varying) Mountain Hike might not require a high-memorisation ability to correctly estimate the Q-value, we would not have expected the bistable cells to underperform in comparison to the GRU. The GRU architecture may allow to better filter the position, given the altitude observations. Indeed, being able to reconstruct the belief — that is, the probability distribution over the positions — allows to correctly estimate the value of the trajectory. Our hypothesis is thus that the recurrent architectures of the bistable cells are less suitable than that of the GRU for the filtering of the belief in the particular case of the (Varying) Mountain Hike POMDP. This will be studied in Chapter 6.

5.6 Double Inverted Pendulum on Cart

As highlighted in (Hausknecht & Stone, 2015), in fully observable environments, using DRQN instead of DQN can still provide a better policy, certainly because using RNNs offers a higher representation capacity for the Q-functions. Because of that, it is interesting to compare the performance of DRQN to DQN for the Double Inverted Pendulum on Cart (DIPOC) environment. In addition, it will be interesting to see if the bistable cells still provide an advantage in the performance and in the rapidity of learning in environments that do not require a high memorisation ability, such as the DIPOC. However, using recurrent estimators that depends on the complete trajectory $\tau_{0:t}$ would be an overkill for just giving a small boost in performance. Instead, it is chosen to always feed $\mathcal{Q}_\theta$ with only the last W observations of the state. Consequently, the RNN that represents $\mathcal{Q}_\theta$ takes $\tau_{t-W+1:t}$ as input, where $\tau_{t-W+1:t}$ is defined by $(\mathbf{a}_{t-W}, \mathbf{o}_{t-W+1}, \ldots, \mathbf{o}_{t-1}, \mathbf{a}_{t-1}, \mathbf{o}_t)$. For this environment, W was chosen to 8. If $t < W$, $\mathbf{a}_k$ and $\mathbf{o}_k$ are taken to zero, for all $k < 0$, such that the RNN always consumes of sequence of length W



(a) Learning curves

(b) Best performances obtained during training

Figure 5.7: Learning curves and best performances for the Double Inverted Pendulum on Cart

before outputting its prediction. In this fully observable environment, it can also be noted that $\mathbf{o}_t = \mathbf{s}_t$.

As Figure 5.7a shows, the recurrent architectures provide a slight advantage in this environment, especially in terms of rapidity of learning. Indeed, the feedforward architecture, for which the two layers of RNNs are replaced by two feedforward layers, reaches the cumulative reward of 800 nearly twice later than the recurrent architectures. Moreover, the two bistable cells are learning slightly faster than the two standard cells. In terms of the best cumulative return obtained over the course of training, we cannot use Figure 5.7a to conclude, because of the exponential moving average filtering. However, Figure 5.7b shows this best performance obtained over the training, averaged over the 4 training session. This figure also shows the standard deviation of this best performance. Even if the differences in performance are tiny, it is clear, by looking at the estimated standard deviations, that the BRC and the non recurrent DQN are slightly better than the other cells.

Chapter 6

Belief filtering and generalisation in recurrent Q-learning

This chapter addresses the second research question of this master's thesis. Since the policies are derived from learned $\mathcal{Q}$-function, the quality of some of these functions is studied. In addition, since these learned $\mathcal{Q}$-function should embed the belief-filtering computation in the successive hidden states, the mutual information between these and the true belief is studied. The mutual information is assessed by the KSG estimator, and by the quality of the mappings that can be constructed between the hidden states and the belief, as explained in Section 3.5. Finally, for the T-Maze environments, the ability of each RNN to generalise to unseen POMDPs will be studied.

6.1 Description of the analyses

In addition to the second research question studying the belief-filtering ability of the different RNNs, this chapter analyses the quality of the Q-values. Indeed, the belief is a sufficient statistic from the trajectory to estimate the Q-values. Although a bad estimation of the $\mathcal{Q}$-function does not imply that the policy derived from this $\mathcal{Q}$-function is bad, it is interesting to observe these three metrics in parallel: the mutual information with the belief, the quality of the Q-values and the performance of the policies. The generalisation over the different lengths of the T-Mazes is also interesting to study, because, while it is likely that the learned $\mathcal{Q}$-functions do not generalise well, the policies could generalise well on these similar environments. In addition, the bistability mechanism could influence this capacity of generalisation.

6.1.1 Q-values

As explained in Section 3.2, computing $Q(b_t, \mathbf{a}_t)$ is a complicated task. However, for the first considered environment, the Deterministic T-Maze, the belief exactly predicts the state (*i.e.*, $b_t(\mathbf{s}_t) = 1$ and $b_t(\mathbf{s}) = 0, \mathbf{s} \neq \mathbf{s}_t$). For this environment, we thus have that the state-action value function $Q(\mathbf{s}_t, \mathbf{a}_t)$ is equal to the belief-action value function $Q(b_t, \mathbf{a})$. The learned $\mathcal{Q}$-function $\mathcal{Q}_\theta(\tau_{0:t}, \mathbf{a}_t)$ will thus be evaluated by comparing it to $Q(\mathbf{s}_t, \mathbf{a}_t)$. The later is easily estimated in the case of the Deterministic T-Maze. The quality of the Q-values is estimated by computing the MSE loss between $\mathcal{Q}_\theta(\tau_{0:t}, \mathbf{a}_t)$ and $Q(\mathbf{s}_t, \mathbf{a}_t)$ for a dataset of partial trajectories $\tau_{0:t}$ generated according to the ε-greedy policy on the learned $\mathcal{Q}$-function $\mathcal{Q}_\theta(\tau_{0:t}, \mathbf{a}_t)$. These trajectories are thus generated according to the same distribution as during the training process.

6.1.2 Belief reconstruction

As suggested by (4.9), the $\mathcal{Q}$-function that we learn should embed the filtering of the belief given the trajectory, in addition to outputting the belief-action value. As RNNs were chosen as function approximators of the $\mathcal{Q}$-function, they will be responsible to extract and memorise information from the observations that is relevant to estimate the value of a trajectory. This information will be conveyed through the hidden state. As (3.21) and (4.6) suggest, a sufficient statistic to correctly estimate the value of a trajectory is precisely the belief. Note, however, that partial information about the belief could be sufficient to correctly estimate its value. That being said, we assume, in the general case, that the hidden state should have a high mutual information with the belief to correctly estimate the value of the trajectory with the recurrent cell.

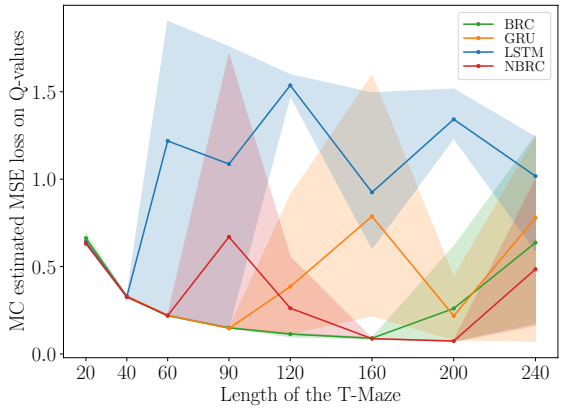
In view of that, we propose to study the mutual information between the hidden state and the belief with three distinct methods. First, the continuous mutual information between the hidden state $\mathbf{h}_t$ and the belief b_t will be estimated using the non-parametric mutual information estimation method known as the Kraskov estimator (Kraskov et al., 2004), as described in Section 3.5. These algorithms having a high time complexity for large datasets and requiring exponentially large datasets to correctly estimate the mutual information in the case of highly dependent variables, we decided to rely on two additional methods to control these results. These methods, motivated by $I(X;Y) = h(X) - h(Y \mid X)$ and the links that exist between maximising the mutual information and minimising the cross-entropy (Boudiaf et al., 2020), estimate the quality of the mapping that we can construct between $\mathbf{h}_t \sim X$ and $b_t \sim Y$. The first method estimates the cross-entropy loss on a test set after having trained a multilayer perceptron (MLP) to output b_t given $\mathbf{h}_t$ using a training and a validation set. The second, non-parametric, uses classification random forests to map $\mathbf{h}_t$ to b_t. The classification random forest is trained to output a state that is drawn according to the belief. In this case, the accuracy of predicting the most likely state of the belief is reported, this accuracy being computed on a separate test set. Finally, it can be noted that the dataset of beliefs and hidden states is extracted from trajectories generated according to the ε-greedy policies, as for the Q-values.

6.1.3 Generalisation to unseen environments

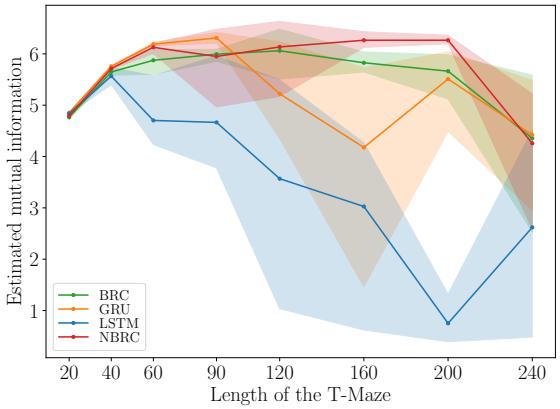
The T-Maze environments, describing a POMDP given a length L of the T-Maze, provide a family of environments that differ slightly from one another. Although we can easily think of a recurrent policy that generalises on all length of the T-Maze, it is not as easy to think of a recurrent Q-value estimation that would do so. Indeed, the Q-value will be very different and more changing, because of the discount factor, if L is bigger. It is thus likely the policy derived from the learned Q-values on a T-Maze of length L will not generalise well on other lengths. However, it will be interesting to see if some of the RNN still succeed in outputting an optimal policy when changing the length of the T-Maze. In other word, to see if the RNN is still able to predict $\mathcal{Q}_\theta(\tau_{0:t}, \mathbf{a}_{\text{up}}) > \mathcal{Q}_\theta(\tau_{0:t}, \mathbf{a}_{\text{down}})$ when $\mathbf{o}_0 = \mathbf{o}_{\text{up}}$ and $\mathbf{o}_t = \mathbf{o}_{\text{junction}}$ and vice versa by exchanging up and down, when the length of the T-Maze has been changed between the training and the evaluation. It is noteworthy that this analysis will be conducted on the T-Maze environments only.

6.2 Deterministic T-Maze

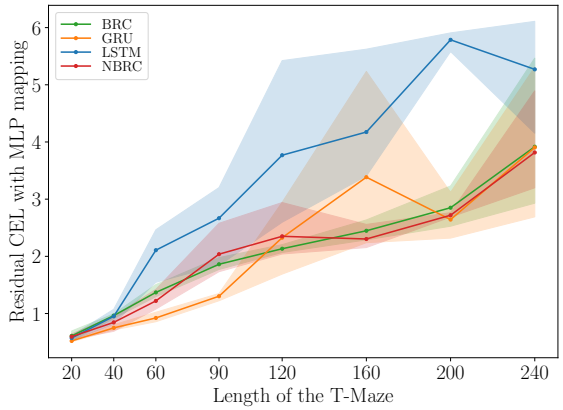
It can be seen in Figure 6.1a that the performance of the policies (see Figure 5.2c) are strongly correlated with the MSE on the learned Q-values. Since the belief (here, the state) is a sufficient statistic for estimating the value of a trajectory, it suggests that the mutual information between the hidden states and the belief could also be correlated with the performance. This is studied



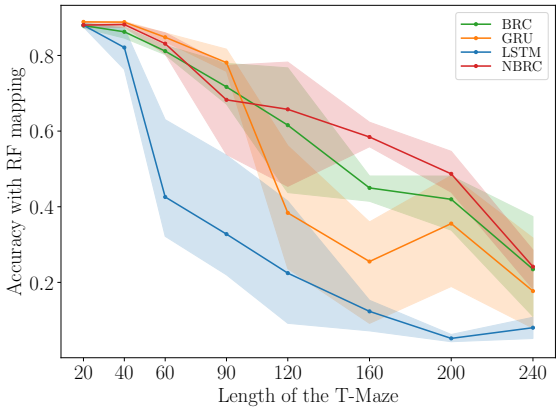
(a) MSE loss on learned Q-values



(b) Estimated mutual information between $\mathbf{h}_t$ and b_t



(c) Residual cross entropy loss of the multilayer perceptron mapping $\mathbf{h}_t$ to b_t

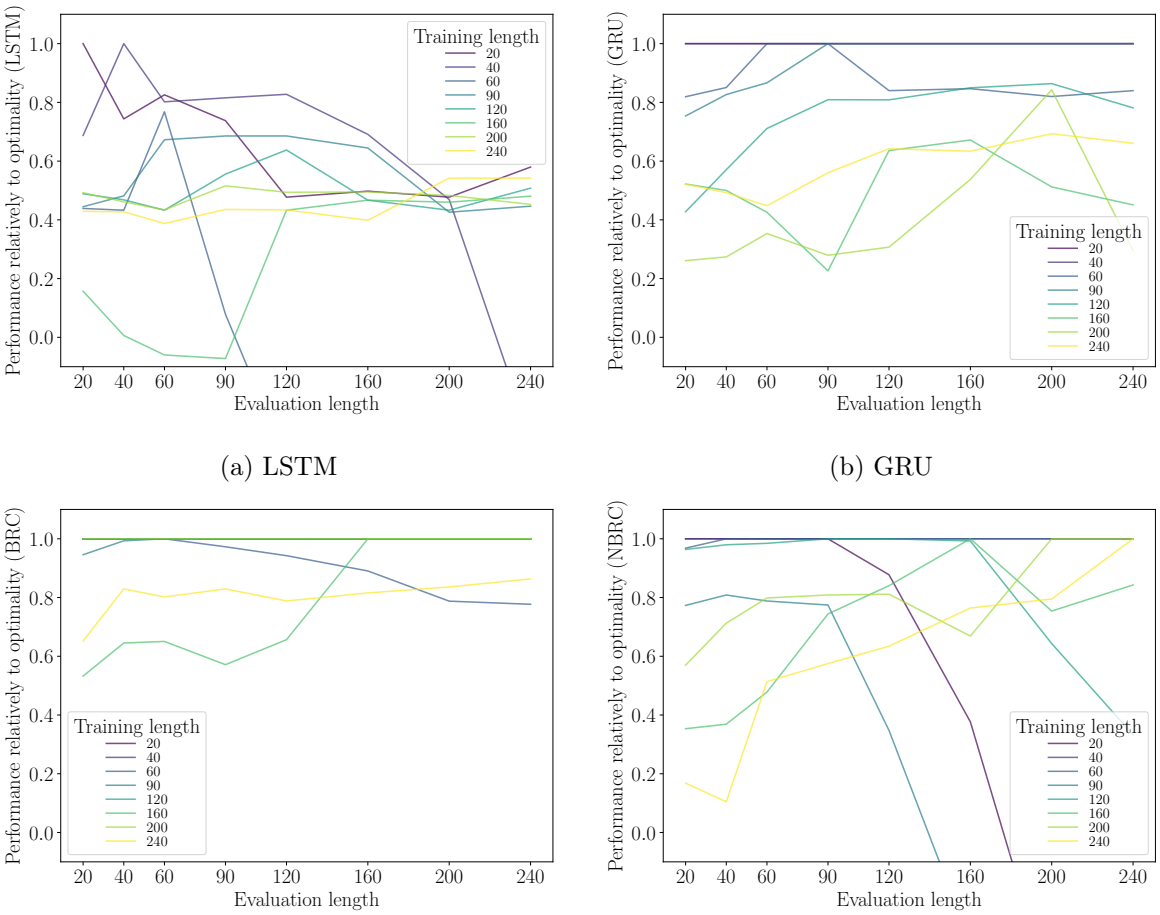


(d) Accuracy of the random forest mapping $\mathbf{h}_t$ to $\hat{s}_t$ (the most likely state from b_t)

Figure 6.1: Analysis of the outputs and recurrent states for the Deterministic T-Maze

in the following figures.

In Figure 6.1b, the mutual information between the hidden states and the beliefs is reported for all lengths and all RNNs. It shows that the mutual information between vanishes for RNNs that fail in producing an optimal policy. We can also see that the decrease in mutual information starts sooner, in terms of the length of the T-Maze, than the decrease in performance of the policy. In other words, a drop in the mutual information between the hidden state and the belief announces the difficulty of the RNNs to produce optimal policies. This suggests that the (un)ability to maintain a high mutual information between the hidden state and the belief is a cause of the (non) optimality of the policy. It can also be noticed that the mutual information is increasing with the length for small lengths of the T-Maze. This is not surprising since the mutual information is probably high for small lengths because the reconstruction of the belief is easy. And we know that the maximal mutual information increases as the length increases since the entropy of b_t is higher for bigger lengths. In Figure 6.1c, we can see that the ability of the MLP to predict the belief is in line with the mutual information that we measured. However, this analysis fails in discriminating the GRU from the bistable cells that perform better. Finally, since the T-Maze transitions are deterministic, we are able to estimate the accuracy of a classification random forest to predict the true state in Figure 6.1d. Once again, the results corroborate those of the mutual information.



(a) LSTM

(b) GRU

(c) BRC

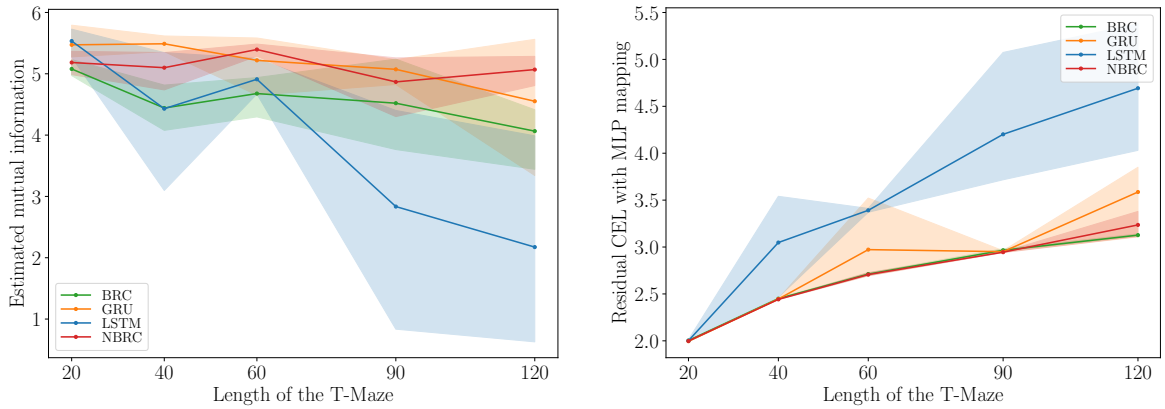
(d) NBRC

Figure 6.2: Generalisation to other lengths for the Deterministic T-Maze

Figure 6.2 compares the performances of the policies, trained on each of the lengths of the T-Maze, being evaluated on every other length of the T-Maze. For each training length, the performances are averaged over the 3 training sessions performed on this length. It can be seen that the BRC provide a better generalisation, with only around three training lengths that do not generalise on all T-Maze lengths, and that for the 3 training sessions. The nBRC and GRU are a bit worse at generalising but it is not bad. Finally, the LSTM is not good at generalising, even for the training sessions that converged towards the optimal policy. The fact that the BRC is clearly better at generalising than the GRU or the nBRC, despite they all perform similarly, could be due to its smaller number of parameters. Indeed, a smaller number of parameters imply a lower hypothesis space, and this could hypothetically prevent the RNN to overfit too much the $\mathcal{Q}$-value of the environment on which it has been trained.

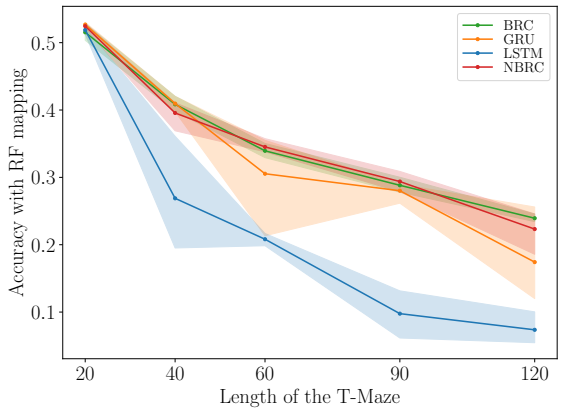
6.3 Stochastic T-Maze

As mentioned in the previous chapter, for the Stochastic T-Maze, λ has been fixed to 0.4. As can be seen in Figure 6.3, the results for the Stochastic T-Maze offer exactly the same conclusions as for the Deterministic T-Maze. Indeed, it is clear, by looking at the Figures 6.3a that the performances of the agent are correlated with the mutual information between their hidden states and the belief. Notably, the mutual information of the nBRC diverges from the others at $L = 120$, where it is the only one to achieve optimality. In Figure 6.3b, the cross-entropy loss for the MLP that maps the hidden state to the belief is reported. In Figure 6.3c, the accuracy



(a) Estimated mutual information between $\mathbf{h}_t$ and b_t

(b) Residual cross-entropy loss of the multilayer perceptron mapping $\mathbf{h}_t$ to b_t



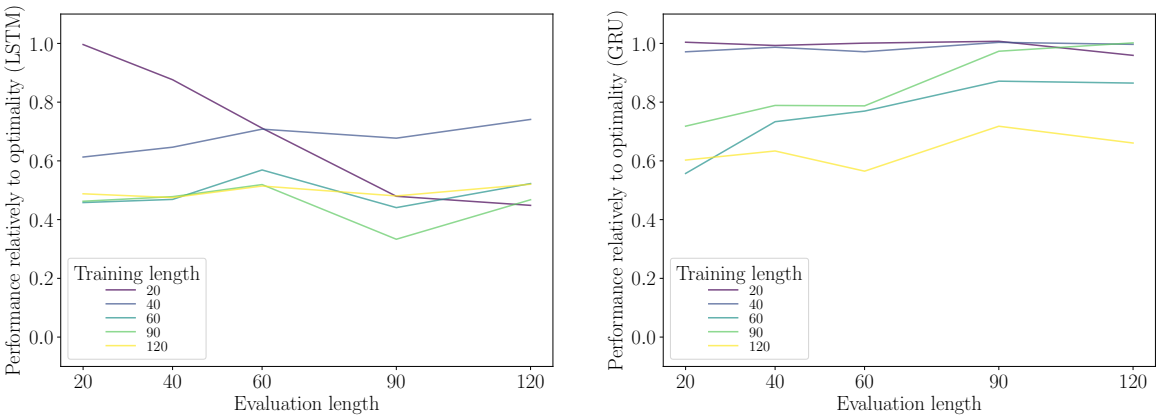
(c) Accuracy of the random forest mapping $\mathbf{h}_t$ to $\hat{s}_t$ (the most likely state from b_t)

Figure 6.3: Analysis of the outputs and recurrent states for the Stochastic T-Maze

of predicting the most likely state of the belief from the hidden state for the random forest is displayed. The results of the MLP and the random forest of Figures and 6.3c confirm the observations from the mutual information estimation, as for the Deterministic T-Maze.

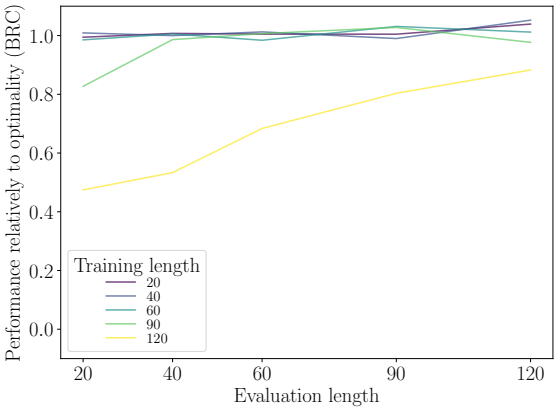
As a conclusion, these results, for both the Deterministic and Stochastic T-Maze, answer in the affirmative to the second research question of this master's thesis: in the context of value-based recurrent RL, the ability of the RNN to filter the belief in its hidden states directly influences the quality of the learned Q-values, and, as a consequence, the performance of the agent.

Finally, we can observe the same patterns of (non-)generalisation across the lengths of the T-Maze in Figure 6.4. As for the Deterministic T-Maze, the performances are averaged over the 3 training sessions performed, for each training length. The BRC is clearly outperforming all other RNNs in terms of generalisation to other lengths, even more than in the Deterministic T-Maze. Furthermore, it is observed that, for the Stochastic T-Maze, the nBRC seems to better generalise than the GRU. It is coherent with the observation that the GRU had a very high performance on the Stochastic T-Mazes, but often slightly suboptimal (see Figure 5.4c and 5.4b).

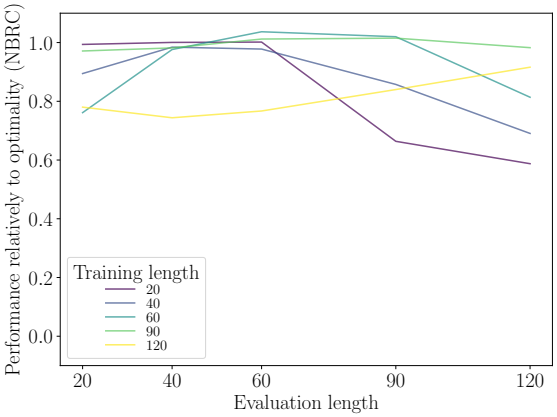


(a) LSTM

(b) GRU

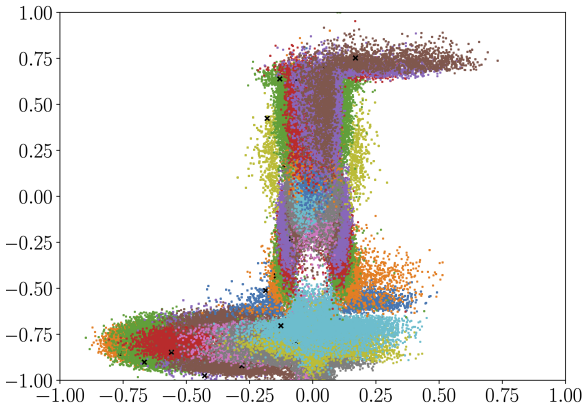


(c) BRC

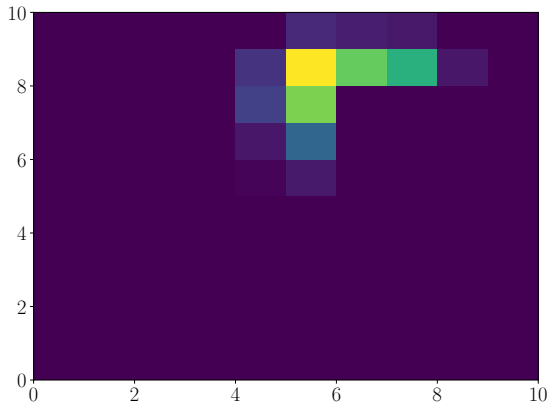


(d) NBRC

Figure 6.4: Generalisation to other lengths for the Stochastic T-Maze



(a) Illustration of the successive samples S_t representing the successive beliefs b_t



(b) Illustration of the parametrised belief $b_t(\cdot;\omega)$ for the last sample of Figure 6.5a (in brown)

Figure 6.5: Illustration of particle filtering and belief parametrisation for the Mountain Hike

6.4 Mountain Hike

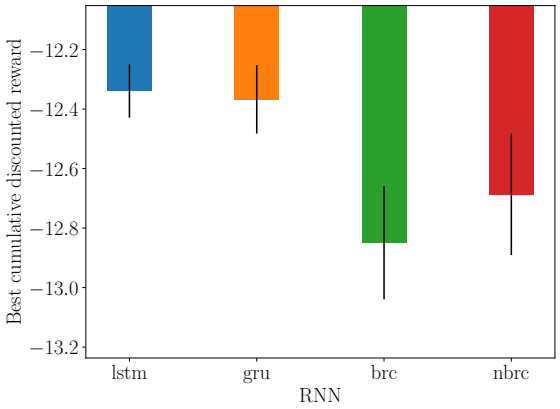
For the Mountain Hike, as explained in Section 3.5, the belief is approximated by a piecewise constant function over a uniform discretisation of the state space. An illustration of the particle filtering of the belief and the parametrised belief is available in Figure 6.5. This approximation is constructed from samples derived from the particle filtering procedure described in Algorithm 1.

The best performance obtained over the course of training is displayed in Figure 6.6a. In the following subfigures of Figure 6.6, the same analysis than for the T-Mazes is performed to evaluate the mutual information. The results on the mutual information are not as conclusive as for the T-Maze environments. Figure 6.6b shows a mutual information that is roughly aligned with the performances summarised in Figure 6.6a, except that the bistable cells have a slightly higher mutual information relatively to their performances. This could be explained by the very comparable performances of the different policies on this environment. Indeed, even if there is a slight difference in performance, it is not as drastic as for the T-Mazes, such that the mutual information does not allow to discriminate the performances of the different RNNs. Concerning the results of Figure 6.6c and Figure 6.6d, it can be noticed that they are not exactly aligned with the mutual information estimation, especially for the accuracy. It can be explained by reminding that the cross-entropy loss, or the accuracy, can be very low if Y is easy to predict without implying that Y convey much information about X and vice versa (*e.g.*, Y is a constant value). Indeed, these metric are rather proxies for $h(Y \mid X)$ than for $I(X;Y) = h(Y) - h(Y \mid X)$.

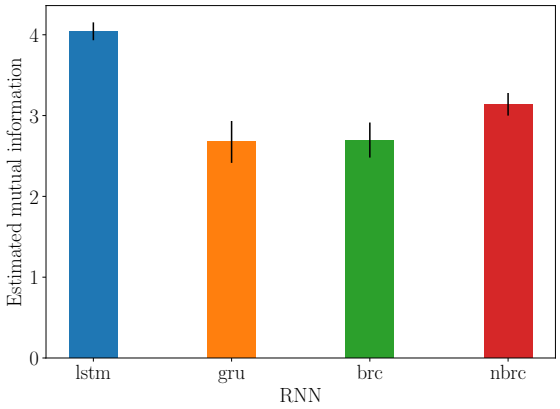
6.5 Varying Mountain Hike

The Varying Mountain Hike is treated exactly like the Mountain Hike. This environment, having four possible mountainous terrains, has a parametrised belief that is defined on the concatenation of the four discretisations of these four possible terrains. As for the Mountain Hike, the belief is represented by the frequencies of the particles in every bin of the concatenated discretisations. The particle filtering and parametrised belief are illustrated in Figure 6.7 for the Varying Mountain Hike. It can be noted that, after a few time steps, the particle filtering allows one to predict the terrain with high confidence. It can also be noticed the lower variance of the transitions in this environment, compared to the classic Mountain Hike.

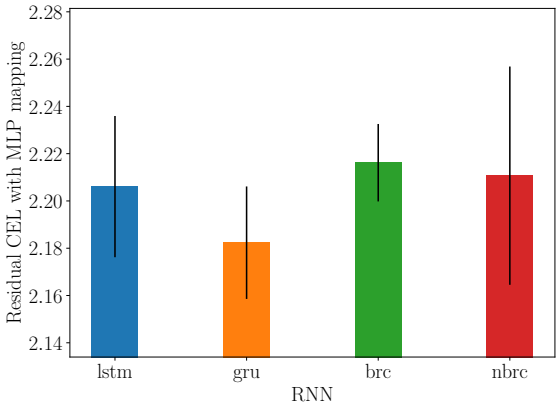
As for the Mountain Hike, the mutual information estimation in Figure 6.8b is not really correlated with the performance of the policies from Figure 6.8a. Moreover, the results of Figure 6.8c



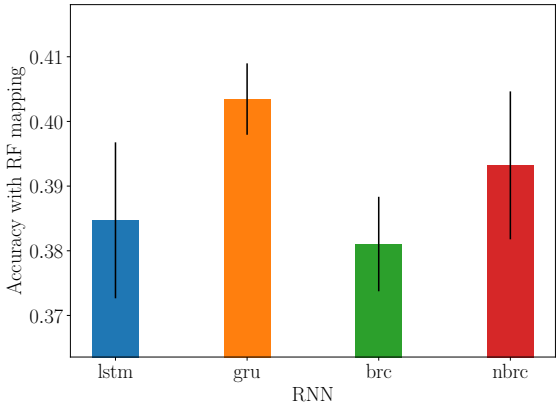
(a) Mean best performance obtained over the course of training



(b) Estimated mutual information between $\mathbf{h}_t$ and $b_t(\cdot \mid \omega)$

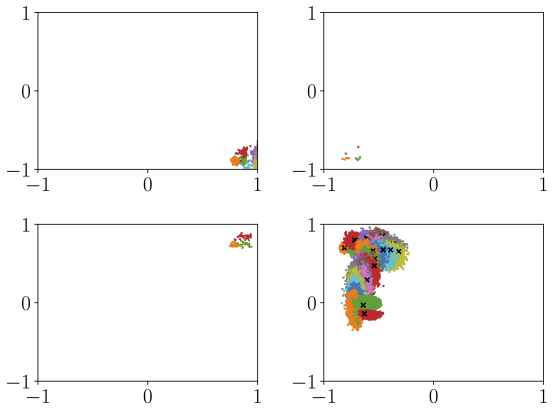


(c) Residual cross entropy loss of the multilayer perceptron mapping $\mathbf{h}_t$ to $b_t(\cdot \mid \omega)$

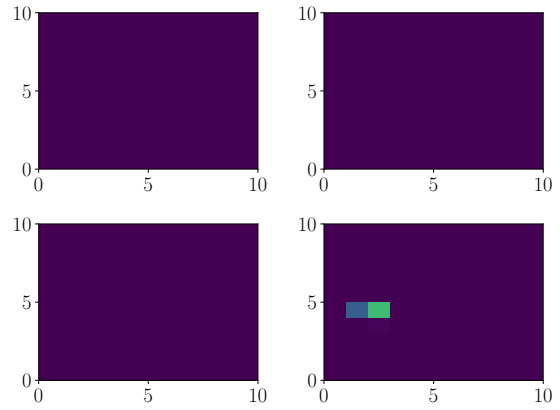


(d) Accuracy of the random forest mapping $\mathbf{h}_t$ to $\hat{s}_t$ (the most likely state from $b_t(\cdot \mid \omega)$)

Figure 6.6: Analysis of the performance and recurrent states for the Mountain Hike



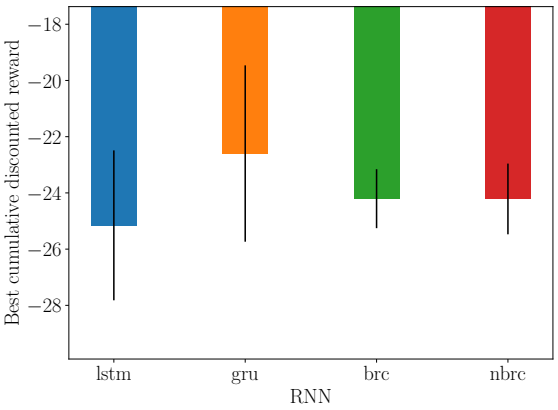
(a) Illustration of the successive samples S_t representing the successive beliefs b_t



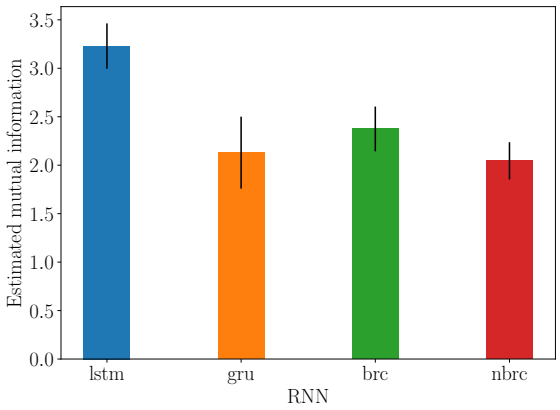
(b) Illustration of the parametrised belief $b_t(\cdot; \omega)$ for the last sample of Figure 6.7a (in red)

Figure 6.7: Illustration of particle filtering and belief parametrisation for the Varying Mountain Hike

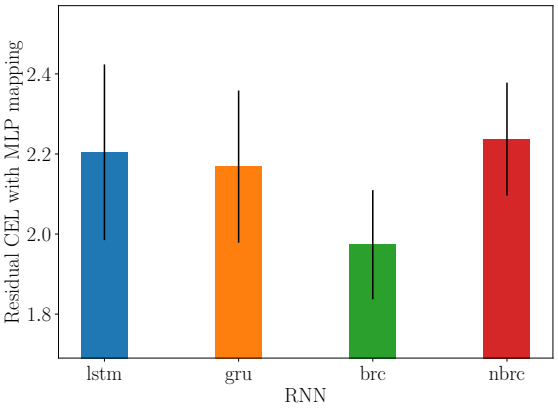
and 6.8d do not corroborate the results. Indeed, as for the Mountain Hike, the mutual information is unexpectedly high for the LSTM and is also higher for the bistable cells relative to their performances. These disappointing results could be explained by the fact that the quality of the policies is not necessarily correlated with the quality of the $\mathcal{Q}$-functions. However, the estimation of the true $\mathcal{Q}$-functions being really hard in this environment, we are not able to conclude on the quality of the Q-values learned by each of the RNNs, and thus to conclude on this hypothesis. The only thing that can be observed is that the learned $\mathcal{Q}$-function have not reached convergence yet after 32 760 episodes, with a loss still quite high, as can be seen in Figure 6.8e, suggesting indeed that the quality of the current $\mathcal{Q}$-functions is not very good.



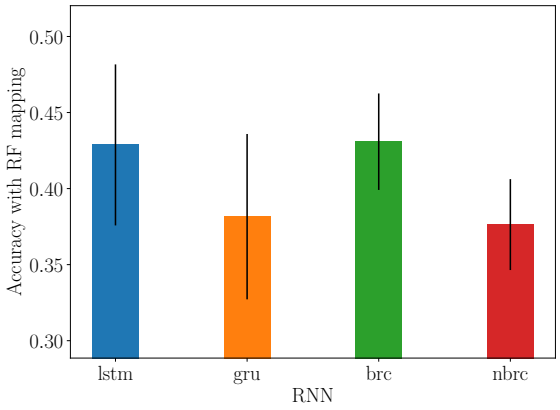
(a) Mean best performance obtained over the course of training



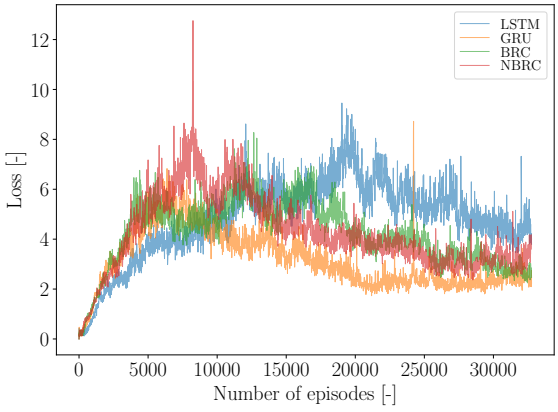
(b) Estimated mutual information between $\mathbf{h}_t$ and $b_t(\cdot \mid \omega)$



(c) Residual cross entropy loss of the multilayer perceptron mapping $\mathbf{h}_t$ to $b_t(\cdot \mid \omega)$



(d) Accuracy of the random forest mapping $\mathbf{h}_t$ to $\hat{s}_t$ (the most likely state from $b_t(\cdot \mid \omega)$)



(e) Mean losses over the learned Q-values over the course of training

Figure 6.8: Analysis of the performance and recurrent states for the Varying Mountain Hike

Chapter 7

Online Fitted Q-Iteration

In this chapter, independently of the rest of the research project, we highlight the lack of theoretical motivation for the presence of the target Q-network in Q-learning algorithms such as Deep Q-Network (DQN) (Mnih et al., 2015). In addition, we suggest that using a target Q-network defines a Q-iteration algorithm instead. We propose to extend the Fitted Q-Iteration algorithm (Ernst, Geurts, & Wehenkel, 2005) to the online learning setting, by introducing the Online Fitted Q-Iteration algorithm. The latter has key differences with the DQN algorithm, allowing to show a better convergence rate and better stability, especially in environments with sparse rewards. Finally, this algorithm gets rid of the target update period hyperparameter in favour of an objective update criterion.

7.1 Q-iteration and Q-learning

In this chapter, MDPs are considered, with states $x \in X$, actions $u \in U$, bounded rewards $r \in \mathbb{R}$. In addition, if $x = x_t$, it is conveniently chosen to denote x_{t+1} by y. On the one hand, the classic Fitted Q-Iteration algorithm as proposed by Ernst et al. (2005) learns the Q-function from a set of transitions $\mathcal{T} = \{(x_k, u_k, r_k, y_k)\}_{k=0}^{K-1}$. The idea is to successively learn approximations of $Q_1, Q_2, \ldots, Q_N$, namely $\hat{Q}_1, \hat{Q}_2, \ldots, \hat{Q}_N$. The function Q_i is the Q-function with horizon i, as defined in Section 3.2. The approximations are learned using the following successive datasets:

$$\mathcal{S}_n = \left\{ \left((x_k, u_k), r_k + \gamma \max_{u' \in U} \hat{Q}_{n-1}(y_k, u') \right) \right\}_{k=0}^{K-1}$$

for $n = 1, \ldots, N$ and with $\hat{Q}_0(x, u) = Q_0(x, u) = 0,\ \forall (x, u) \in X \times U$. This algorithm is known to be very stable in comparison to vanilla Q-learning with function approximators. Specifically, divergence to infinity is impossible for a certain class of kernel (tree) estimators, and convergence can be ensured thanks to the normalizing condition for some of these estimators. This algorithm, initially proposed with trees and random forests as function approximators, was adapted by Riedmiller (2005) to use neural networks as function approximators.

On the other hand, Q-learning is a classic model-free algorithm in reinforcement learning that allows learning the Q-function from observed transitions. The algorithm consists in updating our approximation $\hat{Q}$, initialised at 0 for all $(x, u) \in X \times U$, each time that a new transition (x_k, u_k, r_k, y_k) is encountered, by the following rule:

$$\hat{Q}(x_k, u_k) \leftarrow (1 - \alpha_k)\hat{Q}(x_k, u_k) + \alpha_k \left(r_k + \gamma \max_{u' \in U} \hat{Q}(y_k, u') \right) .$$

This algorithm can be shown to converge under approximate hypotheses on α_k and on the generated transitions. Parametric Q-learning allows approximating the Q-function by a parametric

function approximator Q_θ in large state space, where it would be non-tractable to maintain an approximation for all $(x, u) \in X \times U$. This algorithm consists in successively updating the approximation Q_θ as follows:

$$\theta \leftarrow \theta + \alpha \left(r_k + \gamma \max_{u' \in U} Q_\theta(y_k, u') - Q_\theta(x_k, u_k) \right) \nabla_\theta Q_\theta(x_k, u_k) \ .$$

which corresponds to one gradient step with respect to θ on the mean squared error loss between the temporal difference target $r_k + \gamma \max_{u' \in U} Q_\theta(y_k, u')$ and the current estimation $Q_\theta(x_k, u_k)$, with the temporal difference target considered fixed with respect to θ. The update for parametric Q-learning is known to be unstable, due to the fact that the temporal difference target $r_k + \gamma \max_{u' \in U} Q_\theta(y_k, u')$ is moving for any fixed (x_k, u_k) as the network is updated. Note that this moving target is not observed when regressing in the Fitted Q-Iteration algorithm, where the target is fixed, as in supervised learning, for each n. In other words, Q-learning is updated using its current estimation, while Q-iteration is updated using a well-defined estimation that has already converged.

The most successful application of parametric Q-learning is the Deep Q-Network algorithm (Mnih et al., 2015) in which the instability was addressed by introducing a target network $Q_{\theta'}$ whose parameters θ' are only updated once every C parameters updates to the value of θ. It eases the convergence by keeping the temporal difference target fixed for long periods. In addition, this algorithm makes use of a replay buffer from which to sample transitions, such that the successive transitions used in the update are not necessarily correlated. It also allows increasing the sample efficiency of the algorithm. However, the performance of the DQN heavily relies on the introduction of the target Q-network. We claim that the introduction of this target network makes DQN close to an (online fitted) Q-iteration algorithm and not a (temporal difference) Q-learning algorithm because the introduction of the target is not theoretically motivated in the context of Q-learning while it is the essence of Q-iteration. A direct consequence of understanding DQN as a Q-iteration algorithm is that the policies derived from the Q-functions learned by DQN after k target updates are at best k-steps greedy policies. As C is usually chosen very big to keep the algorithm stable (10 000 is a standard value), the convergence towards an optimal policy instead of a greedy one is very slow for DQN.

Finally, it should be noted that, on the contrary to Q-learning, the Fitted Q-Iteration algorithm is currently offline only. In other words, it requires a fixed set of transitions to learn from. In some applications, we do not have an approximate policy from which to generate interesting transitions and we have to rely on random exploration. However, generating a large set of transitions according to a random policy can be costly and we might prefer to learn online, such that we generate better and better transitions (in the sense that there is more to learn from them than from totally random transitions because they explore a more rewarding part of the state-action space). Consequently, online learning typically reduces the number of transitions needed to converge towards a near-optimal policy. It motivates the introduction of an online version of the Fitted Q-Iteration algorithm. In addition, if DQN should indeed be understood as an (online fitted) Q-iteration algorithm to give grounds to the target network, some adaptations have to be done to make it a correct fitted Q-iteration algorithm, and we propose the Online Fitted Q-Iteration algorithm that includes these adaptations. Anchoring the Online FQI in the Q-Iteration algorithms thus justifies the existence of the target Q-Network, and motivates two differences with the DQN that should improve the stability of the algorithm and the speed of convergence. Conversely, some choices in the design of the Online Fitted Q-Iteration are inspired by the DQN, which is known to perform very well in complex environments.

7.2 Online Fitted Q-Iteration

To generalise the Fitted Q-Iteration to the online case, several choices are possible. The most straightforward choice would be to generate a set $\mathcal{S}_1$ of $S \ll K$ transitions and learn $\hat{Q}_N^{(1)}$ over this set. And then, we could discard this set and start over by sampling a second set $\mathcal{S}_2$ of $S \ll K$ transitions from the ε-greedy policy derived from $\hat{Q}_N^{(1)}$ in order to learn $\hat{Q}_N^{(2)}$. The successive set $\mathcal{S}_1, \mathcal{S}_2, \ldots$ should progressively include better transitions. And we continue until convergence of the policy. However, this method has two drawbacks. First, we have to learn all successive approximators $\hat{Q}_1^{(i)}, \ldots, \hat{Q}_N^{(i)}$ from scratch for all new set $\mathcal{S}_i$. It would be better to learn the N estimators throughout the generation of the successive datasets. Second, some data is discarded and we only keep the transitions generated from the ε-greedy policy $\hat{Q}_N^{(i-1)}$, possibly losing information about less optimal trajectories that were present in the first sets of transitions. Here, it would be better to use a growing batch of transitions, or replay buffer, in which all transitions are added but are not discarded.

In the resulting algorithm, the successive $\hat{Q}_1, \ldots, \hat{Q}_N$ functions are learned successively while new transitions are added to the replay buffer. The estimator $\hat{Q}_n$ is trained on the transitions sampled from the replay buffer using $\hat{Q}_{n-1}$ as target Q-network. The target network $\hat{Q}_{n-1}$ is updated to $\hat{Q}_n$ when convergence is reached for $\hat{Q}_n$. At this moment, we can start to approximate $\hat{Q}_{n+1}$.

Two additional choices can be performed. First, we have to choose the Q-function from which we derive the behaviour policy, typically a ε-greedy policy. Two natural choices are $\hat{Q}_n$, the Q_n-function being learned or $\hat{Q}_{n-1}$, the Q_{n-1}-function that has reached convergence and that is used as target network. To be similar to the DQN algorithm, we choose to use $\hat{Q}_n$. In addition, there is more exploration when using $\hat{Q}_n$ since it is updated and changes during its training while $\hat{Q}_{n-1}$ is fixed. Second, as we use neural networks as function approximators in this work, each time that we increment n, we can either reinitialise the weights θ randomly to estimate $\hat{Q}_n$, or we can keep θ at its value of convergence for $\hat{Q}_{n-1}$. We choose the second alternative, because the distance between Q_n and Q_{n-1} is bounded in the case of bounded rewards (*i.e.*, Q_n is close to Q_{n-1}). In addition, strengthens the similarity with DQN.

The Online Fitted Q-Iteration algorithm is detailed in Algorithm 3 and can be compared to the Deep Q-Network (Algorithm 4) and the batch-mode Fitted Q-Iteration (Algorithm 5). The OFQI and FQI algorithms are only detailed for parametric function approximators, but the adaptation to other approximators is straightforward, using batch-mode approximators for FQI and online approximators for OFQI. The differences between DQN and OFQI are highlighted in grey. As far as the differences with batch-mode FQI are concerned, we have two main ones. First, the amount of data available to fit the estimators $\hat{Q}_n$ increases as n increases. Second, the data available to fit the estimator $\hat{Q}_n$ will not be generated by a single exploration policy, but by a sequence of policies, each of them being modified at each gradient step, going from an $(i-1)$-steps greedy policy to an i-steps greedy policy, for $i = 1, \ldots, n$. This is comparable to the DQN algorithm. In comparison to batch-mode FQI, we are also limited to online estimators such as neural network, or online random forest (*e.g.*, Mondrian Forests such as proposed by Lakshminarayanan, Roy, and Teh (2014)).

A natural objection to the Online Fitted Q-Iteration algorithm is that the first estimators are trained on few data in comparison to the amount of data that will be generated over the course of training. Since all the following estimators will recursively use the previous estimators to compute their targets, the bad quality of the formers could worsen the quality of the latter. However, as n increases, the influence of the first estimators vanishes, and the Q-functions should converge towards the unique solution of the Bellman equation. In other words, the OFQI algorithm should mainly speed up the convergence during the early stages of learning, despite

having a worse approximation of the last rewards of a trajectory, that are heavily discounted in the cumulative reward as n increases. But in the final stages of learning, this algorithm should converge towards the Q-function thanks to the Q iterations that should be more and more frequent.

Algorithm 3: Online Parametric Fitted Q-Iteration

Initialise weights θ randomly to represent Q_n
Initialise weights θ' such that $Q_{\theta'}(x, u) = 0$, $\forall (x, u) \in X \times U$ to represent Q_{n-1}
Add S transitions from a random policy to the replay buffer
Reset environment and observe initial state y
Set $n \leftarrow 1$
Set $s \leftarrow 0$
for $t = 0, \dots, T-1$ **do**
    $x \leftarrow y$
    Select $u \sim \mathcal{U}(U)$ with probability ε, otherwise select $u \leftarrow \arg\max_{u \in U} \{Q_\theta(x, u)\}$
    Take action u and observe r and y
    Store (x, u, r, y) in the replay buffer
    Sample B transitions (x_b, u_b, r_b, y_b) from the replay buffer
    Compute target $y_b = \begin{cases} r_b + \gamma \max_{u' \in U} \{Q_{\theta'}(y_b, u')\} & \text{if } y_b \text{ is terminal} \\ r_b & \text{otherwise} \end{cases}$
    Compute loss $L = \sum_{b=0}^{B-1} (y_b - Q_\theta(x_b, u_b))$
    Optimise θ to minimise L using Adam optimiser with learning rate α
    if *y is terminal* **then**
        Reset environment and observe initial state y
    if *Q_θ has converged* **then**
        $\theta' \leftarrow \theta$
        $n \leftarrow n + 1$

7.3 Experiments

As mentioned in the previous section, the target update of the OFQI algorithm is performed when convergence is detected. In practice, we propose to implement this by introducing a new hyperparameter U that replaces the target update period C present in the DQN algorithm. This hyperparameter U, typically chosen between 10 and 50, concludes for the convergence of Q_n if no improvement of the MSE loss has been observed on the last U mini-batches. This is just one possibility for concluding on the convergence of Q_n. Note that the results are averaged from 4 training sessions in the following figures, as for the Mountain Hike environments.

7.3.1 Double Inverted Pendulum on Cart

The first considered environment is the Double Inverted Pendulum on Cart, as introduced in Section 5.6 and formally described in Appendix A.4. In Figure 7.1a, a classic choice of $C = 1000$ is made for the DQN algorithm, and $U = 10$ is chosen for the OFQI algorithm. It is clear that the slow target update completely delays the DQN convergence. Indeed, we can see in Figure 7.1b that the losses for the OFQI algorithm are far higher than for the DQN algorithm while still being reasonable. It suggests that the OFQI is constantly trying to learn as soon as it has converged, by updating the target, instead of idly waiting for the next target update. This is precisely the desired behaviour for the OFQI algorithm.

Algorithm 4: Deep Q-Network

Initialise weights θ randomly
$\theta' \leftarrow \theta$
Add S transitions from a random policy to the replay buffer
Reset environment and observe initial state y
for $t = 0, \ldots, T-1$ **do**
    $x \leftarrow y$
    Select $u \sim \mathcal{U}(U)$ with probability ε, otherwise select $u \leftarrow \arg\max_{u \in U} \{Q_\theta(x, u)\}$
    Take action u and observe r and y
    Store (x, u, r, y) in the replay buffer
    Sample B transitions (x_b, u_b, r_b, y_b) from the replay buffer
    Compute target $y_b = \begin{cases} r_b + \gamma \max_{u' \in U} \{Q_{\theta'}(y_b, u')\} & \text{if } y_b \text{ is terminal} \\ r_b & \text{otherwise} \end{cases}$
    Compute loss $L = \sum_{b=0}^{B-1} (y_b - Q_\theta(x_b, u_b))$
    Optimise θ to minimise L using Adam optimiser with learning rate α
    if *y is terminal* **then**
        Reset environment and observe initial state y
    if $e \equiv 0 \pmod{C}$ **then**
        $\theta' \leftarrow \theta$

Algorithm 5: Parametric Fitted Q-Iteration

Input: Set of K transitions
Initialise weights θ randomly to represent Q_n
Initialise weights θ' such that $Q_{\theta'}(x, u) = 0$, $\forall (x, u) \in X \times U$ to represent Q_{n-1}
Reset environment and observe initial state y
for $n = 1, \ldots, N-1$ **do**
    while *Q_θ has not converged* **do**
        Sample B transitions (x_b, u_b, r_b, y_b) from the set of transitions
        Compute target $y_b = \begin{cases} r_b + \gamma \max_{u' \in U} \{Q_{\theta'}(y_b, u')\} & \text{if } y_b \text{ is terminal} \\ r_b & \text{otherwise} \end{cases}$
        Compute loss $L = \sum_{b=0}^{B-1} (y_b - Q_\theta(x_b, u_b))$
        Optimise θ to minimise L using Adam optimiser with learning rate α
    $\theta' \leftarrow \theta$

However, we can try to improve the DQN algorithm by choosing a less standard but more adapted parameter. Indeed, we can try with $C = \bar{C}_U$ where $\bar{C}_U$ is defined as the average update period performed by the OFQI algorithm with hyperparameter U. For the DIPOC, we observed that $\bar{C}_{10} \approx 18$. Since the OFQI algorithm is good with this average update period, we might think that it is a good period for the DQN algorithm. However, as Figure 7.1c shows, the DQN algorithm with $C = 18$ is not as good as the OFQI algorithm. This is very interesting since it suggests that the better performance of the OFQI is not only due to its automatic tuning of the parameter C, but also probably to the fact that it updates the target at the *right* moment. In addition, and this was the main reason for the introduction of the target Q network in the DQN algorithm, the DQN algorithm is not stable at all when having a too high target update frequency. We can indeed see in Figure 7.1d that the losses have exploded at one point for the DQN algorithm with $C = 18$, while they have not for the OFQI with $U = 10$ and $\bar{C}_{10} \approx 18$. However, we can also see in Figure 7.1c that an adequate choice of the parameter $C = 100$ for the DQN algorithm allows getting similar performance to the OFQI algorithm. This, however, has required to tune C for this particular environment.

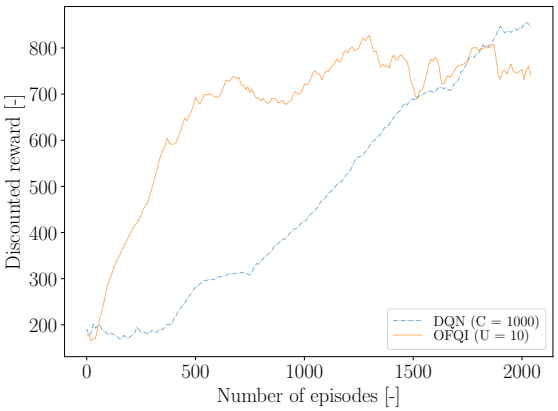
Finally, Figure 7.1e shows all training sessions together, such that they can be compared. We can also see that the OFQI algorithm with $U = 20$ achieve a similar performance than with $U = 10$. Furthermore, the DQN algorithm with $C = 5$ has shown an even worse explosion in the losses, as can been seen in Figure 7.1f. As suggested in the previous section, we can also notice in Figure 7.1c that the OFQI advantage is especially present in the early stages of training.

7.3.2 Car on the Hill

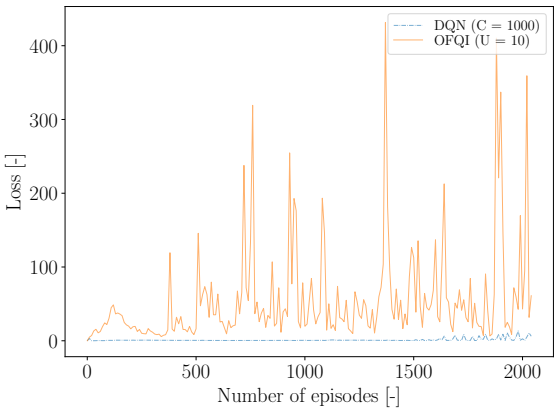
The second environment is the Car on the Hill environment. This environment is taken exactly the same as in the original FQI paper (Ernst et al., 2005), where it is described very precisely in the Appendix C.2. In this environment, a car is travelling on a hill, and the agent has two actions available: full acceleration and full deceleration. The reward is 1 in the terminal state that is on top of the hill. There are also terminal states when the car goes out of the road or exceed the speed limit of 3, with a -1 reward in both situations. In all other states, the reward is 0. The goal of the agent is thus to reach the top of the hill as soon as possible given these constraints. The dynamics of the system are such that the car cannot just climb the hill straightforwardly. Indeed, the car has to gain so momentum in order to succeed in climbing the hill.

All the results, averaged over the 4 training sessions, are displayed in Figure 7.2a. In this case, as can be seen in Figure 7.2b, no explosion in the losses was observed. This is probably due to the very sparse rewards in this environment. In this context, it can be seen that the DQN can still suffer from slow convergence when the hyperparameter C is badly chosen. However, here, without risk of explosion in the loss, we can observe that using the DQN algorithm with C taken to $\bar{C}_{10} \approx 18$ provide a very good learning process. In this case, we can thus conclude that the OFQI algorithm is not offering much more than an automatic tuning for the parameter C.

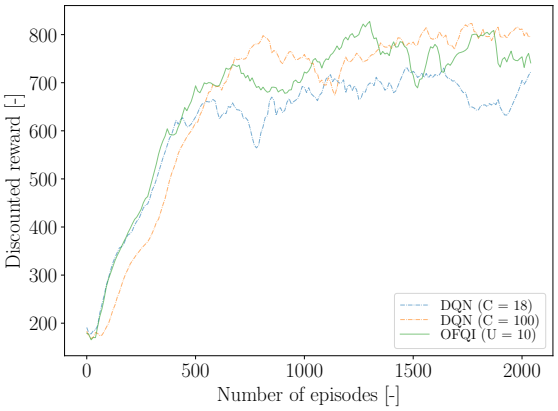
Finally, it should be noted that these analyses are quite limited, and focus on rather simple environments, that are known to require a lower target update period. But these analyses motivate future research on the target update and can be seen as glimpses of the benefits that could be brought by this OFQI algorithm in some environments. Moreover, it is reasonable to say that the OFQI advantages would probably be more dramatic in more complex environments, where C is typically chosen very large just to ensure stability over the course of learning, possibly losing thousands of gradient updates overfitting the successive Q_i-functions.



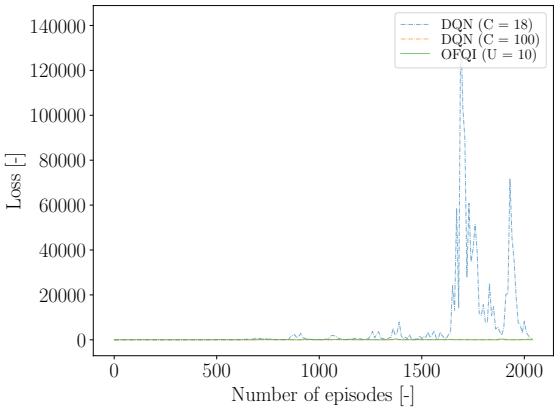
(a) Learning curves for a classic choice of hyperparameters



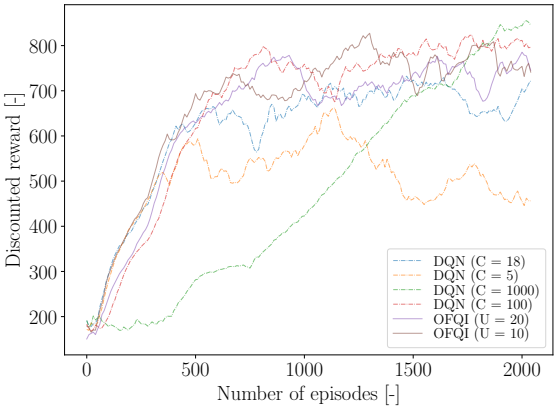
(b) Losses for a classic choice of hyperparameters



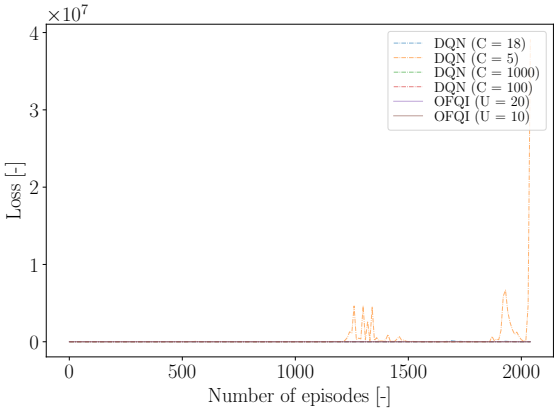
(c) Learning curves for tuned hyperparameters



(d) Losses for tuned hyperparameter

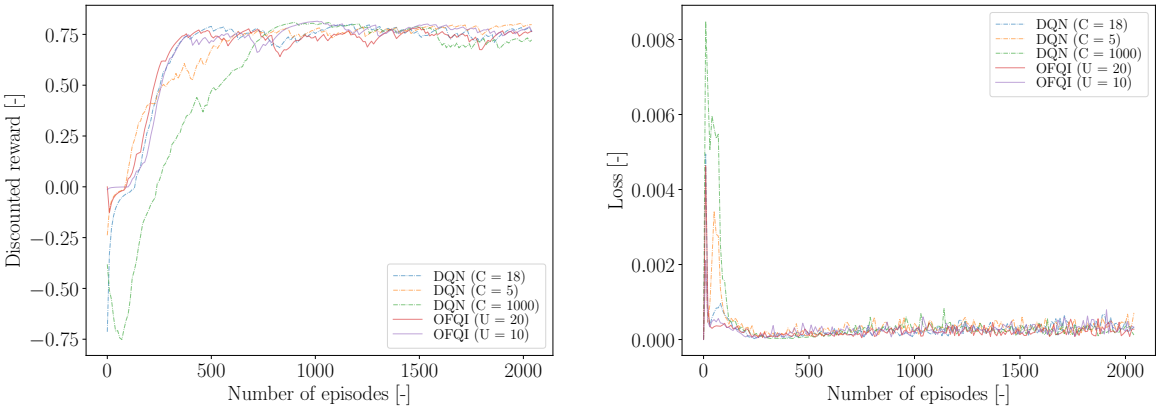


(e) All learning curves



(f) All losses

Figure 7.1: Learning curves and losses for the Double Inverted Pendulum on Cart



(a) Learning curves

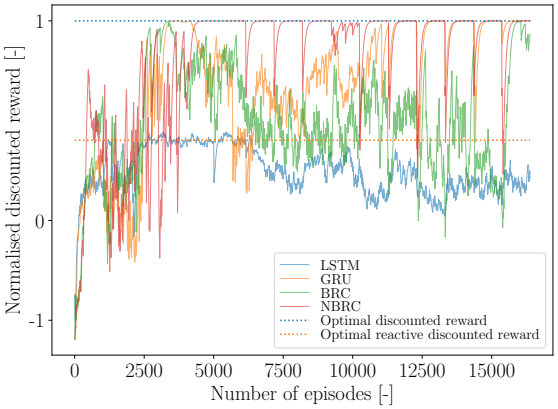
(b) Losses

Figure 7.2: Learning curves and losses for the Car on the Hill

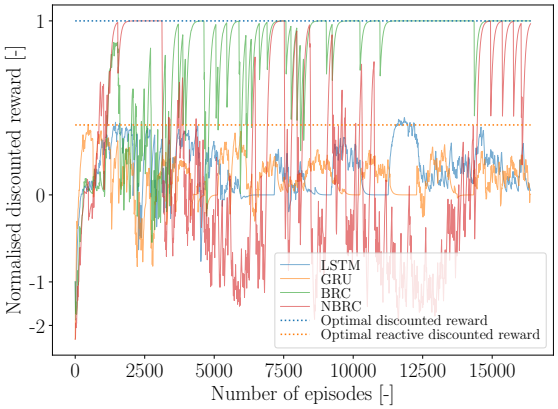
7.4 Online Recurrent Fitted Q-Iteration

As for the DQN whose adaptation to DRQN is rather straightforward, the OFQI algorithm can be adapted to the recurrent case, which results in the Online Recurrent Fitted Q-Iteration (ORFQI). The algorithm is derived from the DRQN algorithm by introducing the same changes that those highlighted in grey in the OFQI (Algorithm 3). During the research project, the ORFQI algorithm was experimented in the Deterministic T-Maze environment. It was compared to the DRQN algorithm described in the previous section with $C = 1000$, a hyperparameter that had been manually tuned on the Deterministic T-Maze environment. As stated in the introduction (Chapter 1), it was chosen to use the DRQN algorithm in the experiments of Chapter 5 and Chapter 6 and not the ORFQI algorithm. As mentioned in Chapter 4, the target update period for the DRQN algorithm was chosen to 25. This aggressive target update period corresponds in fact to the average observed $\bar{C}_{10}$ of the ORFQI algorithm on the different Deterministic T-Maze environments. It can be noted that this update period, automatically tuned by the ORFQI algorithm, is very far from the manually tuned value of $C = 1000$.

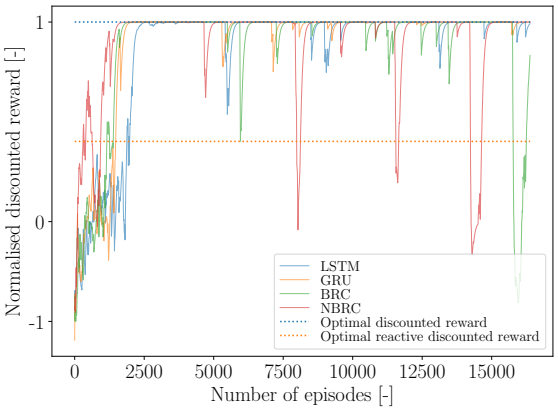
In Figures 7.3, 7.4 and 7.5, one learning curve is displayed for different length of the T-Maze, for the DRQN algorithm with $C = 1000$, for the ORFQI algorithm with $U = 10$ and for the DRQN algorithm with $C = \bar{C}_{10} = 25$. The training sessions have been repeated with each of the four RNNs considered in this work. The results are dramatically better for the ORFQI algorithm. It is hypothesised that the effect of the new target update strategy is even more important in the recurrent case than in the reactive case because of the weight sharing that exists in the RNN. Indeed, as long as the network has not converged, having important losses around certain time steps and performing gradient steps can perturb the outputs at other time steps, because of the weight sharing in the RNN. It probably hurts the stability of the learning process (*i.e.*, the stability of the performance over the course of training). Here, by having an environment with very sparse rewards, rarely updating the target in the DRQN algorithm slows the propagation of these sparse rewards, and thus the convergence. The ORFQI algorithm precisely addresses this problem by allowing a faster propagation of the rewards, and then ultimately a more stable process for learning Q. In addition, it can be seen that the learning curves of the ORFQI are equivalent to those of the tuned DRQN with $C = \bar{C}_{10}$. However, since a single training session has been performed here, this last observation should be taken carefully.



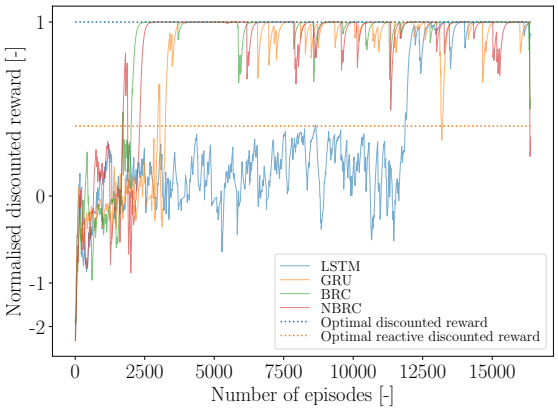
(a) Untuned DRQN with $C = 1000$, $L = 20$



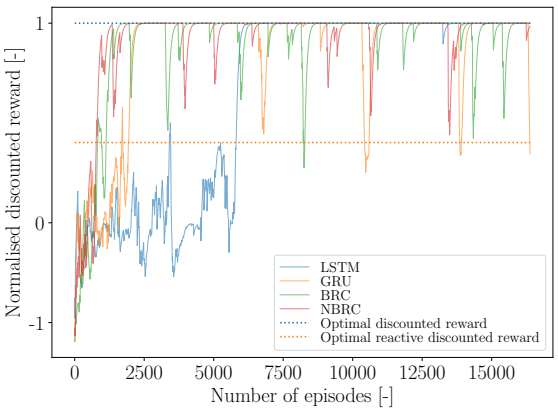
(b) Untuned DRQN with $C = 1000$, $L = 40$



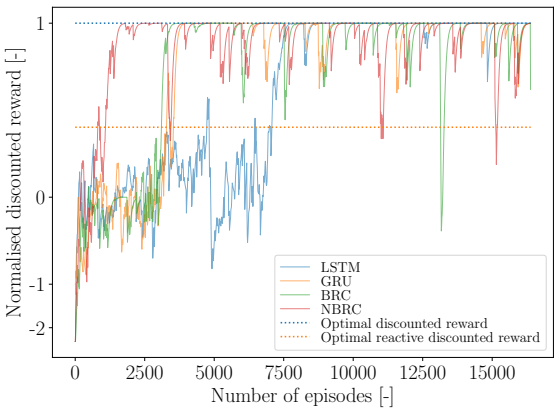
(c) ORFQI with $U = 10$, $L = 20$



(d) ORFQI with $U = 10$, $L = 40$

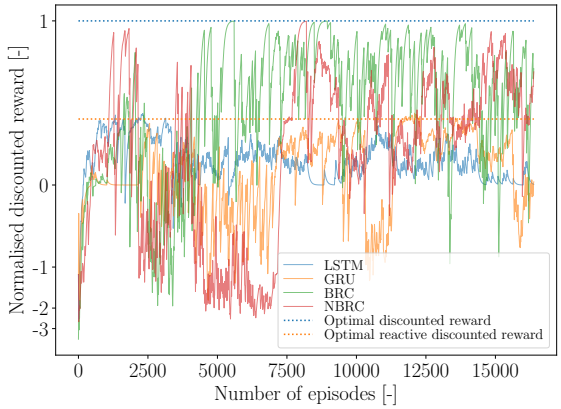


(e) Tuned DRQN with $C = \hat{C}_{10} = 25$, $L = 20$

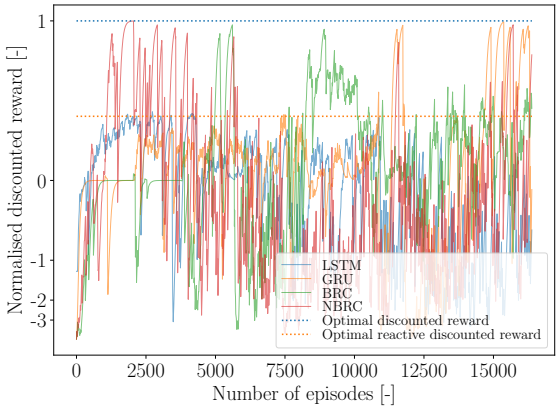


(f) Tuned DRQN with $C = \hat{C}_{10} = 25$, $L = 40$

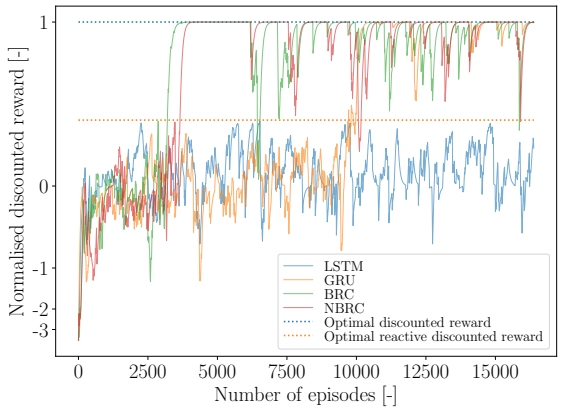
Figure 7.3: Comparison of DRQN and ORFQI for the Deterministic T-Mazes of length $L = 20$ and $L = 40$



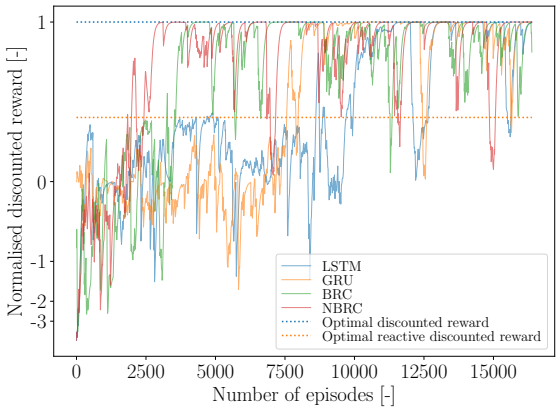
(a) Untuned DRQN with $C = 1000$, $L = 60$



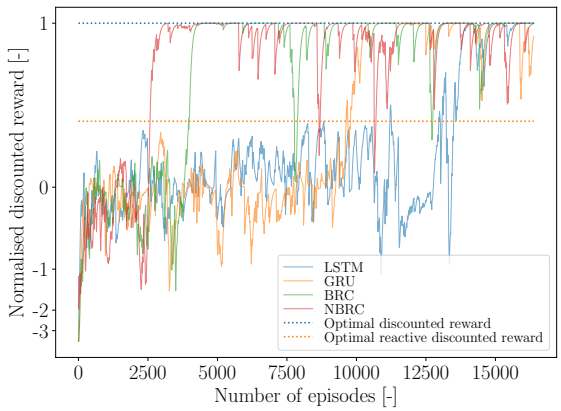
(b) Untuned DRQN with $C = 1000$, $L = 90$



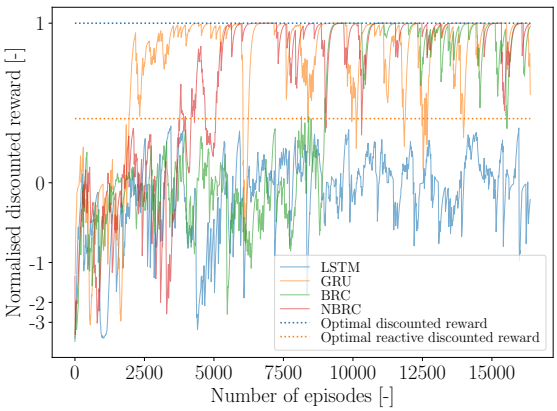
(c) ORFQI with $U = 10$, $L = 60$



(d) ORFQI with $U = 10$, $L = 90$

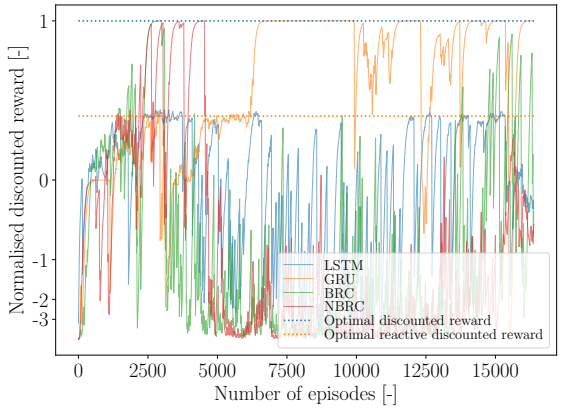


(e) Tuned DRQN with $C = \hat{C}_{10} = 25$, $L = 60$

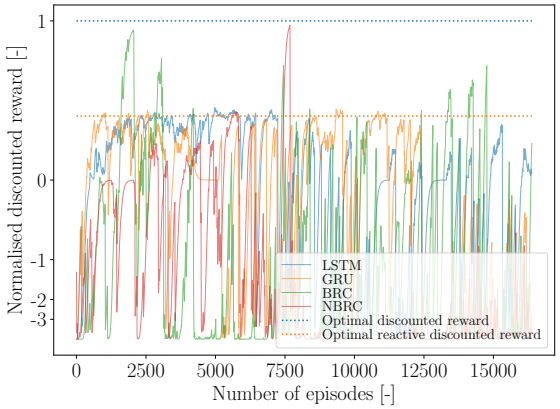


(f) Tuned DRQN with $C = \hat{C}_{10} = 25$, $L = 90$

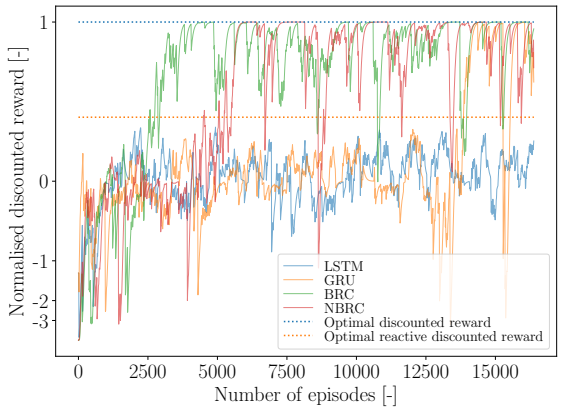
Figure 7.4: Comparison of DRQN and ORFQI for the Deterministic T-Mazes of length $L = 60$ and $L = 90$



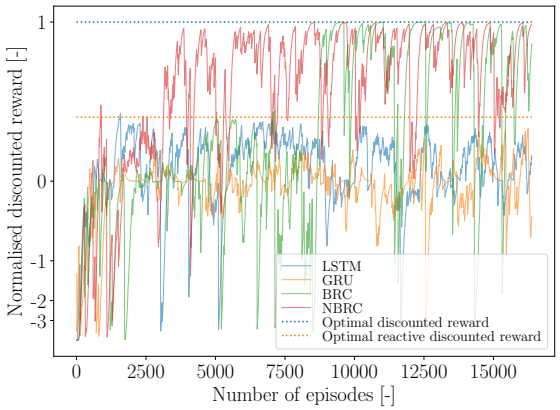
(a) Untuned DRQN with $C = 1000$, $L = 120$



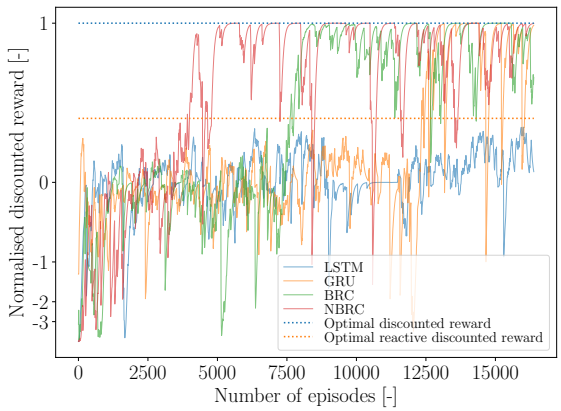
(b) Untuned DRQN with $C = 1000$, $L = 160$



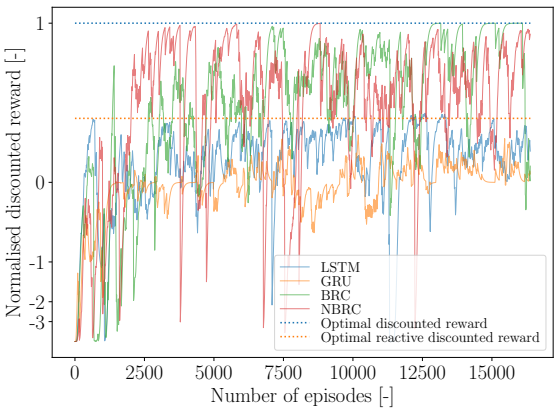
(c) ORFQI with $U = 10$, $L = 120$



(d) ORFQI with $U = 10$, $L = 160$



(e) Tuned DRQN with $C = \hat{C}_{10} = 25$, $L = 120$



(f) Tuned DRQN with $C = \hat{C}_{10} = 25$, $L = 160$

Figure 7.5: Comparison of DRQN and ORFQI for the Deterministic T-Mazes of length $L = 120$ and $L = 160$

Chapter 8

Conclusion

This work tackled the problem of planning in POMDPs when the model is not available. Specifically, it studied the effect of using bistable cells in Q-learning methods for POMDPs. The performances of these cells (BRC and nBRC) were compared on several problems to two standard cells (LSTM and GRU). In addition, the internal states of the RNN and their belief-filtering ability were heavily studied in close relationship with the theory of optimal control. Finally, an independent study on the target update strategy in the DQN algorithm lead to the introduction of a new algorithm, the OFQI algorithm.

8.1 Contributions

First of all, the methods, algorithms, functions and approximators that are used in RNN-based model-free RL for POMDPs have been extensively described and formally defined.

Second, to study the memorisation ability of the bistable cells in the RL setting, a suitable environment has been chosen: the T-Maze environment. Indeed, this environment is simple enough such that the results on the memorisation ability of the agents are not confounded by other difficulties. Furthermore, it allows playing very precisely on the size of the time-dependencies that we want the RNNs to learn. Despite having the most simple environment to measure the memorisation ability of the RNNs was important, it was also interesting to see how the results generalise to increasingly complex environments. This was done by generalising the T-Maze environment to the stochastic case and by introducing two other environments with continuous state spaces, namely the Mountain Hike and Varying Mountain Hike. These environments are far less observable, and the varying one was a difficult problem. This work has shown the superiority of the bistable cells in the RL setting, in terms of rapidity of learning and stability of learning, for environments requiring a high-memorisation ability, such as the T-Mazes. The bistability mechanism was hypothesised to provide this advantage, notably by helping in discovering long-term dependencies. In addition, it has been shown that the bistable cells could compete with the GRU in less memory-demanding but more complex environments, while the LSTM was not as good. Finally, these bistable cells have also been shown to provide a higher speed of learning in a fully observable environment, compared to the other cells and the reactive policy.

Third, a new research question was introduced. Indeed, motivated by the theory of optimal control, the RNNs were studied for their ability to embed the belief filtering throughout their recurrent states. This analysis required the development of several methods for estimating the belief, representing it and estimating the mutual information between the hidden states and the beliefs. The results have been very conclusive for the T-Maze environments, in which the ability to filter the belief seemed clearly correlated with the performance of the agent and the quality of the learned Q-values. Once again, the bistable cells had significantly higher mutual

information between their hidden states and the beliefs in memory-demanding environments. However, we know from the theory of optimal control that belief filtering is a computation that is dependent on the dynamics of the environment. It means that the better ability of the bistable cells to reconstruct the belief is not generalisable to other environments. Indeed, the belief-filtering ability was not particularly better in the Mountain Hike environments. In addition, the generalisation of the policies to other T-Maze lengths was studied. The bistable cells have clearly shown an advantage in comparison to the other cells, especially for the BRC in the stochastic environments.

Fourth, an observation was made on the low speed of convergence of the Q-learning algorithms, especially in sparse reward environments. This slowness, imputed to the target network update, lead to the development of another learning algorithm, adapting the DQN algorithm to the Q-iteration setting, and adapting the Fitted Q-Iteration algorithm to the online-learning setting. This new algorithm, called the Online Fitted Q-Iteration algorithm, has shown itself capable, in simple environments, to achieve better performance than the DQN algorithm when no hyper-parameter tuning is performed. In addition, despite the better performance of this algorithm rely on its ability to automatically finding a good target period update, it was observed that the algorithm was offering more than just an automatic tuning of the target update period. Finally, this algorithm comes with a justification for the existence of the target Q-network.

8.2 Limitations

First of all, despite this first benchmark of the bistable cells in the RL setting is encouraging by showing their ability to represent better $\mathcal{Q}$-function and recurrent policies in memory-demanding environments, it was also observed that they were outperformed by the GRU in some environments. These environments (Mountain Hike and Varying Mountain Hike) were nevertheless not particularly memory-demanding, while the bistable cells have been specifically designed for this purpose.

On the other hand, while the study on the belief-filtering ability of the different recurrent cells was very informative in the case of the T-Maze and the Stochastic T-Maze, the results were not conclusive for the more complex environments. This could be due to the estimation of the belief via particle filtering, or due to its representation by a piecewise linear function over a uniform discretisation of the continuous state space. This could also be linked to the difficulty of estimating the mutual information between continuous variables.

Finally, the OFQI algorithm presented in Chapter 7, despite being motivated theoretically, was not extensively studied empirically. This small study was a preview of the benefits that this algorithm could bring to the value-based RL field. Despite it does not allow to draw concluding observations, it appeals for a more thorough study of this new algorithm. A more complete empirical study of the OFQI algorithm would indeed be a very interesting research project. More precisely, it would be interesting to focus on more complex, high-dimensional environments, where the stability of Q-learning was, until now, addressed by using a huge target update period.

Acknowledgements

I would like to thank Professor Ernst for introducing me to the fascinating field of RL and the exciting world of research. In particular, I would like to thank him for his valuable advice, encouragements and guidance throughout this enriching year.

I would also particularly like to thank Adrien Bolland for the time he devoted to me with his numerous pieces of advice and feedbacks, and for the interesting discussions that we had. I look forward to working with him again.

Finally, I want to thank those who are close to me, my family, my friends and Elif, for having supported me. Above all, I would like to thank them for all the good times that we had during these studies and that we will have in the future.

Bibliography

Spaan, M. T. (2012). Partially observable markov decision processes. In *Reinforcement learning* (pp. 387–414). Springer.

Bakker, B. (2001). Reinforcement learning with long short-term memory. In *Nips* (pp. 1475–1482).

Wierstra, D., Foerster, A., Peters, J., & Schmidhuber, J. (2007). Solving deep memory pomdps with recurrent policy gradients. In *International conference on artificial neural networks* (pp. 697–706). Springer.

Zhang, M., McCarthy, Z., Finn, C., Levine, S., & Abbeel, P. (2016). Learning deep neural network policies with continuous memory states. In *2016 ieee international conference on robotics and automation (icra)* (pp. 520–527). IEEE.

Vecoven, N., Ernst, D., & Drion, G. (2020). A bio-inspired bistable recurrent cell allows for long-lasting memory. *arXiv preprint arXiv:2006.05252*.

Hausknecht, M., & Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*.

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, *32*(11), 1238–1274.

Ernst, D., Glavic, M., & Wehenkel, L. (2004). Power systems stability control: Reinforcement learning framework. *IEEE transactions on power systems*, *19*(1), 427–435.

Vlachogiannis, J. G., & Hatziargyriou, N. D. (2004). Reinforcement learning for reactive power control. *IEEE transactions on power systems*, *19*(3), 1317–1325.

Ernst, D., Glavic, M., Capitanescu, F., & Wehenkel, L. (2008). Reinforcement learning versus model predictive control: A comparison on a power system problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *39*(2), 517–529.

Zheng, G., Zhang, F., Zheng, Z., Xiang, Y., Yuan, N. J., Xie, X., & Li, Z. (2018). Drn: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 world wide web conference* (pp. 167–176).

Kutschinski, E., Uthmann, T., & Polani, D. (2003). Learning competitive pricing strategies by multi-agent reinforcement learning. *Journal of Economic Dynamics and Control*, *27*(11-12), 2207–2218.

Barrett, E., Howley, E., & Duggan, J. (2013). Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, *25*(12), 1656–1674.

Wiering, M., & Van Otterlo, M. (2012). *Reinforcement learning*. Springer.

Moerland, T. M., Broekens, J., & Jonker, C. M. (2020). Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

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

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

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, *101*(1-2), 99–134.

Shankar, T., Dwivedy, S. K., & Guha, P. (2016). Reinforcement learning via recurrent convolutional neural networks. In *2016 23rd international conference on pattern recognition (icpr)* (pp. 2592–2597). IEEE.

Gupta, S., Davidson, J., Levine, S., Sukthankar, R., & Malik, J. (2017). Cognitive mapping and planning for visual navigation. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 2616–2625).

Karkus, P., Hsu, D., & Lee, W. S. (2017). Qmdp-net: Deep learning for planning under partial observability. *arXiv preprint arXiv:1703.06692*.

Heess, N., Hunt, J. J., Lillicrap, T. P., & Silver, D. (2015). Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Igl, M., Zintgraf, L., Le, T. A., Wood, F., & Whiteson, S. (2018). Deep variational reinforcement learning for pomdps. In *International conference on machine learning* (pp. 2117–2126). PMLR.

Hutter, M. (2014). Extreme state aggregation beyond mdps. In *International conference on algorithmic learning theory* (pp. 185–199). Springer.

Majeed, S. J., & Hutter, M. (2018). On q-learning convergence for non-markov decision processes. In *Ijcai* (pp. 2546–2552).

Smallwood, R. D., & Sondik, E. J. (1973). The optimal control of partially observable markov processes over a finite horizon. *Operations research*, *21*(5), 1071–1088.

Pineau, J., Gordon, G., Thrun, S., et al. (2003). Point-based value iteration: An anytime algorithm for pomdps. In *Ijcai* (Vol. 3, pp. 1025–1032). Citeseer.

Porta, J. M., Spaan, M. T., & Vlassis, N. (2004). Value iteration for continuous-state pomdps.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning*. MIT press Cambridge.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Kraskov, A., Stögbauer, H., & Grassberger, P. (2004). Estimating mutual information. *Physical review E*, *69*(6), 066138.

Kozachenko, L., & Leonenko, N. N. (1987). Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, *23*(2), 9–16.

Gao, S., Ver Steeg, G., & Galstyan, A. (2015). Efficient estimation of mutual information for strongly dependent variables. In *Artificial intelligence and statistics* (pp. 277–286). PMLR.

Gao, S., Steeg, G. V., & Galstyan, A. (2015). Estimating mutual information by local gaussian approximation. *arXiv preprint arXiv:1508.00536*.

McAllester, D., & Stratos, K. (2020). Formal limitations on the measurement of mutual information. In *International conference on artificial intelligence and statistics* (pp. 875–884). PMLR.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Zhu, P., Li, X., Poupart, P., & Miao, G. (2017). On improving deep reinforcement learning for pomdps. *arXiv preprint arXiv:1704.07978*.

Boudiaf, M., Rony, J., Ziko, I. M., Granger, E., Pedersoli, M., Piantanida, P., & Ayed, I. B. (2020). A unifying mutual information view of metric learning: Cross-entropy vs. pairwise losses. In *European conference on computer vision* (pp. 548–564). Springer.

Ernst, D., Geurts, P., & Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, *6*, 503–556.

Riedmiller, M. (2005). Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European conference on machine learning* (pp. 317–328). Springer.

Lakshminarayanan, B., Roy, D. M., & Teh, Y. W. (2014). Mondrian forests: Efficient online random forests. *arXiv preprint arXiv:1406.2673*.

Appendix A

Formal description of the environments

In this appendix, the five main environments of this master's thesis are described, namely the Deterministic and Stochastic T-Mazes, the Mountain Hike and Varying Mountain Hike, and the Double Inverted Pendulum on Cart. The Car on the Hill environment, chosen identical to the one described in (Ernst et al., 2005), is not formalised here (see Appendix C.2 of this article for a precise description).

A.1 Deterministic and Stochastic T-Maze

Given a length $L > 1$ and a stochasticity parameter $\lambda \in [0, 1[$, the T-Maze environment defines a POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, p_0)$ with discrete state, observation and action spaces $\mathcal{S}, \mathcal{A}$ and $\mathcal{O}$. The agent can move in this T-shaped tabular environment. Bouncing onto the wall gives a negative reward. In the initial cell, it observes the position of the goal that is either up or down in the bar of the T. Then, it should remember this initial observation to take the correct direction at the crossroad, where it receives a positive reward in this case. The discounted factor defining the value function that we want to maximise is $\gamma = 0.98$. This environment is a generalisation of the Deterministic T-Maze initially proposed by Bakker (2001), the later corresponding to the case where $\lambda = 0$. The environment is represented in Figure A.1.

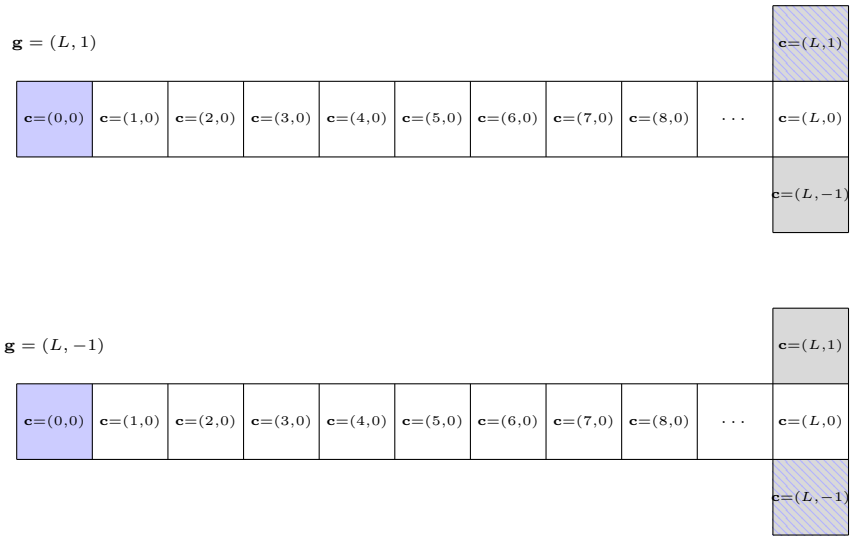


Figure A.1: T-Maze environment - Initial states in blue, terminal states in gray, and goal states hatched

State space The discrete state space $\mathcal{S}$ is composed of the possible cells for the goal $\mathcal{G}$ and the possible cells for the agents $\mathcal{C}$, both are expressed in a Cartesian coordinate system. Formally, we have:

$$\mathcal{S} = \mathcal{G} \times \mathcal{C} \tag{A.1}$$
$$\mathcal{G} = \{(L, 1), (L, -1)\} \tag{A.2}$$
$$\mathcal{C} = \{(0, 0), \ldots, (L, 0)\} \cup \mathcal{G} \tag{A.3}$$

A state $\mathbf{s}_t \in \mathcal{S}$ is thus defined by $\mathbf{s}_t = (\mathbf{g}_t, \mathbf{c}_t)$ with $\mathbf{g}_t \in \mathcal{G}$ the cell in which the goal is and $\mathbf{c}_t \in \mathcal{C}$ the cell in which the agent is. Let us also define $\mathcal{F} = \{\mathbf{s}_t = (\mathbf{g}_t, \mathbf{c}_t) \in \mathcal{S} \mid \mathbf{c}_t \in \mathcal{G}\}$ the set of all terminal states. Terminal states are defined as states from which taking an action does not change the state, and yields a zero reward. Here, these states correspond to the states where the agent is in one of the possible cells of the goal.

Action space The discrete action space $\mathcal{A}$ is composed of the four possible moves that the agent can take:

$$\mathcal{A} = \big\{(1, 0), (0, 1), (-1, 0), \big(0, -1\big)\big\} \tag{A.4}$$

that correspond to *right*, *up*, *left* and *down* respectively.

Observation space The discrete observation space $\mathcal{O}$ is composed of the five partial observations of the state that the agent can perceive:

$$\mathcal{O} = \{\text{up}, \text{down}, \text{corridor}, \text{junction}\} \tag{A.5}$$

Initial state probability distribution function The two possible initial states are $\mathbf{s}_0^{\text{up}} = (\mathbf{g}_0^{\text{up}}, \mathbf{c}_0^{\text{up}}) = ((L, 1), (0, 0))$ and $\mathbf{s}_0^{\text{down}} = (\mathbf{g}_0^{\text{down}}, \mathbf{c}_0^{\text{down}}) = ((L, -1), (0, 0))$, according to the fact that the goal is located *up* or *down* respectively. The initial state is drawn with uniform probability, such that:

$$p_0(\mathbf{s}_0) = \begin{cases} 0.5 & \text{if } \mathbf{s}_0 = \mathbf{s}_0^{\text{up}} \\ 0.5 & \text{if } \mathbf{s}_0 = \mathbf{s}_0^{\text{down}} \\ 0 & \text{otherwise} \end{cases} \tag{A.6}$$

Transition probability distribution function Let us first define the deterministic transition function $f \colon \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. This function defines the transitions in the case where $\lambda = 0$:

$$f(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} \mathbf{s}_{t+1} = (\mathbf{g}_t, \mathbf{c}_t + \mathbf{a}_t) & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{c}_t + \mathbf{a}_t \in \mathcal{C} \\ \mathbf{s}_{t+1} = (\mathbf{g}_t, \mathbf{c}_t) & \text{otherwise} \end{cases} \tag{A.7}$$

with $\mathbf{s}_t = (\mathbf{g}_t, \mathbf{c}_t) \in \mathcal{S}, \mathbf{a}_t \in \mathcal{A}$ and $\mathbf{s}_{t+1} = (\mathbf{g}_{t+1}, \mathbf{c}_{t+1}) \in \mathcal{S}$.

In the general case, the transition probability distribution function $T \colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is defined by the following probabilities:

$$T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) = \begin{cases} 1 & \text{if } \mathbf{s}_t \in \mathcal{F}, \mathbf{s}_{t+1} = \mathbf{s}_t \\ 0 & \text{if } \mathbf{s}_t \in \mathcal{F}, \mathbf{s}_{t+1} \neq \mathbf{s}_t \\ 1 - \lambda + \lambda/4 & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t) \\ \lambda/4 & \text{if } \mathbf{s}_t \notin \mathcal{F}, \exists \mathbf{a} \in \mathcal{A} \setminus \{\mathbf{a}_t\} \text{ s.t. } \mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}) \\ 0 & \text{otherwise} \end{cases} \tag{A.8}$$

with $\mathbf{s}_t = (\mathbf{g}_t, \mathbf{c}_t) \in \mathcal{S}, \mathbf{a}_t \in \mathcal{A}$ and $\mathbf{s}_{t+1} = (\mathbf{g}_{t+1}, \mathbf{c}_{t+1}) \in \mathcal{S}$.

Reward function The reward function $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ can be defined as follows:

$$R(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} 0 & \text{if } \mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_t \notin \mathcal{F}, \mathbf{c}_{t+1} \notin \mathcal{G}, \mathbf{c}_t \neq \mathbf{c}_{t+1} \\ -0.1 & \text{if } \mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_t \notin \mathcal{F}, \mathbf{c}_{t+1} \notin \mathcal{G}, \mathbf{c}_t = \mathbf{c}_{t+1} \\ 4 & \text{if } \mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_t \notin \mathcal{F}, \mathbf{c}_{t+1} \in \mathcal{G}, \mathbf{c}_{t+1} = \mathbf{g}_{t+1} \\ -0.1 & \text{if } \mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_t \notin \mathcal{F}, \mathbf{c}_{t+1} \in \mathcal{G}, \mathbf{c}_{t+1} \neq \mathbf{g}_{t+1} \\ 0 & \text{if } \mathbf{s}_t \in \mathcal{F} \end{cases} \tag{A.9}$$

with $\mathbf{s}_t = (\mathbf{g}_t, \mathbf{c}_t) \in \mathcal{S}, \mathbf{a}_t \in \mathcal{A}$.

Observation probability distribution function The observation probability distribution function $O\colon \mathcal{S} \times \mathcal{O} \to [0, 1]$ is defined by the following probabilities:

$$O(\mathbf{o}_t \mid \mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{o}_t = \text{up}, \mathbf{c}_t = (0, 0), \mathbf{g}_t = (L, 1) \\ 1 & \text{if } \mathbf{o}_t = \text{down}, \mathbf{c}_t = (0, 0), \mathbf{g}_t = (L, -1) \\ 1 & \text{if } \mathbf{o}_t = \text{corridor}, \mathbf{c}_t \in \{(1, 0), \dots, (L-1, 0)\} \\ 1 & \text{if } \mathbf{o}_t = \text{junction}, \mathbf{c}_t \in \{(L, 0), (L, 1), (L, -1)\} \\ 0 & \text{otherwise} \end{cases} \tag{A.10}$$

with $\mathbf{s}_t = (\mathbf{g}_t, \mathbf{c}_t) \in \mathcal{S}$ and $\mathbf{o}_t \in \mathcal{O}$.

A.2 Mountain Hike

The Mountain Hike is a POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, p_0)$ with continuous state and observation spaces and discrete action space. The state of the agent is its position on a two-dimensional map. The agent only has access to a noisy observation of its altitude. Its reward is also equal to the altitude plus some noise. The altitude is always negative, such that there is an incentive to terminate the trajectory as soon as possible. In addition, the states that are at less than a certain distance of the highest point of the map are terminal states. Finally, the agent can take four actions, corresponding to North, West, South and East. Taking an action results in a translation of fixed length in this direction, plus some Gaussian noise. The standard deviation of this Gaussian noise is half the step size. Figure A.2 shows the state space and altitude of this environment. The discounted factor defining the value function that we want to maximise is $\gamma = 0.99$. The environment is thus described by an altitude function $h\colon \mathcal{S} \to \mathbb{R}^- = \mathcal{O}$. This environment was inspired by the Mountain Hike environment proposed by Igl et al. (2018) in which the agent had a noisy information about its position, instead of the altitude only.

State space The continuous state space is the set of position on the two dimensional map $\mathcal{S} = [-1, 1]^2 \subset \mathbb{R}^2$. The set of terminal states is $\mathcal{F} = \{\mathbf{s} \in \mathcal{S} \mid ||\mathbf{s} - (0.8, 0.8)|| < 0.1\}$.

Action space The discrete action space $\mathcal{A}$ is composed of the four possible moves that the agent can take:

$$\mathcal{A} = \big\{(0.1, 0), (0, 0.1), (-0.1, 0), \big(0, -0.1\big)\big\} \tag{A.11}$$

that correspond to *North*, *West*, *Down* and *East* respectively.

Observation space The continuous observation space is $\mathcal{O} = \mathbb{R}^-$.

Initial state probability distribution function The initial state is always $\mathbf{s} = (-0.8, -0.8)$, such that:

$$p_0(\mathbf{s}_0) = \begin{cases} 1 & \text{if } \mathbf{s}_0 = (-0.8, -0.8) \\ 0 & \text{otherwise} \end{cases} \tag{A.12}$$

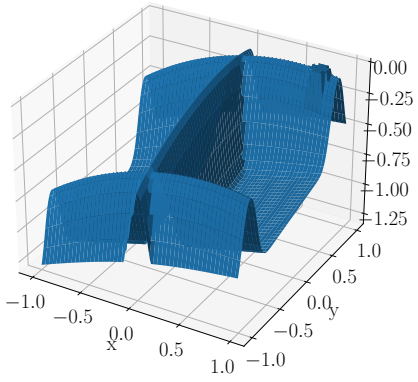


Figure A.2: Mountain hike altitude function h

Transition probability distribution function The transition probability distribution function $T\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is defined by the following probabilities:

$$T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) = \begin{cases} 1 & \text{if } \mathbf{s}_t \in \mathcal{F}, \mathbf{s}_{t+1} = \mathbf{s}_t \\ 0 & \text{if } \mathbf{s}_t \in \mathcal{F}, \mathbf{s}_{t+1} \neq \mathbf{s}_t \\ \phi(\mathbf{s}_{t+1} \mid \mathbf{s}_t + \mathbf{a}_t, 0.05^2) & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} \in]-1, 1[^2 \\ \int_{-\infty}^{-1} \phi(\mathbf{s}_{t+1} + (x, 0) \mid \mathbf{s}_t + \mathbf{a}_t, 0.05^2) dx & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} \in \{-1\} \times]-1, 1[\\ \int_{1}^{\infty} \phi(\mathbf{s}_{t+1} + (x, 0) \mid \mathbf{s}_t + \mathbf{a}_t, 0.05^2) dx & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} \in \{1\} \times]-1, 1[\\ \int_{-\infty}^{-1} \phi(\mathbf{s}_{t+1} + (0, y) \mid \mathbf{s}_t + \mathbf{a}_t, 0.05^2) dy & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} \in]-1, 1[\times \{-1\} \\ \int_{1}^{\infty} \phi(\mathbf{s}_{t+1} + (0, y) \mid \mathbf{s}_t + \mathbf{a}_t, 0.05^2) dy & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} \in]-1, 1[\times \{1\} \\ \int_{-\infty}^{-1} \int_{-\infty}^{-1} \phi(\mathbf{s}_{t+1} + (x, y) \mid \mathbf{s}_t + \mathbf{a}_t, 0.05^2) dx dy & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} = (-1, -1) \\ \int_{1}^{\infty} \int_{-\infty}^{-1} \phi(\mathbf{s}_{t+1} + (x, y) \mid \mathbf{s}_t + \mathbf{a}_t, 0.05^2) dx dy & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} = (-1, 1) \\ \int_{-\infty}^{-1} \int_{1}^{\infty} \phi(\mathbf{s}_{t+1} + (x, y) \mid \mathbf{s}_t + \mathbf{a}_t, 0.05^2) dx dy & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} = (1, -1) \\ \int_{1}^{\infty} \int_{1}^{\infty} \phi(\mathbf{s}_{t+1} + (x, y) \mid \mathbf{s}_t + \mathbf{a}_t, 0.05^2) dx dy & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} = (1, 1) \\ 0 & \text{otherwise} \end{cases} \tag{A.13}$$

where $\mathbf{s}_t \in \mathcal{S}, \mathbf{a}_t \in \mathcal{A}$ and $\mathbf{s}_{t+1} \in \mathcal{S}$, with $\phi(\mathbf{s}_{t+1} \mid \mu, \sigma^2)$ that denotes the probability density function of a bivariate normal distribution with mean μ and diagonal covariance matrix of standard deviation σ. This complicated expression simply means that the next state $\mathbf{s}_{t+1}$ corresponds to $\mathbf{s}_t + \mathbf{a}_t + \mathcal{N}(0, 0.05^2)$, except when it corresponds to a point outside of $S = [-1, 1]^2$, in this case, $\mathbf{s}_{t+1}$ is the closest point to $\mathbf{s}_t + \mathbf{a}_t + \mathcal{N}(0, 0.05^2)$ that lies inside $\mathcal{S}$.

Reward function The reward function $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ can be defined as follows:

$$R(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} 0 & \text{if } \mathbf{s}_t \in \mathcal{F} \\ h(\mathbf{s}_t) + \mathcal{N}(0, 0.1^2) & \text{otherwise} \end{cases} \tag{A.14}$$

with $\mathbf{s}_t \in \mathcal{S}, \mathbf{a}_t \in \mathcal{A}$. This reward function can be visualised in Figure A.3.

Observation probability distribution function The observation probability distribution function $O\colon \mathcal{S} \times \mathcal{O} \to [0, 1]$ is defined by the following probabilities:

$$O(\mathbf{o}_t \mid \mathbf{s}_t) = \phi(h(\mathbf{s}_t), 0.1^2) \tag{A.15}$$

with $\mathbf{s}_t \in \mathcal{S}$ and $\mathbf{o}_t \in \mathcal{O}$, and where ϕ denotes the probability density function of a Gaussian random variable.

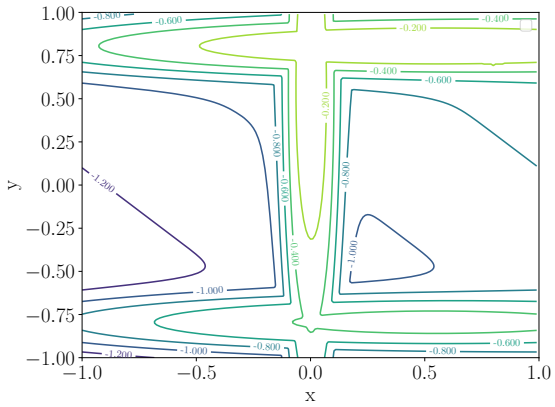


Figure A.3: Expected immediate rewards obtained in the state space $\mathcal{S}$ of the Mountain Hike environment

A.3 Varying Mountain Hike

Despite being a difficult problem, the Mountain Hike can be made even harder by introducing variations of the mountainous terrain. This problem is called the Varying mountain hike. As for the classic mountain hike, only the altitude is observed, but at each environment initialisation, one terrain is from one out of four variations of the original terrain. Note that the information about which terrain has been drawn is not given to the agent, it is only able to deduce it from the altitude observations. The four variations are constituted of the four rotations of the original terrain around $C = (0, 0)$. Note that the initial position of the agent and the position of the terminal states are rotated along with the terrain.

The environment description can be straightforwardly adapted from the Mountain Hike environment by adding a variable identifying the terrain $i \in \{1, \dots, 4\}$ in the state. This variable is sampled uniformly in p_0, and never change afterwards. The observation and reward functions are now given by $h_i(\mathbf{s}_t)$ instead of $h(\mathbf{s}_t)$, where $h_1, \dots, h_4$ denote the four variations of the terrain. Since this environment is harder, the Gaussian noise on the transitions is chosen to have a standard deviation of 0.02 instead of 0.05 as for the Mountain Hike.

A.4 Double Inverted Pendulum on Cart

The Double Inverted Pendulum on Cart is an MDP $(\mathcal{S}, \mathcal{A}, T, R, p_0)$ that we solve for a reward discounted by $\gamma = .99$. The double inverted pendulum starts in the upright position, and the agent is responsible for applying force on the cart. The agent observes the state of the physical system and has to take actions such that the environment does not reach a terminal state and it stays the closest to the upright position. The cart horizontal position cannot move outside of $[-1, 1]$.

State space The state space $\mathcal{S} = [0, 1] \times \mathbb{R} \times [0, 2\pi] \times \mathbb{R} \times [0, 2\pi] \times \mathbb{R}$ is composed of the following components:

- x, the horizontal position of the cart
- $\dot{x}$, the horizontal velocity of the cart
- θ_1, the angle between the upright and the lower pole
- $\dot{\theta_1}$, the angular velocity of the lower pole

- θ_2, the angle between the upright and the upper pole
- $\dot{\theta_2}$, the angular velocity of the upper pole

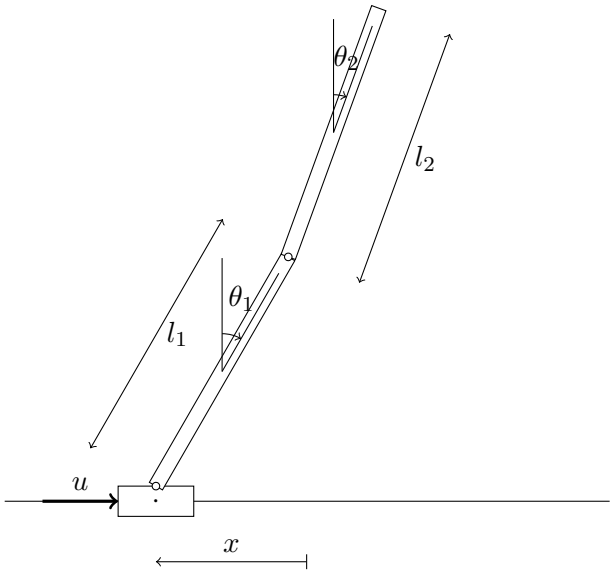


Figure A.4: Double inverted pendulum on cart environment

We can also note that some states are terminal state, these states are reached when the double pendulum is a less than 70% of its maximal height. More precisely, denoting by (x_2, y_2) the position of the upper tip, a terminal state is reached when

$$y_2 = l_1 \cos\theta_1 + l_2 \cos\theta_2 \leq 0.7 \tag{A.16}$$

with $l_1 = l_2 = 0.5$.

Action space The action space is a 1-dimensional space $\mathcal{A} = [-1, 1]$, defining the fraction of the maximum force that is applied to the cart. This action space is discretised in practice to make it compatible with Q-learning.

Dynamics The dynamics are not covered here, but they follow the laws of motion from the Lagrangian mechanics. Note that the joints have damping such that the energy brought by the work of the force $\mathbf{a}$ is dissipated by the system.

Reward function The reward function $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined in the following way:

$$R(\mathbf{s}_t, \mathbf{a}_t) = 10 + (0.01\ x_2)^2 + (y_2 + 0.3 - 2)^2 \tag{A.17}$$

where $\mathbf{s}_t = (x, \dot{x}, \theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$, and $(x_2, y_2) = (l_1 \sin\theta_1 + l_2 \sin\theta_2, l_1 \cos\theta_1 + l_2 \cos\theta_2)$.

Appendix B

Piecewise linearity and convexity of the value function in the discrete case

The piecewise linear and convex (PWLC) nature of the value function in POMDPs has been proved in (Smallwood & Sondik, 1973) initially. This proof is detailed and explained in this appendix. As explained in Section 3.2, the horizon $n = 1$ value is given by $V_1(b_t) = \max_{\mathbf{a}_t \in \mathcal{A}} \sum_{\mathbf{s}_t \in \mathcal{S}} b_t(\mathbf{s}_t) R(\mathbf{s}_t, \mathbf{a}_t)$. As a consequence, the value function V_1 is the maximum of $|\mathcal{A}|$ linear functions, which corresponds to a PWLC function. It can be verified that V_1 has the form of equation (3.25) with $K_1 = |\mathcal{A}|$. Precisely, V_1 is defined by the following set of α-vector:

$$\alpha_{1,k} = \left(R(\mathbf{s}_t^{(1)}, \mathbf{a}^{(k)}) \quad \dots \quad R(\mathbf{s}_t^{(|\mathcal{S}|)}, \mathbf{a}^{(k)}) \right), \; k = 1, \dots, K_1 \; . \tag{B.1}$$

where $\mathbf{a}^{(k)}$ denotes the k^{th} action and $\mathbf{s}^{(i)}$ the i^{th} state.

For $n > 1$, V_n is computed from V_{n-1} by:

$$V_n(b_t) = \max_{\mathbf{a}_t \in \mathcal{A}} \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim b_t \\ \mathbf{s}_{t+1} \sim T(\cdot|\mathbf{s}_t, \mathbf{a}_t) \\ \mathbf{o}_{t+1} \sim O(\cdot|\mathbf{s}_t)}} \{ r_t + \gamma V_{n-1}(b_{t+1}) \} \tag{B.2}$$

$$= \max_{\mathbf{a}_t \in \mathcal{A}} R(b_t, \mathbf{a}_t) + \gamma \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim b_t \\ \mathbf{s}_{t+1} \sim T(\cdot|\mathbf{s}_t, \mathbf{a}_t)}} \left\{ \sum_{\mathbf{o}_{t+1} \in \mathcal{O}} O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1}) V_{n-1}(b_{t+1}) \right\} \tag{B.3}$$

$$= \max_{\mathbf{a}_t \in \mathcal{A}} R(b_t, \mathbf{a}_t) + \gamma \sum_{\mathbf{o}_{t+1} \in \mathcal{O}} \Pr(\mathbf{o}_{t+1} \mid b_t, \mathbf{a}_t) V_{n-1}(b_{t+1}) \tag{B.4}$$

where $b_{t+1} = \text{forward}(b_t; \mathbf{a}_t, \mathbf{o}_{t+1})$ and:

$$\Pr(\mathbf{o}_{t+1} \mid b_t, \mathbf{a}_t) = \sum_{\mathbf{s}_t \in \mathcal{S}} \sum_{\mathbf{s}_{t+1} \in \mathcal{S}} b_t(\mathbf{s}_t) T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1}) \; . \tag{B.5}$$

It can be noticed that V_n is the maximum over any action $\mathbf{a}_t$, of a function of two terms. The first term is $R(b_t, \mathbf{a}_t)$ that is linear (and thus PWLC) over the belief. The second term is a weighted sum of V_{n-1} evaluated at each possible next belief b_{t+1}, resulting from each possible next observation. If this second term is PWLC over the belief space of b_t, it is thus straightforward that V_n is PWLC, since it is the maximum of linear combinations of PWLC functions. It should be understand that we do not need only V_{n-1} to be PWLC in its argument, but we need $V_{n-1}(b_{t+1}) = V_{n-1}(\text{forward}(b_t; \mathbf{a}_t, \mathbf{o}_{t+1}))$ to be PWLC in b_t.

And it can indeed be proven that this second term is PWLC in the belief b_t. It is proved by induction, knowing that V_1 is PWLC in its argument. Assuming that V_{n-1} is PWLC in its

argument (b_{t+1}), we can write:

$$V_{n-1}(b_{t+1}) = \max_{k\in\{1,\dots,K_{n-1}\}} \alpha_{n-1,k} \cdot b_{t+1} \tag{B.6}$$

$$= \max_{k\in\{1,\dots,K_{n-1}\}} \sum_{\mathbf{s}_{t+1}\in\mathcal{S}} \alpha_{n-1,k}^{(\mathbf{s}_{t+1})} \frac{\sum_{\mathbf{s}_t\in\mathcal{S}} b_t(\mathbf{s}_t) T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1})}{\sum_{\mathbf{s}_t\in\mathcal{S}} \sum_{\mathbf{s}_{t+1}\in\mathcal{S}} b_t(\mathbf{s}_t) T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1})} \tag{B.7}$$

$$= \sum_{\mathbf{s}_{t+1}\in\mathcal{S}} \alpha_{n-1,l(b_t,\mathbf{a}_t,\mathbf{o}_{t+1})}^{(\mathbf{s}_{t+1})} \frac{\sum_{\mathbf{s}_t\in\mathcal{S}} b_t(\mathbf{s}_t) T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1})}{\sum_{\mathbf{s}_t\in\mathcal{S}} \sum_{\mathbf{s}_{t+1}\in\mathcal{S}} b_t(\mathbf{s}_t) T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1})} \tag{B.8}$$

$$= \sum_{\mathbf{s}_t\in\mathcal{S}} b_t(\mathbf{s}_t) \sum_{\mathbf{s}_{t+1}\in\mathcal{S}} \alpha_{n-1,l(b_t,\mathbf{a}_t,\mathbf{o}_{t+1})}^{(\mathbf{s}_{t+1})} \frac{T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1})}{\sum_{\mathbf{s}_t\in\mathcal{S}} \sum_{\mathbf{s}_{t+1}\in\mathcal{S}} b_t(\mathbf{s}_t) T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1})} \tag{B.9}$$

where $l(b_t, \mathbf{a}_t, \mathbf{o}_{t+1}) \in \{0, \dots, K_{n-1}\}$ gives the index of the α-vector supporting V_{n-1} at $b_{t+1} = \text{forward}(b_t, \mathbf{a}_t, \mathbf{o}_{t+1})$. By substituting (B.9) and (B.5) into (B.4), we get:

$$\begin{aligned} V_n(b_t) = &\max_{\mathbf{a}_t\in\mathcal{A}} R(b_t, \mathbf{a}_t) + \gamma \sum_{\mathbf{o}_{t+1}\in\mathcal{O}} \\ &\sum_{\mathbf{s}'_t\in\mathcal{S}} \sum_{\mathbf{s}'_{t+1}\in\mathcal{S}} b_t(\mathbf{s}'_t) T(\mathbf{s}'_{t+1} \mid \mathbf{s}'_t, \mathbf{a}_t) O(\mathbf{o}_{t+1} \mid \mathbf{s}'_{t+1}) \sum_{\mathbf{s}_t\in\mathcal{S}} b_t(\mathbf{s}_t) \\ &\sum_{\mathbf{s}_{t+1}\in\mathcal{S}} \alpha_{n-1,l(b_t,\mathbf{a}_t,\mathbf{o}_{t+1})}^{(\mathbf{s}_{t+1})} \frac{T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1})}{\sum_{\mathbf{s}_t\in\mathcal{S}} \sum_{\mathbf{s}_{t+1}\in\mathcal{S}} b_t(\mathbf{s}_t) T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1})} \end{aligned} \tag{B.10}$$

$$= \max_{\mathbf{a}_t\in\mathcal{A}} R(b_t, \mathbf{a}_t) + \gamma \sum_{\mathbf{s}_t\in\mathcal{S}} b_t(\mathbf{s}_t) \sum_{\mathbf{o}_{t+1}\in\mathcal{O}} \sum_{\mathbf{s}_{t+1}\in\mathcal{S}} \alpha_{n-1,l(b_t,\mathbf{a}_t,\mathbf{o}_{t+1})}^{(\mathbf{s}_{t+1})} T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) O(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1}) \tag{B.11}$$

In this last equation, the second term of (B.4) has been rewritten as the dot product of b_t with another vector. This vector gives, for every $\mathbf{s}_t$, a weighted sum (over all possible $\mathbf{s}_{t+1}$ and $\mathbf{o}_{t+1}$) of the values that support the α-vector that maximises V_{n-1} at $b_{t+1} = \text{forward}(\mathbf{b}_t(\cdot \mid \mathbf{s}_t); \mathbf{a}_t, \mathbf{o}_{t+1})$, with $\mathbf{b}_t(\cdot \mid \mathbf{s}_t)$ that denotes the belief assigning probability 1 to state $\mathbf{s}_t$. It has the consequence of making this second term a linear combination with weights b_t, making it linear with respect to b_t on each piece having a different $l(b_t, \mathbf{a}_t, \mathbf{o}_{t+1})$. In addition, since the α-vector selected in each element of the weighted sum is chosen to be the maximising α-vector, by definition of l, V_{n-1} naturally describes a convex function. As explained before, it proves the piecewise linearity and convexity of $V_n(b_t)$ since it is the maximum over $\mathbf{a}_t$ of linear combinations of $R(b_t, \mathbf{a}_t)$, PWLC in b_t, and $V_{n-1}(b_{t+1})$ evaluated at several $b_{t+1} = b_{t+1}(\cdot \mid \mathbf{a}_t, \mathbf{o}_{t+1})$, which is also PWLC in b_t.

Appendix C

Training hyperparameters

The general training hyperparameters for the DRQN algorithm are those of Table C.1. It is noteworthy that they have thus be chosen identical for all environments. The hyperparameters that are not displayed here have been discussed directly in the different sections.

Description	Name	Value
Number of stacked RNNs	S	2
RNNs hidden size	H	32
Exploration rate	ε	0.2
Replay buffer capacity	N	65 536
Mini batch size for updates	B	128
Target update period	C	25
Adam learning rate	α	5×10^{-4}
Evaluation period	M	10
Number of Monte Carlo rollouts	R	50

Table C.1: General hyperparameters

Only the number of generated episodes has changed from one environment to another. These different numbers of episodes E are reported in Table C.2. It can be noted that the number of episodes is different for the Double Inverted Pendulum on Cart when used in the DRQN algorithm and when used in the OFQI and DQN algorithms. It comes from the fact that a single gradient step is performed at each generated episode in this environment, while one gradient step is performed at each generated transition in the DQN and OFQI algorithms.

Environment	E
Deterministic T-Maze	16 384
Stochastic T-Maze	16 384
Mountain Hike	16 384
Varying Mountain Hike	32 768
Double Inverted Pendulum on Cart (with DRQN)	16 384
Double Inverted Pendulum on Cart (with OFQI and DQN)	2048
Car on the Hill (with OFQI and DQN)	2048

Table C.2: Number of episodes generated for each environment

In Table C.3, the time horizon T used in each environment is given. In addition, when BPTT

was truncated after a few time steps, this window size W is indicated.

Environment	T	W
Deterministic T-Maze ($L = 20$)	59	-
Deterministic T-Maze ($L = 40$)	119	-
Deterministic T-Maze ($L = 60$)	179	-
Deterministic T-Maze ($L = 90$)	269	-
Deterministic T-Maze ($L = 120$)	359	-
Deterministic T-Maze ($L = 160$)	479	-
Deterministic T-Maze ($L = 200$)	599	-
Deterministic T-Maze ($L = 240$)	719	-
Stochastic T-Maze ($\lambda = 0.4, L = 20$)	99	-
Stochastic T-Maze ($\lambda = 0.4, L = 40$)	199	-
Stochastic T-Maze ($\lambda = 0.4, L = 60$)	299	-
Stochastic T-Maze ($\lambda = 0.4, L = 90$)	449	-
Stochastic T-Maze ($\lambda = 0.4, L = 120$)	599	-
Mountain Hike	80	-
Varying Mountain Hike	80	-
Double Inverted Pendulum on Cart (with DRQN)	1000	8
Double Inverted Pendulum on Cart (with OFQI and DQN)	1000	/
Car on the Hill (with OFQI and DQN)	1000	/

Table C.3: Time horizon for each environment