Deep Reinforcement Learning with Temporal Logics

Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate

Department of Computer Science, University of Oxford
{hosein.hasanbeig, daniel.kroening, alessandro.abate}@cs.ox.ac.uk

Abstract. The combination of data-driven learning methods with formal reasoning has seen a surge of interest, as either area has the potential to bolstering the other. For instance, formal methods promise to expand the use of state-of-the-art learning approaches in the direction of certification and sample efficiency. In this work, we propose a deep Reinforcement Learning (RL) method for policy synthesis in continuous-state/action unknown environments, under requirements expressed in Linear Temporal Logic (LTL). We show that this combination lifts the applicability of deep RL to complex temporal and memory-dependent policy synthesis goals. We express an LTL specification as a Limit Deterministic Büchi Automaton (LDBA) and synchronise it on-the-fly with the agent/environment. The LDBA in practice monitors the environment, acting as a modular reward machine for the agent: accordingly, a modular Deep Deterministic Policy Gradient (DDPG) architecture is proposed to generate a low-level control policy that maximises the probability of the given LTL formula. We evaluate our framework in a cart-pole example and in a Mars rover experiment, where we achieve near-perfect success rates, while baselines based on standard RL are shown to fail in practice.

Keywords: Model-Free Reinforcement Learning · Deep Learning · Linear Temporal Logic · Continuous-State and Continuous-Action Markov Decision Processes

1 Introduction

Deep Reinforcement Learning (RL) is an emerging paradigm for autonomous decision-making tasks in complex and unknown environments. Deep RL has achieved impressive results over the past few years, but often the learned solution is evaluated only by statistical testing and there is no systematic method to guarantee that the policy synthesised using RL meets the expectations of the designer of the algorithm. This particularly becomes a pressing issue when applying RL to safety-critical systems.

Furthermore, tasks featuring extremely sparse rewards are often difficult to solve by deep RL if exploration is limited to low-level primitive action selection. Despite its generality, deep RL is not a natural representation for how humans perceive sparse reward problems: humans already have prior knowledge and associations regarding elements and their corresponding function in a given environment, e.g. “keys open doors” in video games. Given useful domain knowledge and associations, a human expert can find solutions to problems involving sparse rewards, e.g. when the rewards are only received when a task is eventually fulfilled (e.g., finally unlocking a door). These assumed high-level associations can provide initial knowledge about the problem, whether in abstract video games or in numerous real world applications, to efficiently find the global optimal policies, while avoiding an exhaustive unnecessary exploration, particularly in early stages.

The idea of separating the learning process into two (or more) synchronised low- and high-level learning steps has led to hierarchical RL, which specifically targets sparse reward problems [49]. Practical approaches in hierarchical RL, e.g. options [43], depend on state representations and on the underlying problem simplicity and/or structure, such that suitable reward signals can be effectively engineered by hand. These methods often require detailed supervision in the form of explicitly specified high-level actions or intermediate supervisory signals [2, 9, 29, 30, 43, 52]. Furthermore, most hierarchical RL approaches either only work in discrete domains, or require pre-trained low-level controllers. HAC [32], a state-of-the-art method in hierarchical RL for continuous-state/action Markov Decision Processes (MDPs), introduces the notion of Universal MDPs, which have an augmented state space that is obtained by a set of strictly sequential goals.

This contribution extends our earlier work [19, 20, 22, 57] and proposes a one-shot[1] and online deep RL framework, where the learner is presented with a modular high-level mission task over a continuous-state/action MDP. Unlike hierarchical RL, the mission task is not limited to sequential tasks, and is instead specified as a Linear Temporal Logic (LTL) formula, namely a formal, ungrounded, and high-level representation of the task and of its components. Without requiring any supervision, each component of the LTL property systematically structures a complex mission task into partial task “modules”. The LTL property essentially acts as a high-level exploration guide for the agent, where low-level planning is handled by a deep RL architecture. LTL is a temporal logic that allows to formally express tasks and constraints in interpretable form: there exists a substantial body of research on extraction of LTL properties from natural languages [16, 38, 55].

We synchronise the high-level LTL task with the agent/environment: we first convert the LTL property to an automaton, namely a finite-state machine accepting sequences of symbols [3]; we then construct on-the-fly[2] a synchronous product between the automaton and the agent/environment; we also define a reward function based on the structure of the automaton. With this automated reward-shaping procedure, an RL agent learns to accomplish the LTL task with max probability, and with no supervisory assistance: this is in general hard, if at all

[1] One-shot means that there is no need to master easy tasks first, then compose them together to accomplish a more complex tasks.

[2] On-the-fly means that the algorithm tracks (or executes) the state of an underlying structure (or a function) without explicitly constructing it.

possible, by conventional or handcrafted RL reward shaping methods [43, 49, 52]. Furthermore, as elaborated later, the structure of the product partitions the state space of the MDP, so that partitions are solely relevant to specific task modules. Thus, when dealing with sparse-reward problems, the agent's low-level exploration is efficiently guided by task modules, saving the agent from exhaustively searching through the whole state space.

Related Work The closest lines of work comprise model-based [4, 8, 12, 13, 14, 15, 24, 25, 44, 42] and model-free [7, 11, 17, 27, 28, 51] RL approaches, aiming to synthesise policies abiding by a given temporal logic property. In model-based RL, a model of the MDP is firstly inferred and later an optimal policy is generated over the learned model. This approach is known to hardly scale to large-dimensional problems, which are in practice studied with model-free RL. Additionally, in standard work on RL for LTL, formulae are translated to Deterministic Rabin Automata (DRA), which are known to be doubly exponential in the size of the original LTL formula. Conversely, in this work we use a specific Limit Deterministic Büchi Automaton (LDBA) [45], which we have employed in the context of RL in [18]: this is only an exponential-sized automaton for LTL\GU (a fragment of LTL), and has otherwise the same size as DRA for the rest of LTL. This can significantly enhance the convergence rate of RL. Other variants of LDBAs have been employed in cognate work [7, 17, 28, 39].

Another closely-related line of work is the "curriculum learning" approach [2], in which the agent masters easier instruction-based sub-tasks first, to then compose them together in order to accomplish a more complex task. In this work, instead, the complex task is expressed as an LTL property, which guides learning and directly generates policies: it thus has no need to start from easier sub-tasks and to later compose corresponding policies together. In other words, the proposed method learns policies for the general complex task in a "one-shot" scenario.

To the best of authors' knowledge, no research has so far enabled model-free RL to generate policies for general LTL properties over continuous-state/action MDPs: relevant results are applicable to finite MDPs [8, 11, 12, 17, 21, 44, 51], or are focused on sub-fragments of LTL [10, 23, 26, 31, 33, 42], such as finite-horizon formulae. Many practical problems require continuous, real-valued actions to be taken in over uncountable state variables: the simplest approach to solve such problems is to discretise state and action spaces of the MDP [1]. However, beyond requiring the knowledge of the MDP itself, discretisation schemes are expected to suffer from the trade off between accuracy and curse of dimensionality.

Contributions To tackle the discussed issues and push the envelope of state of the art in RL, in this work and propose a modular Deep Deterministic Policy Gradient (DDPG) based on [34, 47]. This modular DDPG is an actor-critic architecture that uses deep function approximators, which can learn policies in continuous state and action spaces, optimising over task-specific LTL satisfaction probabilities. The contributions of this work are as follows:

- We deal with continuous-state/action, unknown MDPs. The proposed model-free RL algorithm significantly increases the applicability of RL for LTL synthesis.
- The use of LTL (and associated LDBAs) with deep RL allows us to efficiently solve problems with sparse rewards, by exploiting relationships between sub-tasks. Rewards are automatically assigned on-the-fly with no supervision, allowing one to automatically modularise a global complex task into easy sub-tasks.
- The use of LDBA in DDPG introduces technical issues to the learning process, such as non-determinism, which are addressed in this work.

2 Problem Framework

The environment with which the agent interacts is assumed to be an unknown black-box. We describe the underlying agent/environment model as an MDP, however we emphasise that the MDP is unknown and the learning agent is unaware of the transition (i.e., the dynamics) and the spatial labels (environmental features grounded to tasks). This works assumes that the dynamics of the interaction are Markovian, namely memory-less.

Definition 1 (General MDP [5]). *The tuple $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ is a general MDP over a set of continuous states $\mathcal{S} = \mathbb{R}^n$, where $\mathcal{A} = \mathbb{R}^m$ is a set of continuous actions, and $s_0 \in \mathcal{S}$ is the initial state. $P : \mathcal{B}(\mathbb{R}^n) \times \mathcal{S} \times \mathcal{A} \to [0,1]$ is a Borel-measurable conditional transition kernel which assigns to any pair of state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ a probability measure $P(\cdot|s,a)$ on the Borel space $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$, where $\mathcal{B}(\mathbb{R}^n)$ is the set of all Borel sets on $\mathbb{R}^n$. $\mathcal{AP}$ is a finite set of atomic propositions and a labelling function $L : \mathcal{S} \to 2^{\mathcal{AP}}$ assigns to each state $s \in \mathcal{S}$ a set of atomic propositions $L(s) \subseteq 2^{\mathcal{AP}}$.*

Definition 2 (Path). *An infinite path ρ starting at s_0 is a sequence of states $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ...$ such that every transition $s_i \xrightarrow{a_i} s_{i+1}$ is allowed in $\mathfrak{M}$, i.e. s_{i+1} belongs to the smallest Borel set B such that $P(B|s_i, a_i) = 1$.*

At each state $s \in \mathcal{S}$, an agent behaviour is determined by a Markov policy π, which is a mapping from states to a probability distribution over the actions, i.e. $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. If $\mathcal{P}(\mathcal{A})$ is a degenerate distribution then the policy π is said to be deterministic.

Definition 3 (Expected Discounted Return [49]). *For a policy π on an MDP $\mathfrak{M}$, the expected discounted return is defined as:*

$$U^{\pi}(s) = \mathbb{E}^{\pi}[\sum_{n=0}^{\infty} \gamma^n \ R(s_n, a_n)|s_0 = s],$$

where $\mathbb{E}^{\pi}[\cdot]$ denotes the expected value given that the agent follows policy π, $\gamma \in [0,1)$ ($\gamma \in [0,1]$ when episodic) is a discount factor, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the *reward, and* $s_0, a_0, s_1, a_1, \ldots$ *is the sequence of state-action pairs generated by policy* π*.*

It has been shown that constant discount factors might yield sub-optimal policies [7, 17]. In general the discount factor γ is a hyper-parameter that has to be tuned. There is standard work in RL on state-dependent discount factors [37, 40, 54, 56], that is shown to preserve convergence and optimality guarantees. A possible tuning strategy to resolve the issues of constant discounting is as follows:

$$\gamma(s) = \begin{cases} \eta & \text{if } R(s,a) > 0, \\ 1 & \text{otherwise,} \end{cases} \tag{1}$$

where $0 < \eta < 1$ is a constant. Hence, Definition 3 is reduced to [37]:

$$U^\pi(s) = \mathbb{E}^\pi[\sum_{n=0}^{\infty} \gamma(s_n)^{N(s_n)} \ R(s_n, \pi(s_n))|s_0 = s], \ 0 \leq \gamma \leq 1, \tag{2}$$

where $N(s_n)$ is the number of times a positive reward has been observed at s_n.

The function $U^\pi(s)$ is often referred to as value function (under the policy π). Another crucial notion in RL is action-value function $Q^\pi(s,a)$, which describes the expected discounted return after taking an action a in state s and thereafter following policy π:

$$Q^\pi(s,a) = \mathbb{E}^\pi[\sum_{n=1}^{\infty} \gamma^n \ R(s_n, a_n)|s_0 = s, a_0 = a].$$

Accordingly, the recursive form of the action-value function can be obtained as:

$$Q^\pi(s,a) = R(s,a) + \gamma Q^\pi(s_1, a_1), \tag{3}$$

where $a_1 = \pi(s_1)$. Q-learning (QL) [53] is the most extensively used model-free RL algorithm built upon (3), for MDPs with finite-state and finite-action spaces. For all state-action pairs QL initializes a Q-function $Q^\beta(s,a)$ with an arbitrary finite value, where β is an arbitrary stochastic policy.

$$Q^\beta(s,a) \leftarrow Q^\beta(s,a) + \mu[R(s,a) + \gamma \max_{a' \in \mathcal{A}}(Q^\beta(s',a')) - Q^\beta(s,a)]. \tag{4}$$

where $Q^\beta(s,a)$ is the Q-value corresponding to state-action (s,a), $0 < \mu \leq 1$ is called learning rate (or step size), $R(s,a)$ is the reward function, γ is the discount factor, and s' is the state reached after performing action a. The Q-function for the remaining of the state-action pairs is not changed in this operation. QL is an off-policy RL scheme, namely policy β has no effect on the convergence of the Q-function, as long as every state-action pair is visited infinitely many times. Thus, for the sake of simplicity, we may drop the policy index β from the action-value function. Under mild assumptions, QL converges to a unique limit, and a greedy policy π^* can be obtained as follows:

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \ Q(s,a), \tag{5}$$

and π^* corresponds to the optimal policy that is generated by dynamic programming [6] to maximise the expected return, if the MDP was fully known:

$$\pi^*(s) = \arg\sup_{\pi \in \mathcal{D}} \ U^{\pi}(s), \tag{6}$$

where $\mathcal{D}$ is the set of stationary deterministic policies over the state space $\mathcal{S}$. The Deterministic Policy Gradient (DPG) algorithm [47] introduces a parameterised function $\mu(s|\theta^{\mu})$ called actor to represent the current policy by deterministically mapping states to actions, where θ^{μ} is the function approximation parameters for the actor function. Further, an action-value function $Q(s,a|\theta^{Q})$ is called critic and is learned as described next.

Assume that at time step t the agent is at state s_t, takes action a_t, and receives a scalar reward $R(s_t, a_t)$. In case when the agent policy is deterministic, the action-value function update can be approximated by parameterising Q using a parameter set θ^{Q}, i.e. $Q(s_t, a_t|\theta^{Q})$, and by minimizing the following loss function:

$$L(\theta^{Q}) = \mathbb{E}^{\pi}_{s_t \sim \rho^{\beta}}[(Q(s_t, a_t|\theta^{Q}) - y_t)^2], \tag{7}$$

where ρ^{β} is the probability distribution of state visits over $\mathcal{S}$, under any given arbitrary stochastic policy β, and $y_t = R(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}|\theta^{Q})$ such that $a_{t+1} = \pi(s_{t+1})$.

The actor parameters are updated along the derivative of the expected return, which [47] has shown to be a policy gradient, as follows:

$$\begin{aligned} \nabla_{\theta^{\mu}} U^{\mu}(s_t) &\approx \mathbb{E}_{s_t \sim p^{\beta}}[\nabla_{\theta^{\mu}} Q(s,a|\theta^{Q})|_{s=s_t, a=\mu(s_t|\theta^{\mu})}] \\ &= \mathbb{E}_{s_t \sim p^{\beta}}[\nabla_{a} Q(s,a|\theta^{Q})|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s=s_t}]. \end{aligned} \tag{8}$$

DDPG further extends DPG by employing a deep neural network as function approximator and updating the network parameters via a "soft update" method, which is explained later in the paper.

2.1 Linear Temporal Logic (LTL)

We employ LTL to encode the structure of the high-level mission task and to automatically shape the reward function. LTL formulae φ over a given set of atomic propositions $\mathcal{AP}$ are syntactically defined as [41]:

$$\varphi ::= true \mid \alpha \in \mathcal{AP} \mid \varphi \land \varphi \mid \neg \varphi \mid \bigcirc \varphi \mid \varphi \ \mathrm{U} \ \varphi, \tag{9}$$

where the operators $\bigcirc$ and U are called "next" and "until", respectively. For a given path ρ, we define the i-th state of ρ to be $\rho[i]$ where $\rho[i] = s_i$, and the i-th suffix of ρ to be $\rho[i..]$ where $\rho[i..] = s_i \xrightarrow{a_i} s_{i+1} \xrightarrow{a_{i+1}} s_{i+2} \ldots$

Definition 4 (LTL Semantics [41]). *For an LTL formula φ and for a path ρ, the satisfaction relation $\rho \models \varphi$ is defined as*

$$\begin{aligned}
&\rho \models \alpha \in \mathcal{AP} \iff \alpha \in L(\rho[0]),\\
&\rho \models \varphi_1 \wedge \varphi_2 \iff \rho \models \varphi_1 \wedge \rho \models \varphi_2,\\
&\rho \models \neg\varphi \iff \rho \not\models \varphi,\\
&\rho \models \bigcirc\varphi \iff \rho[1..] \models \varphi,\\
&\rho \models \varphi_1 \mathrm{U} \varphi_2 \iff \exists j \geq 0 : \rho[j..] \models \varphi_2 \wedge \forall i, 0 \leq i < j, \rho[i..] \models \varphi_1.
\end{aligned}$$

The operator next $\bigcirc$ requires φ to be satisfied starting from the next-state suffix of ρ. The operator until U is satisfied over ρ if φ_1 continuously holds until φ_2 becomes true. Using the until operator U we can define two temporal modalities: (1) eventually, $\Diamond\varphi = \mathit{true}\ \mathrm{U}\ \varphi$; and (2) always, $\Box\varphi = \neg\Diamond\neg\varphi$. LTL extends propositional logic using the temporal modalities until U, eventually $\Diamond$, and always $\Box$. For instance, constraints such as "eventually reach this point", "visit these points in a particular sequential order", or "always stay safe" are expressible by these modalities. Further, these modalities can be combined with logical connectives and nesting to provide more complex task specifications. Any LTL task specification φ over $\mathcal{AP}$ expresses the following set of words $\mathit{Words}(\varphi) = \{\sigma \in (2^{\mathcal{AP}})^\omega \text{ s.t. } \sigma \models \varphi\}$, where $(2^{\mathcal{AP}})^\omega$ is set of all infinite words over $2^{\mathcal{AP}}$. The set of associated words $\mathit{Words}(\varphi)$ is expressible using a finite-state machine [3]. Limit Deterministic Büchi Automata (LDBA) [45] are shown to be succinct finite-state machines for this purpose [46]. We first define a Generalized Büchi Automaton (GBA), then we formally introduce LDBA.

Definition 5 (Generalized Büchi Automaton). *A GBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ is a state machine, where $\mathcal{Q}$ is a finite set of states, $q_0 \subseteq \mathcal{Q}$ is the set of initial states, $\Sigma = 2^{\mathcal{AP}}$ is a finite alphabet, $\mathcal{F} = \{F_1, ..., F_f\}$ is the set of accepting conditions where $F_j \subseteq \mathcal{Q}, 1 \leq j \leq f$, and $\Delta : \mathcal{Q} \times \Sigma \to 2^{\mathcal{Q}}$ is a transition relation.*

Let Σ^ω be the set of all infinite words over Σ. An infinite word $w \in \Sigma^\omega$ is accepted by a GBA $\mathfrak{A}$ if there exists an infinite run $\theta \in \mathcal{Q}^\omega$ starting from q_0 where $\theta[i+1] \in \Delta(\theta[i], \omega[i]),\ i \geq 0$ and, for each $F_j \in \mathcal{F}$, $\mathit{inf}(\theta) \cap F_j \neq \emptyset$, where $\mathit{inf}(\theta)$ is the set of states that are visited infinitely often in the sequence θ.

Definition 6 (LDBA [45]). *A GBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ is limit-deterministic if $\mathcal{Q}$ is composed of two disjoint sets $\mathcal{Q} = \mathcal{Q}_N \cup \mathcal{Q}_D$, such that:*

- *$\Delta(q, \alpha) \subset \mathcal{Q}_D$ and $|\Delta(q, \alpha)| = 1$ for every state $q \in \mathcal{Q}_D$ and for every $\alpha \in \Sigma$,*
- *for every $F_j \in \mathcal{F}$, $F_j \subseteq \mathcal{Q}_D$,*
- *$q_0 \in \mathcal{Q}_N$, and all the transitions from $\mathcal{Q}_N$ to $\mathcal{Q}_D$ are non-deterministic ε-transitions. An ε-transition allows an automaton to change its state without reading any atomic proposition.*

Intuitively, the defined LDBA is a GBA that has two components, an initial ($\mathcal{Q}_N$) and an accepting one ($\mathcal{Q}_D$). The accepting component includes all the accepting states and has deterministic transitions. As it will be further elaborated below, ε-transitions between $\mathcal{Q}_N$ and $\mathcal{Q}_D$ can be interpreted as "guesses" on reaching $\mathcal{Q}_D$. We finally introduce the following notion.

Definition 7 (Non-accepting Sink Component). *A non-accepting sink component of the LDBA $\mathfrak{A}$ is a directed graph induced by a set of states $Q \subseteq \mathcal{Q}$ such that (1) the graph is strongly connected; (2) it does not include all of the accepting sets $F_k,\ k = 1, ..., f$ that are necessary to satisfy the associated LTL formula; and (3) there exist no other strongly connected set $Q' \subseteq \mathcal{Q},\ Q' \neq Q$ such that $Q \subseteq Q'$. We denote the union of all non-accepting sink components of $\mathfrak{A}$ as $\mathbb{N}$. The set $\mathbb{N}$ includes those components in the automaton that are non-accepting and impossible to escape from. Thus, a trace reaching them is doomed to not satisfy the given LTL property.*

3 Modular DDPG

We consider an RL problem in which we exploit the structural information provided by the LTL specification, by constructing sub-policies for each state of the associated LDBA. The proposed approach learns a satisfying policy without requiring any information about the grounding of the LTL task to be specified explicitly. Namely, the labelling assignment (as in Definition 1) is a-priori unknown, and the algorithm solely relies on experience samples gathered online.

Given an LTL mission task and an unknown black-box continuous-state/action MDP, we aim to synthesise a policy that satisfies the LTL specification with max probability. For the sake of clarity and to explain the core ideas of the algorithm, for now we assume that the MDP graph and the transition kernel are known: later these assumptions are entirely removed, and we stress that the algorithm can be run model-free. We relate the MDP and the automaton by synchronising them, in order to create a new structure that is firstly compatible with deep RL and that secondly encompasses the given logical property.

Definition 8 (Product MDP). *Given an MDP $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ and an LDBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ with $\Sigma = 2^{\mathcal{AP}}$, the product MDP is defined as $\mathfrak{M}_{\mathfrak{A}} = \mathfrak{M} \otimes \mathfrak{A} = (\mathcal{S}^{\otimes}, \mathcal{A}, s_0^{\otimes}, P^{\otimes}, \mathcal{AP}^{\otimes}, L^{\otimes}, \mathcal{F}^{\otimes})$, where $\mathcal{S}^{\otimes} = \mathcal{S} \times \mathcal{Q}$, $s_0^{\otimes} = (s_0, q_0)$, $\mathcal{AP}^{\otimes} = \mathcal{Q}$, $L^{\otimes} : \mathcal{S}^{\otimes} \rightarrow 2^{\mathcal{Q}}$ such that $L^{\otimes}(s, q) = q$ and $\mathcal{F}^{\otimes} \subseteq \mathcal{S}^{\otimes}$ is the set of accepting states $\mathcal{F}^{\otimes} = \{F_1^{\otimes}, ..., F_f^{\otimes}\}$, where $F_j^{\otimes} = \mathcal{S} \times F_j$. The transition kernel $P^{\otimes}$ is such that given the current state (s_i, q_i) and action a, the new state (s_j, q_j) is obtained such that $s_j \sim P(\cdot|s_i, a)$ and $q_j \in \Delta(q_i, L(s_j))$.*
In order to handle ε-transitions we make the following modifications to the above definition of product MDP:

- *for every potential ε-transition to some state $q \in \mathcal{Q}$ we add a corresponding action ε_q in the product:*

$$\mathcal{A}^{\otimes} = \mathcal{A} \cup \{\varepsilon_q, q \in \mathcal{Q}\}.$$

- *The transition probabilities corresponding to ε-transitions are given by*

$$P^{\otimes}((s_i, q_i), \varepsilon_q, (s_j, q_j)) = \begin{cases} 1 & \text{if } s_i = s_j,\ q_i \xrightarrow{\varepsilon_q} q_j = q, \\ 0 & \text{otherwise.} \end{cases}$$

Remark 1. Recall that an ε-transition between $\mathcal{Q}_N$ and $\mathcal{Q}_D$ corresponds to a guess on reaching $\mathcal{Q}_D$ without reading a label and changing a state in the MDP $\mathfrak{M}$ (see the definition of $P^{\otimes}$ above). This entails that if, after an ε-transition, the associated label in the accepting set of the automaton cannot be read or no accepting state in $\mathcal{Q}_D$ is visited, then the guess was wrong, hence the automaton must have entered the non-accepting sink component $\mathbb{N}$ (Def. 7). These semantics are leveraged in the case studies, and are generally applicable when the constructed LDBA contains ε-transitions. □

By constructing the product MDP, we synchronise the current state of the agent with that of the automaton. This allows to evaluate partial satisfaction of the corresponding LTL property, and consequently to modularise the high-level LTL task into sub-tasks. Hence, with a proper reward assignment based on the LTL property and its associated LDBA, the agent can break down a complex task into a set of easier sub-tasks. We elaborate further on task modularisation in the next subsection.

In the following we define an LTL-based reward function, emphasising that the agent does not need to know the model structure or the transition probabilities (or their product). Before introducing a reward assignment for the RL agent, we need to present the ensuing function:

Definition 9 (Accepting Frontier Function). *For an LDBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$, we define $Acc : \mathcal{Q} \times 2^{\mathcal{Q}} \rightarrow 2^{\mathcal{Q}}$ as the accepting frontier function, which executes the following operation over a given set $\mathbb{F} \subset 2^{\mathcal{Q}}$ for every $F_j \in \mathcal{F}$:*

$$Acc(q, \mathbb{F}) = \begin{cases} \mathbb{F}_{\ \setminus \{F_j\}} & (q \in F_j) \wedge (\mathbb{F} \neq F_j), \\ \{F_k\}_{k=1\ \setminus \{F_j\}}^{f} & (q \in F_j) \wedge (\mathbb{F} = F_j), \\ \mathbb{F} & \textit{otherwise.} \end{cases}$$

In words, once the state $q \in F_j$ and the set $\mathbb{F}$ are introduced to the function Acc, it outputs a set containing the elements of $\mathbb{F}$ minus F_j. However, if $\mathbb{F} = F_j$, then the output is the family set of all accepting sets of the LDBA, minus the set F_j. Finally, if the state q is not an accepting state then the output of Acc is $\mathbb{F}$. The accepting frontier function excludes from $\mathbb{F}$ the accepting set that is currently visited, unless it is the only remaining accepting set. Otherwise, the output of $Acc(q, \mathbb{F})$ is $\mathbb{F}$ itself. Owing to the automaton-based structure of the Acc function, we are able to shape a reward function (as detailed next) without any supervision and regardless of the dynamics of the MDP.

We propose a reward function that observes the current state $s^{\otimes}$, the current action a, and the subsequent state ${s^{\otimes}}'$, to provide the agent with a scalar value according to the current automaton state:

$$R(s^{\otimes}, a) = \begin{cases} r_p \ if \ q' \in \mathbb{A}, \ {s^{\otimes}}' = (s', q'), \\ r_n \ if \ q' \in \mathbb{N}, \ {s^{\otimes}}' = (s', q'), \\ 0, \qquad\qquad \text{otherwise.} \end{cases} \tag{10}$$

Here r_p is a positive reward and r_n is a negative reward. A positive reward r_p is assigned to the agent when it takes an action that leads to a state, the label of which is in $\mathbb{A}$, and a negative reward r_n is given upon reaching $\mathbb{N}$ (Definition 7). The set $\mathbb{A}$ is called the accepting frontier set, is initialised as the family set $\mathbb{A} = \{F_k\}_{k=1}^{f}$, and is updated by the following rule every time after the reward function is evaluated: $\mathbb{A} \leftarrow \ Acc(q', \mathbb{A})$. The set $\mathbb{N}$ is the set of non-accepting sink components of the automaton, as per Definition 7.

Remark 2. The intuition underlying (10) is that set $\mathbb{A}$ contains those accepting states that are visited at a given time. Thus, the agent is guided by the above reward assignment to visit these states and once all of the sets $F_k,\ k = 1, ..., f$, are visited, the accepting frontier $\mathbb{A}$ is reset. As such, the agent is guided to visit the accepting sets infinitely often, and consequently, to satisfy the given LTL property. We shall discuss issues of reward sparseness in Section 3.1. □

Given the product MDP structure in Definition 8 and the automatic formal reward assignment in (10), any algorithm that synthesises a policy maximising the associated expected discounted return over the product MDP, maximises the probability of satisfying the property φ. Note that, unlike the case of finite MDPs [18], proving the aforementioned claim is not trivial as we cannot leverage notions that are specific to finite MDPs, such as that of Accepting Max End Component (AMEC). Thus, the probability of satisfaction cannot be equated to the probability of reaching a set of states in the product MDP (i.e., the AMEC) and we have to directly reason over the accepting condition of the LDBA.

Theorem 1. *Let φ be a given LTL formula and $\mathfrak{M}_\mathfrak{A}$ be the product MDP constructed by synchronising the MDP $\mathfrak{M}$ with the LDBA $\mathfrak{A}$ expressing φ. Then the optimal stationary Markov policy on $\mathfrak{M}_\mathfrak{A}$ that maximises the expected return, maximises the probability of satisfying φ and induces a finite-memory policy on the MDP $\mathfrak{M}$.*

Remark 3. Please see the Appendix for the proof. Note that the optimality of the policy generated by the DDPG scheme depends on a number of factors, such as its structure, number of hidden layers, and activation functions. The quantification of the sub-optimality of the policy generated by DDPG is out of the scope of this work. □

3.1 Task Modularisation

In this section we explain how a complex task can be broken down into simple composable sub-tasks or modules. Each state of the automaton in the product MDP is a "task divider" and each transition between these states is a "sub-task". For example consider a sequential task of visit a and then b and finally c, i.e. $\lozenge(a \wedge \lozenge(b \wedge \lozenge c))$. The corresponding automaton for this LTL task is given in Fig. 1. The entire task is modularised into three sub-tasks, i.e. reaching a, b, and then c, and each automaton state acts as a divider. By synchronising the MDP $\mathfrak{M}$ and the LDBA $\mathfrak{A}$, each automaton state divides those parts of the state space $\mathcal{S}$, whose boundaries are sub-tasks, namely automaton transitions. Furthermore, the LDBA specifies the relations between subs-tasks, e.g. ordering and repetition.

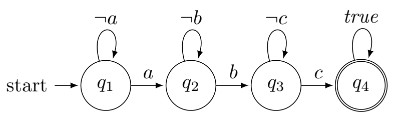


Fig. 1: LDBA for a sequential task expressed by $\lozenge(a \wedge \lozenge(b \wedge \lozenge c))$.

By exploiting the relationship between sub-tasks, and also limiting the agent exploration to the relevant regions of the state space for each sub-task, it can efficiently guide the learning to solve problems with sparse rewards.

Given an LTL task and its LDBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$, we propose a modular architecture of $n = |\mathcal{Q}|$ separate actor and critic neural networks, along with their own replay buffer. A replay buffer is a finite-sized cache in which transitions sampled from exploring the environment are stored. The replay buffer is then used to train the actor and critic networks. The set of neural nets acts as a global modular actor-critic deep RL architecture, which allows the agent to jump from one sub-task to another by just switching between the set of neural nets. For each automaton state q_i an actor function $\mu_{q_i}(s|\theta^{\mu_{q_i}})$ represents the current policy by deterministically mapping states to actions, where $\theta^{\mu_{q_i}}$ is the vector of parameters of the function approximation for the actor. The critic $Q_{q_i}(s, a|\theta^{Q_{q_i}})$ is learned based on (7).

The modular DDPG algorithm is detailed in Algorithm 1. Each actor-critic network set in this algorithm is associated with its own replay buffer $\mathcal{R}_{q_i}$, where $q_i \in \mathcal{Q}$ (line 4, 12). Experience samples are stored in $\mathcal{R}_{q_i}$ in the form of

$$(s_i^{\otimes}, a_i, R_i, s_{i+1}^{\otimes}) = ((s_i, q_i), a_i, R_i, (s_{i+1}, q_{i+1})).$$

When the replay buffer reaches its maximum capacity, the samples are discarded based on a first-in/first-out policy. At each time-step, actor and critic are updated by sampling a mini-batch of size $\mathcal{M}$ uniformly from $\mathcal{R}_{q_i}$. Therefore, in the algorithm the actor-critic network set corresponding to the current automaton state q_t, is trained based on experience samples in $\mathcal{R}_{q_t}$ (line 12-17).

Further, directly implementing the update of the critic parameters as in (7) is shown to be potentially unstable, and as a result the Q-update (line 14) is prone to divergence [36]. Hence, instead of directly updating the networks weights, the standard DDPG [34] introduces two “target networks”, μ' and Q', which are time-delayed copies of the original actor and critic networks μ and Q, respectively. DDPG uses “soft” target updates to improve learning stability, which means that μ' and Q' slowly track the learned networks, μ and Q. These target actor and critic networks are used within the algorithm to gather evidence (line 13) and subsequently to update the actor and critic networks. In our algorithm, for each automaton state q_i we make a copy of the actor and the critic network: $\mu'_{q_i}(s|\theta^{\mu'_{q_i}})$ and $Q'_{q_i}(s, a|\theta^{Q'_{q_i}})$ respectively. The weights of both target networks are then updated as $\theta' = \tau\theta + (1 - \tau)\theta'$ with a rate of $\tau < 1$ (line 18). Summarising, to increase stability and robustness in learning, for each automaton state q_i we

Algorithm 1: Modular DDPG

```
   input  : LTL task $\varphi$, a black-box agent/environment
   output : trained actor and critic networks
1  convert the LTL property $\varphi$ to LDBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$
2  randomly initialise $|\mathcal{Q}|$ actors $\mu_i(s|\theta^{\mu_i})$ and critic $Q_i(s, a|\theta^{Q_i})$ networks with
     weights $\theta^{\mu_i}$ and $\theta^{Q_i}$, for each $q_i \in \mathcal{Q}$, and all state-action pairs $(s, a)$
3  initialize $|\mathcal{Q}|$ target networks $\mu'_i$ and $Q'_i$ with weights $\theta^{\mu'_i} = \theta^{\mu_i}$, $\theta^{Q'_i} = \theta^{Q_i}$
4  initialise $|\mathcal{Q}|$ replay buffers $\mathcal{R}_i$
5  repeat
6      initialise $|\mathcal{Q}|$ random processes $\mathfrak{N}_i$
7      initialise state $s_1^{\otimes} = (s_0, q_0)$
8      for $t = 1$ to $max\_iteration\_number$ do
9          choose action $a_t = \mu_{q_t}(s_t|\theta^{\mu_{q_t}}) + \mathfrak{N}_{q_t}$
10         observe reward $r_t$ and the new state $(s_{t+1}, q_{t+1})$
11         store $((s_t, q_t), a_t, R_t, (s_{t+1}, q_{t+1}))$ in $\mathcal{R}_{q_t}$
12         sample a random mini-batch of $\mathcal{M}$ transitions
             $((s_i, q_i), a_i, R_i, (s_{i+1}, q_{i+1}))$ from $\mathcal{R}_{q_t}$
13         set $y_i = R_i + \gamma Q'_{q_{i+1}}(s_{i+1}, \mu'_{q_{i+1}}(s_{i+1}|\theta^{\mu'_{q_{i+1}}})|\theta^{Q'_{q_{i+1}}})$
14         update critic $Q_{q_t}$ and $\theta^{Q_{q_t}}$ by minimizing the loss:
             $L = 1/|\mathcal{M}| \sum_i (y_i - Q_{q_t}(s_i, a_i|\theta^{Q_{q_t}}))^2$
15         update the actor policy $\mu_{q_t}$ and $\theta^{\mu_{q_t}}$ by maximizing the sampled policy
             gradient:
16         $\nabla_{\theta^{\mu_{q_t}}} U^{\mu_{q_t}} \approx 1/|\mathcal{M}| \sum_i [\nabla_a Q_{q_t}(s, a|\theta^{Q_{q_t}})|_{s=s_i, a=\mu_{q_t}(s_i|\theta^{\mu_{q_t}})}$
17             $\nabla_{\theta^{\mu_{q_t}}} \mu_{q_t}(s|\theta^{\mu_{q_t}})|_{s=s_i}]$
18         update the target networks: $\theta^{Q'_{q_t}} \leftarrow \tau\theta^{Q_{q_t}} + (1-\tau)\theta^{Q'_{q_t}}$
             $\theta^{\mu'_{q_t}} \leftarrow \tau\theta^{\mu^{q_t}} + (1-\tau)\theta^{\mu'_{q_t}}$
19     end
20 until end of trial
```

have a pair of actor and critic networks, namely $\mu_{q_i}(s|\theta^{\mu_{q_i}})$, $\mu'_{q_i}(s|\theta^{\mu'_{q_i}})$ and $Q'_{q_i}(s, a|\theta^{Q'_{q_i}})$, $Q_{q_i}(s, a|\theta^{Q_{q_i}})$ respectively.

4 Experiments

In this section we showcase the simulation results of Modular DDPG in two case studies: a cart-pole setup and in a mission planning problem for a Mars rover.

In the cart-pole example (Fig. 2) a pendulum is attached to a cart, which moves horizontally along a friction-less track [50]. The agent applies a horizontal force to the cart. The pendulum starts upright, and the goal is (1) to prevent the pendulum from falling over, and (2) to move the cart between the yellow and green regions while avoiding the red (unsafe) parts of the track.

The second case study deals with a Mars rover, exploring areas around the Melas Chasma [35] and the Victoria crater [48]. The Melas Chasma area displays

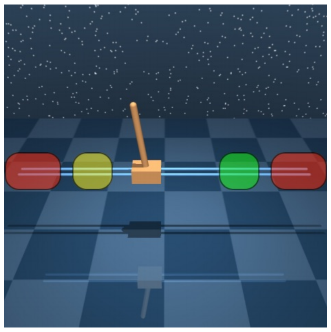


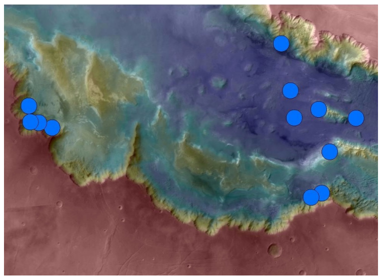
Fig. 2: Cart-pole case study (a.k.a. inverted pendulum on a cart) [50].

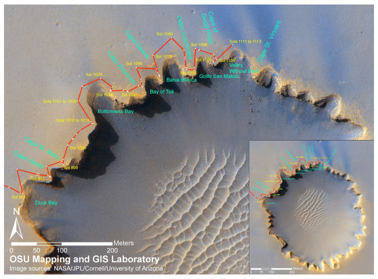
a number of signs of potential presence of water, with possible river valleys and lakes (Fig. 3. a). The blue dots, provided by NASA, indicate Recurring Slope Lineae (RSL) on the surface of Mars [35], which are possible locations of liquid water. The agent task is to first visit low-risk RSL (on the right of the area) and then to visit high-risk RSL (on the left), while avoiding unsafe red regions. The Victoria crater (Fig. 3. b) is an impact crater located near the equator of Mars. Layered sedimentary rocks are exposed along the wall of the crater, providing information about the ancient surface condition of Mars [48]. A NASA Mars rover has been operating around the crater and its mission path is given in Fig. 3. b. The scenario of interest in this work is to train an RL agent that can autonomously control the rover to accomplish the safety-critical complex task.

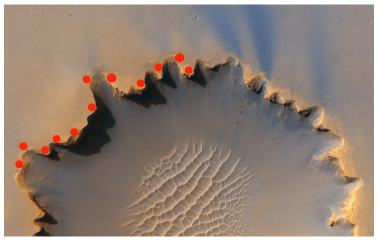
In each experiment, we convert tasks expressed as LTL properties into corresponding LDBAs, and use them to monitor the modular DDPG algorithm, thus implicitly forming a product automaton. For each actor/critic structure, we have used a feed-forward neural net with 2 fully connected hidden layers and 400 ReLu units in each layer.

MDP structure In the cart-pole experiment the pendulum starts upright with an initial angle between -0.05 and 0.05 rads. The mass of the cart is 1 kg and that of the pole is 100 g. The length of the pole is 1 m. The force applied by the agent to the cart ranges from -10 to 10 N. A learning episode terminates if the pendulum deviates more than 0.21 rad from the vertical position, or if the cart enters any of the red regions at any time. The yellow region ranges from -2.15 to -1.85 m, and symmetrically the green region is from 1.85 to 2.15 m. The unsafe red region lies at the left of -4 m and at the right of 4 m.

In the Mars rover experiments, let the area of each image be the state space $\mathcal{S}$ of the MDP, where the rover location is a state $s \in \mathcal{S}$. At each state s the rover has a continuous range of actions $\mathcal{A} = [0, 2\pi)$: when the rover takes an action it moves to another state (e.g., s') towards the direction of the action and within a range that is randomly drawn within the interval $(0, D]$, unless the rover hits the boundary of the image, which restarts the learning episode.


(a)


N
Meters
0 50 100 200
OSU Mapping and GIS Laboratory
Image sources: NASA/JPL/Cornell/University of Arizona
(b)


(c)

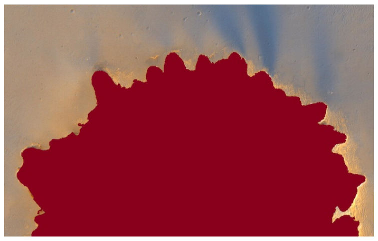
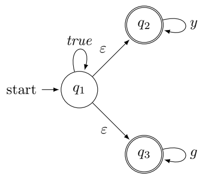
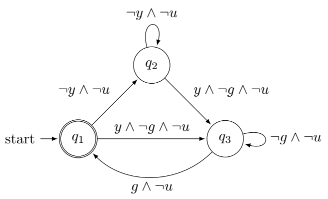

(d)

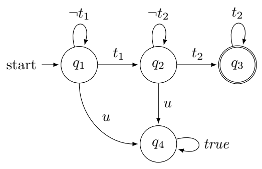
Fig. 3: (a) Melas Chasma in the Coprates quadrangle, map color spectrum represents elevation, where red is high (unsafe) and blue is low. (b) Victoria crater and Opportunity rover mission traverse map [48], (c) replicated points of interest, and (d) unsafe area (red). Image courtesy of NASA, JPL, Cornell, and Arizona University.



(a)



(b)



(c)

Fig. 4: LDBAs expressing formula (11) in (a), (12) in (b), and (13) in (c).

In the actual Melas Chasma exploration mission (Fig. 3. a), the rover is deployed on a landing location that is not precisely known. We therefore encompass randomness over the initial state s_0. Conversely, in the second experiment (Fig. 3. b) the rover is supposed to have already landed and it starts its mission from a known state.

The dimension of the area of interest in Fig. 3. a is 456.98×322.58 km, whereas in Fig. 3. b is 746.98×530.12 m. Other parameters in this numerical

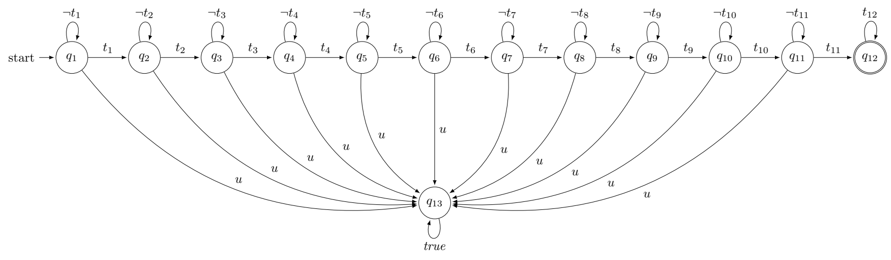

start
q_1 q_2 q_3 q_4 q_5 q_6 q_7 q_8 q_9 q_{10} q_{11} q_{12} q_{13}
$\neg t_1$ $\neg t_2$ $\neg t_3$ $\neg t_4$ $\neg t_5$ $\neg t_6$ $\neg t_7$ $\neg t_8$ $\neg t_9$ $\neg t_{10}$ $\neg t_{11}$ t_{12}
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11}
u
true

Fig. 5: LDBA expressing the LTL formula in (14).

example have been set as $D = 2\ km$ for Melas Chasma, $D = 10\ m$ for the Victoria crater. We have used the satellite image with additive noise as the black-box MDP for the experiment. Note that the rover only needs its current state (coordinates in this case), the state label (without seeing the entire map), and the LTL task (Algorithm 1, line 11). The given maps in the paper are for illustration purposes and to elucidate the performance of the output policy.

Specifications In the cart-pole setup the properties of interest are expressed by the following two LTL formulae:

$$\Diamond\Box y \vee \Diamond\Box g, \tag{11}$$

$$\Box\Diamond y \wedge \Box\Diamond g \wedge \Box\neg u, \tag{12}$$

where y is the label of the yellow region, g denotes the green region, and u is the label denoting when either the pendulum falls or when the cart enters the red regions on the track. We call the experiment with (11) *Cart-pole-1* and that with (12) *Cart-pole-2*. Note that the task in Cart-pole-2 is a surveillance finite-memory specification: such tasks can be easily expressed in LTL and achieved by the modular DDPG architecture, but are impossible to solve with conventional RL.

In the first of the Mars rover case studies, over the Melas Chasma area (Fig. 3. a) the control objective is expressed by the following LTL formula:

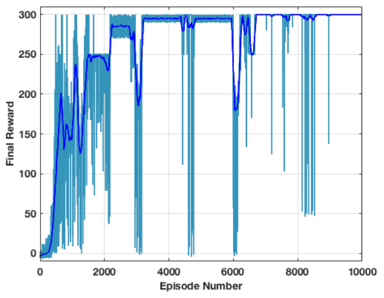
$$\Diamond(t_1 \wedge \Diamond t_2) \wedge \Box(\neg u), \tag{13}$$

where t_1 stands for “target 1”, t_2 stands for “target 2” and u stands for “unsafe” (the red region in Fig. 3. d). Target 1 corresponds to the RSL (blue dots) on the right with a lower risk of the rover going to unsafe region, whereas the “target 2” label goes on the left RSL that are a bit riskier to explore. Conforming to (13), the rover has to visit any of the right dots at least once and then proceed to the any of the left dots, while avoiding unsafe areas. From (13) we build the associated LDBA as in Fig. 4.

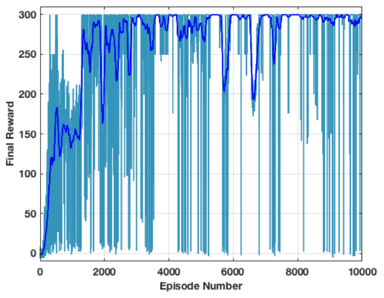
The mission task for the Victoria crater is taken from a NASA rover mission [48] and is expressed by the following LTL formula:

$$\Diamond(t_1 \wedge \Diamond(t_2 \wedge \Diamond(t_3 \wedge \Diamond(t_4 \wedge \Diamond(... \wedge \Diamond(t_{12})))) \wedge \Box(\neg u), \tag{14}$$

where t_i represents the "i-th target", and u represents "unsafe" regions. The i-th target in Fig. 3. c is the i-th red circle from the bottom left along the crater rim. According to (14) the rover is required to visit the checkpoints from the bottom left to the top right sequentially, while avoiding a fall into the crater, which mimicks the actual path in Fig. 3. b. From (14), we build the associated LDBA as shown in Fig 5.

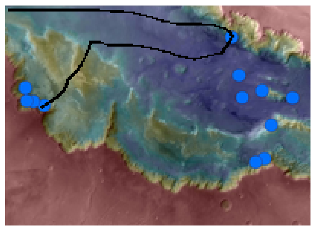


(a)

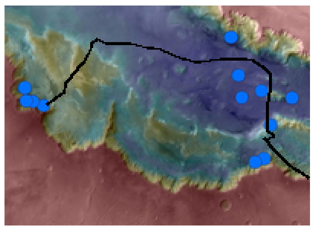


(b)

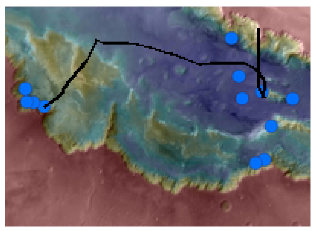
Fig. 6: Learning curves (dark blue) obtained averaging over 10 randomly initialised experiments in the cart-pole setup with the task specified in (11) for (a) and in (12) for (b). Shaded areas (light blue) represent the envelopes of the 10 generated learning curves.



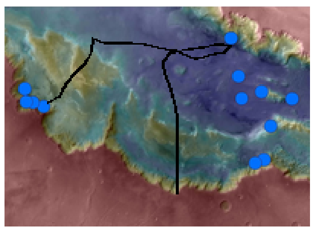
(a) Coordinates (2,2)



(b) Coordinates (113,199)



(c) Coordinates (14,165)



(d) Coordinates (122,113)

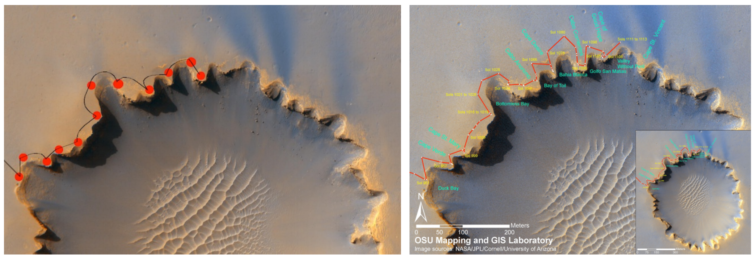
Fig. 7: Paths generated by the learnt policy in the Melas Chasma experiment.

Experimental Outcomes In each cart-pole experiment we have employed three actor-critic neural network pairs and ran simulations for 10,000 episodes. We have then tested the trained network across 200 runs. Modular DDPG has achieved a success rate of 100% (Table 1) and Fig. 6 shows the learning progress. The learning run time has been 57 minutes.

In the Melas Chasma experiment we have employed 4 actor-critic neural network pairs and ran simulations for 10,000 episodes. We have then tested the trained network for all safe starting position. Our algorithm has achieved a success rate of 98.8% across 18,202 landing positions, as reported Table 1. Fig. 7 gives the example paths generated by our algorithm. The learning run time has been 5 hours.

In the Victoria crater experiment we have used 13 actor-critic neural network pairs. We have ran simulations for a total of 17,000 episodes, at which point it had already converged. The learning run time was 2 days. We have then tested the trained network across 200 runs. Our algorithm has achieved a success rate of 100% across all runs starting from t_1 (Table 1). Fig. 8 shows a generated path that bends away from the crater, due to the back-propagation of the negative reward in (10) associated with violating the safety constraint.

Discussion In all the experiments, the modular DDPG algorithm has been able to automatically modularise the given LTL task and to synthesise a successful policy. We have employed stand-alone DDPG as a baseline for comparison. In the cart-pole example, without synchronising the LDBA with the actor/environment, stand-alone DDPG cannot learn a policy for the non-Markovian task in (12). Hierarchical RL methods, e.g. [32], are able to generate goal-oriented policies, but only when the mission task is sequential and not particularly complex: as such, they would not be useful for (12). Furthermore, in state-of-the-art hierarchical RL there are a number of extra hyper-parameters to tune, such as the sub-goal horizons and the number of hierarchical levels, which conversely the one-shot modular DDPG does not have. The task in (11) is chosen to elucidate how the


N
Meters
0 50 100 200
OSU Mapping and GIS Laboratory
Image sources: NASA/JPL/Cornell/University of Arizona

(a) (b)

Fig. 8: (a) Path generated by the policy learnt via modular DDPG around the Victoria crater, vs (b) the actual Mars rover traverse map [48].

Table 1: Success Rates: statistics are taken over at least 200 trials

Case Study	Algorithm	Success Rate
Cart-pole-1	Stand-alone DDPG	100%
	Modular DDPG	100%
Cart-pole-2	Stand-alone DDPG	0 %
	Modular DDPG	100%
Melas Chasma*	Stand-alone DDPG	21.4%
	Modular DDPG	98.8%
Victoria Crater	Stand-alone DDPG	0%
	Modular DDPG	100%

*The statistics for the M. Chasma study are taken over different initial positions

modular DDPG algorithm can handle cases in which the generated LDBA has non-deterministic ε-transitions.

In the Mars rover examples, the reward function for the stand-alone DDPG is r_p if the agent visits all the checkpoints in proper order, and r_n if the agent enters regions with unsafe labels. However, the performance of stand-alone DDPG has been quite poor, due to inefficient navigation. In relatively simple tasks, e.g. the Melas Chasma experiment, stand-alone DDPG has achieved a positive success rate, however at the cost of very high sample complexity in comparison to Modular DDPG. Specifically, due to its modular structure, the proposed architecture requires fewer samples to achieve the same success rate. Each module encompasses a pair of local actor-critic networks, which are trained towards their own objective and, as discussed before, only samples relevant to the sub-task are fed to the networks. On the other hand, in standard DDPG the whole sample set is fed into a large-scale pair of actor-critic networks, which reduces sample efficiency.

5 Conclusions

We have discussed a deep RL scheme for continuous-state/action decision making problems under LTL specifications. The synchronisation of the automaton expressing the LTL formula with deep RL automatically modularises a global complex task into easier sub-tasks. This setup assists the agent to find an optimal policy with a one-shot learning scheme. The high-level relations between sub-tasks become crucial when dealing with sparse reward problems, as the agent exploration is efficiently guided by the task modules, saving the agent from exhaustively exploring the whole state space, and thus improving sample efficiency.

Acknowledgements The authors would like to thank Lim Zun Yuan for valuable discussions and technical support, and to anonymous reviewers for feedback on previous drafts of this manuscript.

References

1. Abate, A., Katoen, J.P., Lygeros, J., Prandini, M.: Approximate model checking of stochastic hybrid systems. European Journal of Control **16**(6), 624–641 (2010)
2. Andreas, J., Klein, D., Levine, S.: Modular multitask reinforcement learning with policy sketches. In: ICML. pp. 166–175 (2017)
3. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
4. Belzner, L., Wirsing, M.: Synthesizing Safe Policies under Probabilistic Constraints with Reinforcement Learning and Bayesian Model Checking. arXiv preprint arXiv:2005.03898 (2020)
5. Bertsekas, D.P., Shreve, S.: Stochastic optimal control: the discrete-time case. Athena Scientific (2004)
6. Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-dynamic Programming, vol. 1. Athena Scientific (1996)
7. Bozkurt, A.K., Wang, Y., Zavlanos, M.M., Pajic, M.: Control synthesis from linear temporal logic specifications using model-free reinforcement learning. arXiv preprint arXiv:1909.07299 (2019)
8. Brázdil, T., Chatterjee, K., Chmelík, M., Forejt, V., Křetínskỳ, J., Kwiatkowska, M., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. In: ATVA. pp. 98–114 (2014)
9. Daniel, C., Neumann, G., Peters, J.: Hierarchical relative entropy policy search. In: Artificial Intelligence and Statistics. pp. 273–281 (2012)
10. De Giacomo, G., Favorito, M., Iocchi, L., Patrizi, F.: Imitation learning over heterogeneous agents with restraining bolts. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 30, pp. 517–521 (2020)
11. De Giacomo, G., Iocchi, L., Favorito, M., Patrizi, F.: Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In: ICAPS. vol. 29, pp. 128–136 (2019)
12. Fu, J., Topcu, U.: Probably approximately correct MDP learning and control with temporal logic constraints. In: Robotics: Science and Systems (2014)
13. Fulton, N.: Verifiably Safe Autonomy for Cyber-Physical Systems. Ph.D. thesis, Carnegie Mellon University Pittsburgh, PA (2018)
14. Fulton, N., Platzer, A.: Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In: Proceedings of the AAAI Conference on Artificial Intelligence (2018)
15. Fulton, N., Platzer, A.: Verifiably safe off-model reinforcement learning. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 413–430 (2019)
16. Gunter, E.: From natural language to linear temporal logic: Aspects of specifying embedded systems in LTL. In: Monterey Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (2003)
17. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Omega-regular objectives in model-free reinforcement learning. In: TACAS. pp. 395–412 (2019)
18. Hasanbeig, M., Abate, A., Kroening, D.: Logically-constrained reinforcement learning. arXiv preprint arXiv:1801.08099 (2018)
19. Hasanbeig, M., Abate, A., Kroening, D.: Certified Reinforcement Learning with Logic Guidance. arXiv preprint arXiv:1902.00778 (2019)
20. Hasanbeig, M., Abate, A., Kroening, D.: Logically-Constrained Neural Fitted Q-Iteration. In: AAMAS. pp. 2012–2014 (2019)

21. Hasanbeig, M., Abate, A., Kroening, D.: Cautious reinforcement learning with logical constraints. In: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems. pp. 483–491. International Foundation for Autonomous Agents and Multiagent Systems (2020)
22. Hasanbeig, M., Kantaros, Y., Abate, A., Kroening, D., Pappas, G.J., Lee, I.: Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In: Proceedings of the 58th Conference on Decision and Control. pp. 5338–5343. IEEE (2019)
23. Hasanbeig, M., Yogananda Jeppu, N., Abate, A., Melham, T., Kroening, D.: Deepsynth: Program synthesis for automatic task segmentation in deep reinforcement learning. arXiv preprint arXiv:1911.10244 (2019)
24. Huang, C., Xu, S., Wang, Z., Lan, S., Li, W., Zhu, Q.: Opportunistic intermittent control with safety guarantees for autonomous systems. arXiv preprint arXiv:2005.03726 (2020)
25. Hunt, N., Fulton, N., Magliacane, S., Hoang, N., Das, S., Solar-Lezama, A.: Verifiably safe exploration for end-to-end reinforcement learning. arXiv preprint arXiv:2007.01223 (2020)
26. Jansen, N., Könighofer, B., Junges, S., Bloem, R.: Shielded decision-making in MDPs. arXiv preprint arXiv:1807.06096 (2018)
27. Junges, S., Jansen, N., Dehnert, C., Topcu, U., Katoen, J.P.: Safety-constrained reinforcement learning for MDPs. In: TACAS. pp. 130–146 (2016)
28. Kazemi, M., Soudjani, S.: Formal policy synthesis for continuous-space systems via reinforcement learning. arXiv preprint arXiv:2005.01319 (2020)
29. Kearns, M., Singh, S.: Near-optimal reinforcement learning in polynomial time. Machine learning **49**(2-3), 209–232 (2002)
30. Kulkarni, T.D., Narasimhan, K., Saeedi, A., Tenenbaum, J.: Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In: NIPS. pp. 3675–3683 (2016)
31. Lavaei, A., Somenzi, F., Soudjani, S., Trivedi, A., Zamani, M.: Formal controller synthesis for continuous-space MDPs via model-free reinforcement learning. In: 2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS). pp. 98–107. IEEE (2020)
32. Levy, A., Konidaris, G., Platt, R., Saenko, K.: Learning multi-level hierarchies with hindsight. In: International Conference on Learning Representations (ICLR) (2019)
33. Li, X., Ma, Y., Belta, C.: A policy search method for temporal logic specified reinforcement learning tasks. In: ACC. pp. 240–245 (2018)
34. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv:1509.02971 (2015)
35. McEwen, A.S., Dundas, C.M., Mattson, S.S., Toigo, A.D., Ojha, L., Wray, J.J., Chojnacki, M., Byrne, S., Murchie, S.L., Thomas, N.: Recurring slope lineae in equatorial regions of Mars. Nature Geoscience **7**(1), 53 (2014)
36. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
37. Newell, R.G., Pizer, W.A.: Discounting the distant future: how much do uncertain rates increase valuations? Journal of Environmental Economics and Management **46**(1), 52–71 (2003)
38. Nikora, A.P., Balcom, G.: Automated identification of LTL patterns in natural language requirements. In: ISSRE. pp. 185–194 (2009)

39. Oura, R., Sakakibara, A., Ushio, T.: Reinforcement learning of control policy for linear temporal logic specifications using limit-deterministic generalized Büchi automata. IEEE Control Systems Letters **4**(3), 761–766 (2020)
40. Pitis, S.: Rethinking the discount factor in reinforcement learning: A decision theoretic approach. arXiv preprint arXiv:1902.02893 (2019)
41. Pnueli, A.: The temporal logic of programs. In: Foundations of Computer Science. pp. 46–57 (1977)
42. Polymenakos, K., Abate, A., Roberts, S.: Safe policy search using gaussian process models. In: Proceedings of AAMAS. pp. 1565–1573 (2019)
43. Precup, D.: Temporal abstraction in reinforcement learning. Ph.D. thesis, University of Massachusetts Amherst (2001)
44. Sadigh, D., Kim, E.S., Coogan, S., Sastry, S.S., Seshia, S.A.: A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In: CDC. pp. 1091–1096 (2014)
45. Sickert, S., Esparza, J., Jaax, S., Křetínskỳ, J.: Limit-deterministic Büchi automata for linear temporal logic. In: CAV. pp. 312–332 (2016)
46. Sickert, S., Křetínskỳ, J.: MoChiBA: Probabilistic LTL model checking using limit-deterministic Büchi automata. In: ATVA. pp. 130–137 (2016)
47. Silver, D., Lever, G., Heess, N., Thomas Degris, D.W., Riedmiller, M.: Deterministic policy gradient algorithms. ICML (2014)
48. Squyres, S.W., Knoll, A.H., Arvidson, R.E., Ashley, J.W., Bell, J., Calvin, W.M., Christensen, P.R., Clark, B.C., Cohen, B.A., De Souza, P., et al.: Exploration of Victoria crater by the Mars rover Opportunity. Science **324**(5930), 1058–1061 (2009)
49. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, vol. 1. MIT press Cambridge (1998)
50. Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D.d.L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al.: Deepmind control suite. arXiv preprint arXiv:1801.00690 (2018)
51. Toro Icarte, R., Klassen, T.Q., Valenzano, R., McIlraith, S.A.: Teaching multiple tasks to an RL agent using LTL. In: AAMAS. pp. 452–461 (2018)
52. Vezhnevets, A., Mnih, V., Osindero, S., Graves, A., Vinyals, O., Agapiou, J., et al.: Strategic attentive writer for learning macro-actions. In: NIPS. pp. 3486–3494 (2016)
53. Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**(3-4), 279–292 (1992)
54. Wei, Q., Guo, X.: Markov decision processes with state-dependent discount factors and unbounded rewards/costs. Operations Research Letters **39**(5), 369–374 (2011)
55. Yan, R., Cheng, C.H., Chai, Y.: Formal consistency checking over specifications in natural languages. In: DATE. pp. 1677–1682 (2015)
56. Yoshida, N., Uchibe, E., Doya, K.: Reinforcement learning with state-dependent discount factor. In: 2013 IEEE third joint international conference on development and learning and epigenetic robotics (ICDL). pp. 1–6. IEEE (2013)
57. Yuan, L.Z., Hasanbeig, M., Abate, A., Kroening, D.: Modular Deep Reinforcement Learning with Temporal Logic Specifications. arXiv preprint arXiv:1909.11591 (2019)

Appendix: Proof of Theorem 1

Theorem 1. ***Let φ be a given LTL formula and $\mathfrak{M}_{\mathfrak{A}}$ be the product MDP constructed by synchronising the MDP $\mathfrak{M}$ with the LDBA $\mathfrak{A}$ associated with φ. Then the optimal stationary Markov policy on $\mathfrak{M}_{\mathfrak{A}}$ that maximises the expected return, maximises the probability of satisfying φ and induces a finite-memory policy on the MDP $\mathfrak{M}$.***

Proof. Assume that the optimal Markov policy on $\mathfrak{M}_{\mathfrak{A}}$ is ${\pi^{\otimes}}^*$, namely at each state $s^{\otimes}$ in $\mathfrak{M}_{\mathfrak{A}}$ we have

$$ {\pi^{\otimes}}^*(s^{\otimes}) = \underset{\pi^{\otimes} \in \mathcal{D}^{\otimes}}{\arg\sup}\ U^{\pi^{\otimes}}(s^{\otimes}) = \underset{\pi^{\otimes} \in \mathcal{D}^{\otimes}}{\arg\sup}\ \mathbb{E}^{\pi^{\otimes}}[\sum_{n=0}^{\infty} \gamma^n\ R(s_n^{\otimes}, a_n)|s_0^{\otimes} = s^{\otimes}], \quad (15) $$

where $\mathcal{D}^{\otimes}$ is the set of stationary deterministic policies over the state space $\mathcal{S}^{\otimes}$, $\mathbb{E}^{\pi^{\otimes}}[\cdot]$ denotes the expectation given that the agent follows policy $\pi^{\otimes}$, and $s_0^{\otimes}, a_0, s_1^{\otimes}, a_1, \ldots$ is a generic path generated by the product MDP under policy $\pi^{\otimes}$.

Recall that an infinite word $w \in \Sigma^{\omega},\ \Sigma = 2^{\mathcal{AP}}$ is accepted by the LDBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ if there exists an infinite run $\theta \in \mathcal{Q}^{\omega}$ starting from q_0 where $\theta[i+1] \in \Delta(\theta[i], \omega[i]),\ i \geq 0$ and, for each $F_j \in \mathcal{F},\ \mathit{inf}(\theta) \cap F_j \neq \emptyset$, where $\mathit{inf}(\theta)$ is the set of states that are visited infinitely often in the sequence θ. From Definition 8, the associated run θ of an infinite path in the product MDP $\rho = s_0^{\otimes} \xrightarrow{a_0} s_1^{\otimes} \xrightarrow{a_1} \ldots$ is $\theta = L^{\otimes}(s_0^{\otimes})L^{\otimes}(s_1^{\otimes})\ldots$. From Definition 9 and (10), and since for an accepting run $\mathit{inf}(\theta) \cap F_j \neq \emptyset,\ \forall F_j \in \mathcal{F}$, all accepting paths starting from $s_0^{\otimes}$, accumulate infinite number of positive rewards r_p (see Remark 2).

In the following, by contradiction, we show that any optimal policy ${\pi^{\otimes}}^*$ satisfies the property with maximum possible probability. Let us assume that there exists a stationary deterministic Markov policy ${\pi^{\otimes}}^+ \neq {\pi^{\otimes}}^*$ over the state space $\mathcal{S}^{\otimes}$ such that probability of satisfying φ under ${\pi^{\otimes}}^+$ is maximum.

This essentially means in the product MDP $\mathfrak{M}_{\mathfrak{A}}$ by following ${\pi^{\otimes}}^+$ the expectation of reaching the point where $\mathit{inf}(\theta) \cap F_j \neq \emptyset,\ \forall F_j \in \mathcal{F}$ and positive reward is received ever after is higher than any other policy, including ${\pi^{\otimes}}^*$. With a tuned discount factor γ, e.g. (1),

$$ \mathbb{E}^{{\pi^{\otimes}}^+}[\sum_{n=0}^{\infty} \gamma^n\ R(s_n^{\otimes}, a_n)|s_0^{\otimes} = s^{\otimes}] > \mathbb{E}^{{\pi^{\otimes}}^*}[\sum_{n=0}^{\infty} \gamma^n\ R(s_n^{\otimes}, a_n)|s_0^{\otimes} = s^{\otimes}] \quad (16) $$

This is in contrast with optimality of ${\pi^{\otimes}}^*$ (15) and concludes ${\pi^{\otimes}}^* = {\pi^{\otimes}}^+$. Namely, an optimal policy that maximises the expected return also maximises the probability of satisfying LTL property φ. It is easy to see that the projection of policy ${\pi^{\otimes}}^*$ on MDP $\mathfrak{M}$ is a finite-memory policy π^*. □