Logically-Constrained Neural Fitted Q-iteration

Extended Abstract

Mohammadhosein Hasanbeig
University of Oxford
Oxford, United Kingdom
hosein.hasanbeig@cs.ox.ac.uk

Alessandro Abate
University of Oxford
Oxford, United Kingdom
alessandro.abate@cs.ox.ac.uk

Daniel Kroening
University of Oxford
Oxford, United Kingdom
daniel.kroening@cs.ox.ac.uk

ABSTRACT

We propose a method for efficient training of Q-functions for continuous-state Markov Decision Processes (MDPs), such that the traces of the resulting policies satisfy a given Linear Temporal Logic (LTL) property. LTL, a modal logic, can express a wide range of time-dependent logical properties (including *safety*) that are quite similar to patterns in natural language. We convert the LTL property into a limit deterministic Büchi automaton and construct an on-the-fly synchronised product MDP. The control policy is then synthesised by defining an adaptive reward function and by applying a modified neural fitted Q-iteration algorithm to the synchronised structure, assuming that no prior knowledge is available from the original MDP (namely, the method is model-free). The proposed method is evaluated in a numerical study to test the quality of the generated control policy and is compared with conventional methods for policy synthesis, such as MDP abstraction (Voronoi quantizer) and approximate dynamic programming (fitted value iteration).

KEYWORDS

Reinforcement Learning; Safety; Formal Methods; Neural Networks

ACM Reference Format:
Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. 2019. Logically-Constrained Neural Fitted Q-iteration. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 3 pages.

1 INTRODUCTION

Reinforcement Learning (RL) is a successful paradigm for training an autonomous agent to make optimal decisions when interacting with an MDP, when the stochastic behaviour of the MDP is unknown. Conventional RL is mostly focused on problems where the set of states of the MDP and the set of possible actions are finite. Nonetheless, many interesting real-world problems require actions to be taken in response to high-dimensional or real-valued sensory inputs [4]. A number of solutions have been proposed to deal with MDPs with large or uncountably infinite state spaces, e.g., CMACs [21], kernel-based modelling [16], tree-based regression [6], generalisation via basis functions [3], etc. Among these methods, neural networks stand out thanks to their ability to construct a generalisation of any non-linear function over continuous (or a very large discrete) domains [12], relying only on a set of samples from that domain. This feature has attracted significant attention in the machine learning community, resulting in numerous applications of neural networks on infinite- or large-state space MDPs, e.g., Deep Q-networks [15], TD-Gammon [23], Asynchronous Deep RL [14], Neural Fitted Q-iteration [17] and CACLA [25].

In this paper, we propose the first formal architecture that enables RL to generate policies that satisfy an arbitrary given Linear Temporal Logic (LTL) property for *continuous-state* MDPs. LTL enables the formalisation of complex mission requirements with a rich time-dependent language and can express sophisticated high-level control objectives that are hard (if at all possible) to achieve by on standard reward-based setups [20, 22] or by recent methods such as Policy Sketching [2]. The most immediate application of LTL in RL is for *safe learning*: for this use case, we show that the proposed architecture is efficient and that it can be combined easily with recently developed RL algorithms, and specifically with architectures apt at dealing with continuous-space MDPs. Detailed discussions and further in-depth analysis can be found in [9].

There exist a number of algorithms that can synthesise policies with LTL constrains in *finite-state* MDPs by employing model-based RL (e.g., [7]). Logic-based synthesis for continuous-space models has been so far limited to DP- or optimisation-based techniques [1, 8, 24]. However, the first model-free formal LTL-constrained RL algorithm for fully unknown finite-state MDPs has appeared in [10]. [10] further shows that the RL procedure sets up an off-policy local value iteration method to efficiently calculate the maximum probability of satisfying the given property, at any given state of the MDP. The work in [10] has been extended to [11] with in-depth discussions on theory and also additional experiments.

2 PRELIMINARIES

Definition 2.1 (Continuous-state Space MDP (CSMDP)). The tuple $\mathbf{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ is a CSMDP over a set of states $\mathcal{S} = \mathbb{R}^n$, where $\mathcal{A}$ is a finite set of actions, s_0 is the initial state and $P : \mathcal{B}(\mathbb{R}^n) \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a Borel-measurable transition kernel that assigns to any state and any action a probability measure on the Borel space $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ [5]. $\mathcal{AP}$ 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 $L(s) \subseteq 2^{\mathcal{AP}}$ [5]. ⌟

When analysing a CSMDP, the standard finite look-up table used in classical RL algorithms such as Q-learning (QL) is of no use. We aim to preserve the key benefit of QL, namely to synthesise policies from finite experiences and without maintaining a model of the MDP. Neural Fitted Q-iteration (NFQ) [17] achieves this by employing a multi-layer perceptron [13] to approximate the Q-function over the set of experience samples. NFQ is the key algorithm behind Deep Reinforcement Learning [15].

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

Definition 2.2 (Policy Satisfaction). Given an LTL[1] formula φ, we state that a stationary deterministic policy $Pol : \mathcal{S} \to \mathcal{A}$ satisfies an LTL formula φ if $\mathbb{P}[L(s_0)L(s_1)L(s_2)\ldots \in Words(\varphi)] > 0$, where every transition $s_i \to s_{i+1},\ i = 0, 1, \ldots$ is executed by taking action $Pol(s_i)$ at state s_i. ⌟

An LTL formula can be expressed by various finite-state automata. We employ *Limit-Deterministic Büchi Automata* (LDBA), which yield a succinct representation, enable fast convergence, and are applicable with MDPs.

Definition 2.3. [Limit Deterministic Büchi Automaton (LDBA)] An LDBA is a special case of Generalised Büchi Automaton (GBA). Formally, a GBA $\mathbf{N} = (Q, q_0, \Sigma, \mathcal{F}, \Delta)$ is a structure where Q is a finite set of states, $q_0 \subseteq Q$ 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 $F_j \subset Q, 1 \leq j \leq f$, and $\Delta : Q \times \Sigma \to 2^Q$ is a transition relation. A GBA $\mathbf{N} = (Q, q_0, \Sigma, \mathcal{F}, \Delta)$ is limit deterministic if Q can be partitioned into two disjoint sets $Q = Q_N \cup Q_D$, such that (1) $\Delta(q, \alpha) \subseteq Q_D$ and $|\Delta(q, \alpha)| = 1$ for every state $q \in Q_D$ and for every corresponding $\alpha \in \Sigma$, (2) for every $F_j \in \mathcal{F}, F_j \subset Q_D$ [19]. ⌟

3 LOGICALLY-CONSTRAINED NFQ

In this section, we propose an NFQ-based algorithm that is able to synthesise a policy (or a set thereof) that satisfies a temporal logic property. We call this algorithm Logically-Constrained NFQ (LCNFQ). We formally relate the MDP and the given automaton by synchronising them *on-the-fly*, creating a new structure that enforces the logical property and that furthermore is prone to RL algorithms.

Definition 3.1 (Product MDP). Given an MDP $\mathbf{M}(\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ and an LDBA $\mathbf{N}(Q, q_0, \Sigma, \mathcal{F}, \Delta)$ with $\Sigma = 2^{\mathcal{AP}}$, the product MDP is defined as $(\mathbf{M} \otimes \mathbf{N}) = \mathbf{M_N} = (\mathcal{S}^\otimes, \mathcal{A}, s_0^\otimes, P^\otimes, \mathcal{AP}^\otimes, L^\otimes, \mathcal{F}^\otimes)$, where $\mathcal{S}^\otimes = \mathcal{S} \times Q$, $s_0^\otimes = (s_0, q_0)$, $\mathcal{AP}^\otimes = Q$, $L^\otimes = \mathcal{S} \times Q \to 2^Q$ 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 is (s_j, q_j), where $s_j \sim P(\cdot|s_i, a)$ and $q_j \in \Delta(q_i, L(s_j))$. ⌟

By synchronising MDP states with LDBA states in the product MDP we add an extra dimension to the state space of the original MDP. The role of the added dimension is to track the automaton state and, hence, to evaluate the satisfaction of the corresponding LTL property. Note that LCNFQ is fully *model-free* and there is no need to construct the product MDP explicitly. The synchronised product MDP subsumes the transition relations of the original MDP and the structure of the Büchi automaton, thus it inherits characteristics of both. Thus, an appropriate reward function[2] can guide RL to find a policy that is optimal with respect to satisfaction of the LTL property φ on the MDP.

We now discuss generalisation issues. In LCNFQ, we employ n separate multi-layer perceptrons with one hidden layer where $n = |Q|$ and Q is the finite cardinality of the automaton $\mathbf{N}$. Each neural net is associated with a state in the LDBA and together the neural nets approximate the Q-function in the product MDP. For each automaton state $q_i \in Q$ the associated neural net is called $B_{q_i} : \mathcal{S}^\otimes \times \mathcal{A} \to \mathbb{R}$. Once the agent is at state $s^\otimes = (s, q_i)$ the neural net B_{q_i} is used for the local Q-function approximation. The set of neural nets acts as a global Q-function approximator $Q : \mathcal{S}^\otimes \times \mathcal{A} \to \mathbb{R}$. Note that the neural nets are not decoupled from each other. For example, assume that by taking action a in state $s^\otimes = (s, q_i)$ the agent is moved to state $s^{\otimes\prime} = (s', q_j)$ where $q_i \neq q_j$. The weights of B_{q_i} are updated such that $B_{q_i}(s^\otimes, a)$ has minimum possible error to $R(s^\otimes, a) + \gamma \max_{a'} B_{q_j}(s^{\otimes\prime}, a')$. Therefore, the value of $B_{q_j}(s^{\otimes\prime}, a')$ affects $B_{q_i}(s^\otimes, a)$.

Let $q_i \in Q$ be a state in the LDBA. Then define $\mathcal{E}_{q_i} := \{(\cdot, \cdot, \cdot, \cdot, x) \in \mathcal{E} | x = q_i\}$ as the set of experiences within $\mathcal{E}$ that is associated with state q_i, i.e., $\mathcal{E}_{q_i}$ is the projection of $\mathcal{E}$ onto q_i. Once the experience set $\mathcal{E}$ is gathered, each neural net B_{q_i} is trained by its associated experience set $\mathcal{E}_{q_i}$. At each iteration a pattern set $\mathcal{P}_{q_i}$ is generated based on $\mathcal{E}_{q_i}$: $\mathcal{P}_{q_i} = \{(input_l, target_l), l = 1, ..., |\mathcal{E}_{q_i}|)\}$, where $input_l = ({s_l}^\otimes, a_l)$ and $target_l = R({s_l}^\otimes, a_l) + \gamma \max_{a' \in \mathcal{A}} Q({s_l}^{\otimes\prime}, a')$ such that $({s_l}^\otimes, a_l, {s_l}^{\otimes\prime}, R({s_l}^\otimes, a_l), q_i) \in \mathcal{E}_{q_i}$. The pattern set is used to train the neural net B_{q_i}. We use Rprop [18] to update the weights in each neural net, as it is known to be an efficient method for batch learning [17]. In each cycle of LCNFQ, the training schedule starts from networks that are associated with accepting states of the automaton and goes backward until it reaches the networks that are associated to the initial states. In this way, we allow the Q-value to back-propagate through the product MDP. LCNFQ stops when the generated policy stops improving for long enough.

4 EXPERIMENTS AND DISCUSSION

We have tested the performance of LCNFQ in a mission planning task for an autonomous Mars rover, and benchmarked it against alternatives, namely the use of a Voronoi Quantiser (VQ) and the use of Fitted Value Iteration (FVI). The results are presented in Table 1, where LCNFQ has outperformed the alternative algorithms. LCNFQ exploits the positive effects of generalisation in neural networks, while at the same time it avoids the negative effects of disturbing previously learned experiences. This means that LCNFQ requires less experience and the learning process is highly data efficient, which leads to increases in scalability and applicability of the proposed algorithm. Let us conclude emphasising that LCNFQ is model-free, meaning that the learning only depends on the sample experiences that the agent gathered by interacting and exploring the MDP.

[1]Syntax and semantics of LTL are thoroughly discussed in [9].
[2]Details can be found in [9].

REFERENCES

[1] A. Abate, J.P. Katoen, J. Lygeros, and M. Prandini. 2010. Approximate Model Checking of Stochastic Hybrid Systems. *European Journal of Control* 16, 6 (2010), 624–641.
[2] Jacob Andreas, Dan Klein, and Sergey Levine. 2017. Modular Multitask Reinforcement Learning with Policy Sketches. In *ICML*, Vol. 70. 166–175.
[3] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. 2010. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Vol. 39. CRC press.
[4] Kenji Doya. 2000. Reinforcement Learning in Continuous Time and Space. *Neural computation* 12, 1 (2000), 219–245.
[5] Richard Durrett. 1999. *Essentials of stochastic processes*. Vol. 1. Springer.
[6] Damien Ernst, Pierre Geurts, and Louis Wehenkel. 2005. Tree-based Batch Mode Reinforcement Learning. *JMLR* 6, Apr (2005), 503–556.
[7] Jie Fu and Ufuk Topcu. 2014. Probably Approximately Correct MDP Learning and Control With Temporal Logic Constraints. In *Robotics: Science and Systems X*.

Table 1: Simulation results

Experiment 1					
Algorithm	**Sample Complexity**	$U^{Pol^*}(s_0)$	**Success Rate**†	**Training Time*****(s)**	**Iteration Num.**
LCNFQ	**7168** samples	**0.0203**	**99%**	95.64	**40**
VQ ($\Delta = 0.4$)	27886 samples	0.0015	**99%**	1732.35	2195
VQ ($\Delta = 1.2$)	7996 samples	0.0104	97%	273.049	913
VQ ($\Delta = 2$)	-	0	0%	-	-
FVI	40000 samples	0.0133	98%	**4.12**	80
Experiment 2					
Algorithm	**Sample Complexity**	$U^{Pol^*}(s_0)$	**Success Rate**†	**Training Time*****(s)**	**Iteration Num.**
LCNFQ	**2680 samples**	**0.1094**	**98%**	166.13	**40**
VQ ($\Delta = 0.4$)	8040 samples	0.0082	**98%**	3666.18	3870
VQ ($\Delta = 1.2$)	3140 samples	0.0562	96%	931.33	2778
VQ ($\Delta = 2$)	-	0	0%	-	-
FVI	25000 samples	0.0717	97%	**2.16**	80

† Testing the trained agent for 100 trials * Average for 10 ep

[8] S. Haesaert, S.E.Z. Soudjani, and A. Abate. 2017. Verification of general Markov decision processes by approximate similarity relations and policy refinement. *SIAM Journal on Control and Optimisation* 55, 4 (2017), 2333–2367.

[9] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. 2018. Logically-Constrained Neural Fitted Q-Iteration. *arXiv preprint arXiv:1809.07823* (2018).

[10] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. 2018. Logically-Constrained Reinforcement Learning. *arXiv preprint arXiv:1801.08099* (2018).

[11] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. 2019. Certified Reinforcement Learning with Logic Guidance. *arXiv preprint arXiv:1902.00778* (2019).

[12] Kurt Hornik. 1991. Approximation Capabilities of Multilayer Feedforward Networks. *Neural networks* 4, 2 (1991), 251–257.

[13] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[14] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *ICML*. 1928–1937.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level Control Through Deep Reinforcement Learning. *Nature* 518, 7540 (2015), 529–533.

[16] Dirk Ormoneit and Śaunak Sen. 2002. Kernel-based Reinforcement Learning. *Machine learning* 49, 2 (2002), 161–178.

[17] Martin Riedmiller. 2005. Neural Fitted Q iteration-First Experiences with a Data Efficient Neural Reinforcement Learning Method. In *ECML*, Vol. 3720. Springer, 317–328.

[18] Martin Riedmiller and Heinrich Braun. 1993. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In *Neural networks*. IEEE, 586–591.

[19] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínskỳ. 2016. Limit-deterministic Büchi automata for linear temporal logic. In *CAV*. Springer, 312–332.

[20] Stephen L Smith, Jana Tumová, Calin Belta, and Daniela Rus. 2011. Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research* 30, 14 (2011), 1695–1708.

[21] Richard S Sutton. 1996. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In *NIPS*. 1038–1044.

[22] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[23] Gerald Tesauro. 1995. TD-Gammon: A self-teaching Backgammon Program. In *Applications of Neural Networks*. Springer, 267–285.

[24] Ilya Tkachev, Alexandru Mereacre, Joost-Pieter Katoen, and Alessandro Abate. 2017. Quantitative model-checking of controlled discrete-time Markov processes. *Information and Computation* 253 (2017), 1–35.

[25] Hado Van Hasselt and Marco A Wiering. 2007. Reinforcement Learning in Continuous Action Spaces. In *ADPRL*. IEEE, 272–279.