DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning

Submission 8323

Abstract

We propose a method for effective training of deep Reinforcement Learning (RL) agents when the reward is sparse and non-Markovian, but at the same time progress towards the reward requires achieving an unknown *sequence* of high-level objectives. Our method employs a novel algorithm for synthesis of compact automata to uncover this sequential structure automatically. We synthesise a human-interpretable automaton from trace data generated through exploration of the environment by the deep RL agent. The state space of the environment is then enriched with the synthesised automaton so that generation of an optimal control policy by deep RL is guided by the discovered structure encoded in the automaton. We evaluate performance via a set of experiments including the Atari game *Montezuma's Revenge*. Compared to existing approaches, we obtain a decrease of *two* orders of magnitude in the number of iterations required for policy synthesis.

1 Introduction

Reinforcement Learning (RL) is the key enabling technique for a variety of applications of artificial intelligence, including advanced robotics (Sutton, Precup, and Singh 1999), resource (Mao et al. 2016) and traffic management (Sadigh et al. 2014), drone control (Abbeel et al. 2007), chemical engineering (Zhou, Li, and Zare 2017), and gaming (Mnih et al. 2015). While RL is a very general architecture, many advances in the last decade have been achieved using specific instances of RL that employ a deep neural network to synthesise optimal policies. A deep RL algorithm, AlphaGo (Silver et al. 2016), played moves in the game of Go that were initially considered glitches by human experts, but secured victory against the world champion master. Similarly, AlphaStar (Vinyals et al. 2019) was able to defeat the world's best players at the real-time strategy game StarCraft II, and to reach top 0.2% in scoreboards with an "unimaginably unusual" playing style.

Whilst Deep RL can autonomously solve many tasks in complex environments, tasks that feature extremely sparse, non-Markovian rewards or other long-term sequential structures are often difficult or impossible to solve by unaided RL. A well-known example is the Atari game *Montezuma's Revenge*, in which deep RL methods (Mnih et al. 2015) fail to score even once. Interestingly, Montezuma's Revenge and many other hard problems often require learning to achieve, possibly in a specific sequence, a set of high-level objectives to obtain the reward. The accomplishment of these objectives can often be identified with passing through designated and semantically distinguished states of the system, and this insight can be a lever that enables us to obtain a manageable, high-level model of the system's behaviour and its dynamics.

In this paper we propose a new framework that automatically infers sequential dependencies of a reward on high-level objectives and exploits this to guide a deep RL agent when the reward signal is history-dependent and significantly delayed. Identification of sequential dependencies on high-level objectives is the key to breaking down a complex task into a sequence of many Markovian ones. In our work, we use automata expressed in terms of high-level objectives to orchestrate sequencing of low-level actions in deep RL and to guide the learning towards sparse rewards.

At the heart of our method is a *model-free* deep RL algorithm that is synchronised in a closed-loop fashion with an automaton inference algorithm, enabling our method to learn a policy that discovers and follows high-level sparse-reward structures. The synchronization is achieved by a product construction, which creates a hybrid architecture for the deep neural-fitted RL. The technical details are in Section 5, where we illustrate our framework with Montezuma's Revenge as the running example. In Section 6 we evaluate the performance of our framework on a selection of benchmarks with sequential and unknown high-level structures. These experiments show that our method is able to *automatically* discover and formalise unknown, sparse, and non-Markovian high-level reward structures to efficiently synthesise successful policies when other deep RL approaches fail.

2 Related Work

This work takes a formal approach to tackle the sparse reward problem in RL. In the RL literature dependencies of rewards upon objectives are often called *options* (Sutton and Barto 1998) and can, in general, be hierarchically structured. Options can be embedded into general learning algorithms to address the problem of sparse rewards. But current approaches to hierarchical RL very much depend on state representations and whether they are structured enough for a suitable reward signal to be effectively engineered manually. Hierarchical RL therefore often requires detailed supervision in the form of explicitly specified high-level actions or intermediate supervisory signals (Precup 2001; Kearns and Singh 2002; Daniel, Neumann, and Peters 2012; Kulkarni et al. 2016; Vezhnevets et al. 2016; Bacon, Harb, and Precup 2017). A key difference between our approach and the options framework is that our method produces a modular, human-interpretable and succinct automaton structure to represent the sequence of tasks, as opposed to complex and sample-inefficient structures such as RNNs (Vezhnevets et al. 2017).

The closest line of work to ours, which aims to avoid these requirements, are recent model-based (Fu and Topcu 2014; Sadigh et al. 2014) or model-free (Hasanbeig, Abate, and Kroening 2018; Toro Icarte et al. 2018; De Giacomo et al. 2019; Hahn et al. 2019; Kazemi and Soudjani 2020) approaches in RL that constrain the agent with a temporal logic property. These approaches are limited to finite-state systems, and more importantly require the temporal sequence to be known a priori. This assumption is relaxed in (Icarte et al. 2019), where an automaton is inferred from deep RL exploration traces. Automata inference in (Icarte et al. 2019) is done using a local search-based algorithm, Tabu search (Glover and Laguna 1998). Conversely, our algorithm for automata inference uses a bactracking search algorithm DPLL (Davis and Putnam 1960). Although local-search-based algorithms are faster, the DPLL algorithm is complete and explores the entire search space efficiently (Cook and Mitchell 1996) and hence finds more accurate representations of the discovered traces. Further related work is *policy sketching* (Andreas, Klein, and Levine 2017), which learns feasible tasks first and then stitches them together to accomplish a complex task. The key problem is that this method assumes that the policy sketches (equivalent to automaton components) are given, which may be unrealistic and is not assumed in this work.

Inferred automata have been used to learn strategies for infinite two-person games where strategies are a function of previously visited states. The construction of *chain automata* for these games provided a means to implement memory-less strategies (Krishnan et al. 1995). But these chain automata had a disproportionally large number of states as compared to the size of the ω-automaton the game was played on. There has also been recent work on learning underlying objectives when an optimal policy or human demonstration is available, e.g. (Koul, Greydanus, and Fern 2018; Memarian et al. 2020).

The most common approach to synthesising automata from traces is *state merging* (Biermann and Feldman 1972). A variant of the state merge algorithm, Evidence-Driven State Merge (EDSM) (Lang, Pearlmutter, and Price 1998), uses both positive and negative instances of behaviour to determine equivalence of states to be merged based on statistical evidence. Some approaches use SAT together with state merge to generate automata from positive and negative traces (Ulyantsev and Tsarev 2011; Heule and Verwer 2013; Ulyantsev, Buzhinsky, and Shalyto 2016; Buzhinsky and Vyatkin 2017; Buzhinsky and Vyatkin 2017). To avoid over-generalisation in the absence of labelled data, the EDSM algorithm was improved to incorporate inherent temporal behaviour (Walkinshaw et al. 2007; Walkinshaw and Bogdanov 2008), which needs to be known a priori. These algorithms do not focus on producing the most succinct automaton but rather on generating a good enough approximation that conforms to the trace (Ulyantsev, Buzhinsky, and Shalyto 2016), which is not the best fit for our framework. The classic automata learning technique, Angluin's L* algorithm (Angluin 1987), employs a series of equivalence and membership queries to an oracle, the results of which are used to construct the automaton. The absence of an oracle restricts the use of this algorithm in our setting.

3 Background on Reinforcement Learning

We first consider a conventional RL setup, consisting of an agent interacting with an environment, which is modelled as an unknown general Markov Decision Process (MDP) with a Markovian reward.

Definition 3.1 (General MDP) *The tuple* $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \Sigma, L)$ *is a general MDP over a set of continuous states* $\mathcal{S} = \mathbb{R}^n$*, where* $\mathcal{A}$ *is a set of finite 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 that 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* $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ *(Bertsekas and Shreve 2004).* Σ *is called the vocabulary set and is a finite set of atomic propositions for which there exists a labelling function* $L : \mathcal{S} \to 2^{\Sigma}$ *that assigns to each state* $s \in \mathcal{S}$ *a set of atomic propositions* $L(s) \in 2^{\Sigma}$.

Definition 3.2 (Path) *In a general MDP* $\mathfrak{M}$*, 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 possible in* $\mathfrak{M}$*, i.e.,* s_{i+1} *belongs to the smallest Borel set* B *such that* $P(B|s_i, a_i) = 1$*. A finite path* $\rho_n = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ... \xrightarrow{a_{n-1}} s_n$ *is a prefix of an infinite path. The set of infinite paths is* $(\mathcal{S} \times \mathcal{A})^{\omega}$ *and the set of finite paths is* $(\mathcal{S} \times \mathcal{A})^*$.

At each state $s \in \mathcal{S}$, an agent action is determined by a policy π, which is a mapping from states to a probability distribution over the actions, i.e., $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. Further, a random variable $R(s,a) \sim \rho(\cdot|s,a) \in \mathcal{P}(\mathbb{R})$ is defined over the MDP $\mathfrak{M}$, to represent the Markovian reward obtained when action a is taken in a given state s, where $\mathcal{P}(\mathbb{R})$ is the set of probability distributions on subsets of $\mathbb{R}$, and ρ is the reward distribution. Similarly, a non-Markovian reward $\widehat{R} : (\mathcal{S} \times \mathcal{A})^* \to \mathbb{R}$ is a mapping from the set of finite paths to real numbers and one possible realisation of R and $\widehat{R}$ at time step n is denoted by r_n and $\widehat{r}_n$ respectively.

Due to space limitations we present the formal background on RL in the Appendix and we only introduce the notations in the following. The expected discounted return for a policy π and state s is denoted by $U^{\pi}(s)$ which is maximised by the optimal policy π^*. Similarly, at each state the optimal policy maximises the Q-function $Q(s,a)$ over the set of actions. The Q-function can be parameterised using a parameter set θ^Q and updated by minimising a loss function $\mathfrak{L}(\theta^Q)$.

4 Background on Automata Synthesis

Automatic inference of the high-level sequential structure used to guide learning is achieved using automata synthesis.

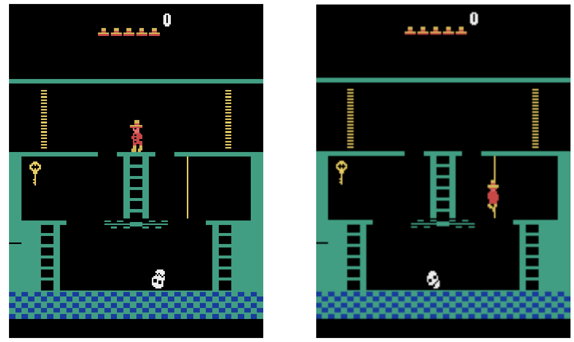
The synthesis algorithm uses trace sequences to construct an automaton that represents the behaviour exemplified by them. The required automaton is generated using a *synthesis from examples* approach (Gulwani 2012; Jeppu et al. 2020). It is a scalable method for learning finite-state models from trace data that produce abstract, concise models. The automata synthesis framework we use makes an algorithmic improvement to scale to long traces by means of a segmentation approach, thus achieving automata learning in close-to-polynomial runtime as supported by empirical evidence.

The synthesis framework takes as input a trace sequence and an optional parameter N for the number of automaton states. By default, N is set to two states. The framework additionally employs two hyper-parameters, w and l, that can be tuned based on the requirement. The hyper-parameter w is used to tackle growing algorithm complexity for long trace input. The synthesis framework divides the trace into segments using a sliding window of size w and unique segments are used for further processing. Here, the framework looks for the presence of patterns in traces. Multiple occurrences of these patterns required to be processed only once, thus reducing size of the input to the algorithm. The second hyper-parameter l is used to control the degree of generalisation in the generated automaton. Automata generated using only 'positive' trace samples tend to overgeneralise (Gold 1978). This is handled by performing a compliance check of the automaton against the trace input to eliminate any transition sequences of length l that are present in the generated automaton but do not appear in the trace. Further details regarding choice of hyper-parameter values is provided in Section 5.

5 DeepSynth

We begin by introducing the first level of Montezuma's Revenge as a running example (Bellemare et al. 2013). Unlike other Atari games where the primary goal is limited to avoiding obstacles or collecting items with no particular order, Montezuma's Revenge requires the agent to perform a long, complex sequence of actions before receiving any reward at all. The agent must find a key and open either door in Fig.1.a. To that end, the agent has to climb down the middle ladder, jump on the rope, climb down the ladder on the right and jump over a skull to reach the key. The reward given by the Atari emulator for collecting the key is 100 and reward for opening one of the doors is another 300. The agent therefore has to perform a long, complex sequence of actions before receiving any reward at all. Due to this reward sparsity most of the existing deep RL algorithms either fail to learn a policy that can even reach the key, e.g. DQN (Mnih et al. 2015), or the learning process is computationally heavy and sample inefficient, e.g. FeUdal (Vezhnevets et al. 2017), and Go-Explore (Ecoffet et al. 2019).

To overcome this problem various methods have been proposed that mostly hinge on intrinsic motivation and object-driven guidance. Unsupervised object detection (or unsupervised semantic segmentation) from raw image input has seen substantial progress in recent years, and became comparable to its supervised counterpart (Ji, Henriques, and Vedaldi 2019; Hwang et al. 2019; Zheng and Yang 2020). In this work, we assume that an image segmentation algorithm can



(a) (b)

Figure 1: (a) the first level of Atari 2600 Montezuma's Revenge; (b) pixel overlap of two segmented objects.

provide plausible object candidates. The key to solve a complex task such as Montezuma's Revenge is discovering the semantic correlations between the candidate objects in the scene. In comparison, when a human player tries to solve this game the semantic correlations, such as "keys open doors", are partially known and the player's behaviour is driven by exploiting these known correlations and exploring unknown objects. This has been a subject of study in psychology, where animals and humans seem to have general motivations (often referred to as intrinsic motivations) that push them to explore and manipulate their environment, encouraging curiosity and cognitive growth (Berlyne 1960; Csikszentmihalyi and Csikzentmihaly 1990; Ryan and Deci 2000).

Owing to the intuitive structure of the automaton, DeepSynth can embed known correlations (if there exists any) into the learning algorithm for exploitation and pushes the exploration to infer the unknown parts. Such automaton-based exploration scheme imitates biological cognitive growth in a formal and explainable way, and is driven by an intrinsic motivation to explore as many objects as possible in order to find the extrinsic optimal sequence of high-level objectives. To evaluate the full potential of DeepSynth, in all the experiments and examples of this paper we assume that semantic correlations are entirely unknown to the agent. The agent starts with neither knowledge of the sparse reward task nor the correlation of the high-level objects.

Let us denote the set of detected objects as Σ. Note that the semantic meaning of the names for individual objects is not of relevant to the algorithm and Σ can thus be a list of distinct identifiers, e.g. $\Sigma = \{\mathtt{obj}_1, \mathtt{obj}_2, ...\}$. However, for the sake of exposition we name the objects according to their actual context in Fig. 1.a, i.e. $\Sigma = \{\mathtt{red\_character}, \mathtt{middle\_ladder}, \mathtt{rope}, \mathtt{right\_ladder}, \mathtt{left\_ladder}, \mathtt{key}, \mathtt{door}\}$. Note that the number of detected objects can be arbitrarily finite as long as the input image is segmented into enough objects whose correlation can guide the agent to achieve the task.

Recall that the task is unknown initially and the extrinsic reward is non-Markovian and extremely sparse. Namely, the agent receives a positive reward $\widehat{R} : (\mathcal{S} \times \mathcal{A})^* \rightarrow \mathbb{R}$ only when a correct sequence of state-action pairs and their asso-

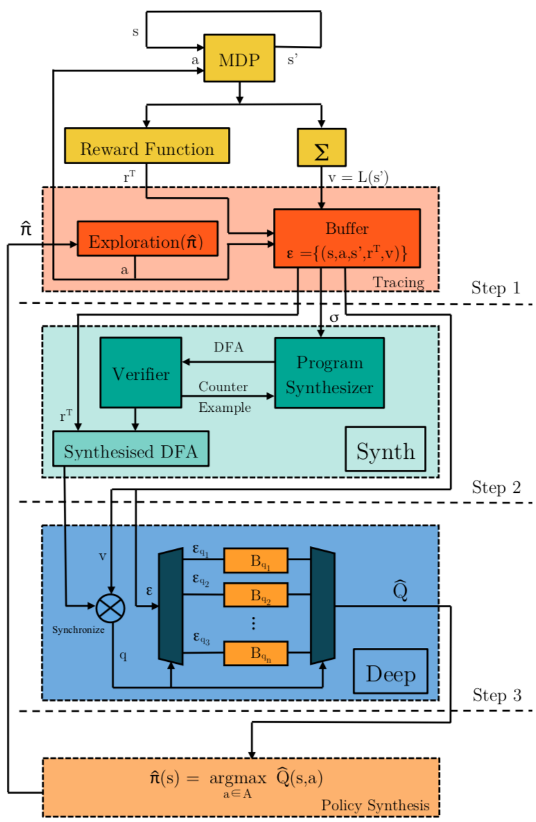


Figure 2: DeepSynth framework

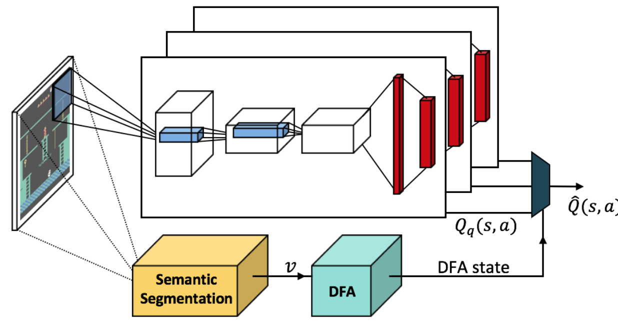


Figure 3: DeepSynth for Montezuma's Revenge: each DQN module is forced by the DFA to focus on a semantically distinct object. The input to the first layer in DQN modules is the input image which is convolved by 32 filters of 8×8 with stride 4 and a ReLU. The second hidden layer convolves 64 filters of 4×4 with stride 2, followed by a ReLU. This is followed by another convolutional layer that convolves 64 filters of 3×3 with stride 1 followed by a rectifier. The final hidden layer is fully connected and consists of 512 ReLUs and the output layer is a fully-connected linear layer with a single output for each action (Mnih et al. 2015).

ciated objects are visited. Thus, at each time step the agent is free to select any object as its intrinsic goal in the hope to find the optimal extrinsic rewarding task sequence. The total reward is composed of the extrinsic non-Markovian reward and an intrinsic automaton-based reward

$$r^T = \widehat{r}^{\,e} + \mu\, r^i, \tag{1}$$

where $\mu > 0$ is a positive coefficient. The underlying mechanism of intrinsic reward depends on the inferred automaton and is explained in detail later. The only extrinsic rewards in Montezuma's Revenge example are the reward of reaching the key $\widehat{r}^{\,e}_{key}$ and reaching one of the doors $\widehat{r}^{\,e}_{door}$. A schematic of the DeepSynth framework is provided in Fig. 2 and the algorithm is described step-by-step in the following sections. The pseudo-code for the algorithm is given in the Appendix.

Tracing (Step 1 in Fig. 2)

In Montezuma's Revenge, states consist of raw pixel images. Each state is a stack of four consecutive frames $84 \times 84 \times 4$ that are preprocessed to reduce input dimensionality (Mnih et al. 2015). The labelling function wields the object vocabulary Σ to detect object pixel overlap in a particular state frame. For example, if the pixels of `red_character` collide with the pixels of `rope` in any of the stacked frames, the labelling function for that particular state s is $L(s)$={`red_character`, `rope`} (Fig. 1.b). In this specific example since we know the only moving object is the character, for sake of succinctness in the paper, we only give the second part of the labelling function to specify to an event, e.g., the above label would be $L(s)$={`rope`}.

Given this labelling function, tracing records the sequence of detected objects $L(s_i)L(s_{i+1})...$ as the agent explores the MDP. Note that the labelling function is a mapping from the state space to the power set of objects in the vocabulary $L : \mathcal{S} \rightarrow 2^\Sigma$ and thus, the label of a state could be the empty set or a set of several objects.

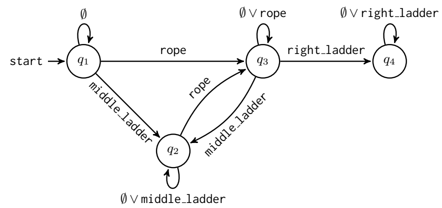
All transitions with the corresponding label are stored as 5-tuples $\langle s, a, s', r^T(s,a), L(s') \rangle$ where s is the current state, a is the executed action, s' is the resulting state, $r^T(s,a)$ is the total reward received after performing action a at state s, and $L(s')$ is the label corresponding to the set of objects in Σ that hold in state s'. The set of past experiences is called the experience replay buffer $\mathcal{E}$. The exploration process generates a set of *traces*, defined as follows:

Definition 5.1 (Trace) *In a general MDP $\mathfrak{M}$, and over a finite path $\rho_n = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ... \xrightarrow{a_{n-1}} s_n$, a trace σ is defined as a sequence of labels $\sigma = \{v_i\}_{i=1}^n$ where $v_i = L(s_i)$ is a trace event.*

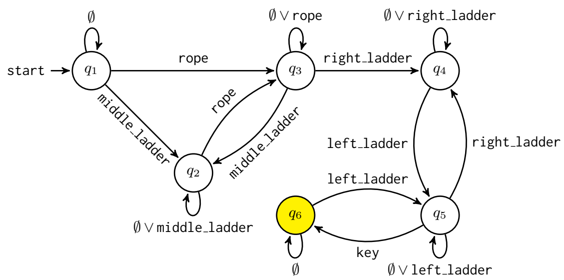
The set of traces associated with $\mathcal{E}$ is denoted by $\mathcal{T}$. The tracing framework is represented by the "Tracing" box in Fig. 2.

Synthesis (Step 2 in Fig. 2)

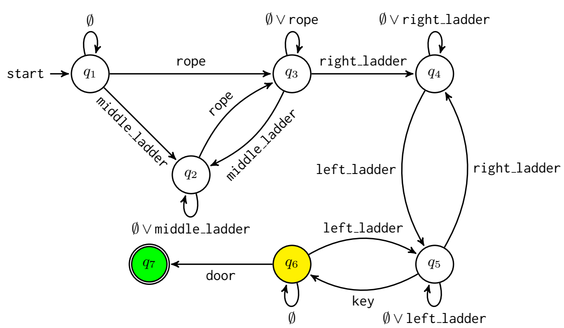
The synthesis framework described in Section 4 is used to generate an automaton that conforms to the trace sequences generated by the Tracing step. Given a trace sequence $\sigma = \{v_i\}_{i=1}^n$, the labels v_i serve as transition predicates in the generated automaton. The synthesis framework further constrains automaton construction such that no two



(a) The right ladder is often discovered by random exploration



(b) The key is found and an extrinsic reward of $\widehat{r}^{\,e}_{key} = +100$ is received



(c) The door is unlocked and an extrinsic reward of $\widehat{r}^{\,e}_{door} = +300$ is received

Figure 4: Step samples of the evolution of the automaton synthesised for Montezuma's Revenge example. Interestingly the agent found a short-cut to reach the key by skipping the middle-ladder and directly jumping over the rope, which is not even obvious to a human player. Without a human-interpretable automaton such observations are hard, if at all possible, to extract from other forms of hierarchy representations, e.g. LSTMs.

transitions from a given state in the generated automaton have the same predicates. The automaton obtained by the synthesis framework is thus deterministic. The learned automaton follows the standard definition of a Deterministic Finite Automaton (DFA) with the alphabet $\Sigma_{\mathfrak{A}}$ where a symbol of the alphabet $v \in \Sigma_{\mathfrak{A}}$ is given in the following by the labelling function $L : \mathcal{S} \to 2^{\Sigma}$ defined earlier. Thus, given a trace sequence $\sigma = \{v_i\}_{i=1}^{n}$ over a finite path $\rho_n = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ... \xrightarrow{a_{n-1}} s_n$ in the MDP, the symbol $v_i \in \Sigma_{\mathfrak{A}}$ is given by $v_i = L(s_i)$.

The Atari emulator provides the number of lives left in the game, which is used to mark the reset of $\Sigma_{\mathfrak{a}} \subseteq \Sigma_{\mathfrak{A}}$ where $\Sigma_{\mathfrak{a}}$ is the set of labels that the agent observed so far over a single life time. Upon losing a life, $\Sigma_{\mathfrak{a}}$ is reset to the empty set.

Definition 5.2 (Deterministic Finite Automaton) *A DFA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma_{\mathfrak{A}}, F, \delta)$ is a state machine, where $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, $\Sigma_{\mathfrak{A}}$ is the alphabet, $F \subset \mathcal{Q}$ is the set of accepting states, and $\delta : \mathcal{Q} \times \Sigma_{\mathfrak{A}} \to \mathcal{Q}$ is a transition function.*

Let $\Sigma_{\mathfrak{A}}^*$ be the set of all finite words over $\Sigma_{\mathfrak{A}}$. A finite word $w = v_1, v_2, ...v_m \in \Sigma_{\mathfrak{A}}^*$ is accepted by a DFA $\mathfrak{A}$ if there exists a finite run $\theta \in \mathcal{Q}^*$ starting from $\theta_0 = q_0$ where $\theta_{i+1} = \delta(\theta_i, v_{i+1})$ for $i \geq 0$ and $\theta_m \in F$. Given the set of collected traces $\mathcal{T}$ we construct a DFA using the approach described in Section 4: starting from $N = 2$, we systematically search for an automaton that conforms to the trace sequence. This ensures that we generate the smallest automaton that conforms to the input trace. The algorithm first divides the trace into segments using a sliding window of size equal to the hyper-parameter w introduced earlier. The hyper-parameter w determines the input size, and consequently the algorithm runtime. Choosing $w = 1$ will not capture any sequential behaviour, but will only ensure that all trace events appear in the automaton. For the automata synthesis in our setting, we would like to choose a value for w that results in the smallest input size but is not trivial $(w = 1)$.

Once a candidate automaton is generated, the automata synthesis framework performs a compliance check by verifying if all transition sequences in the automaton of a given length, equal to the hyper-parameter l, are subsequences of the trace. A higher value for l implies tighter constraints on the automaton, moving towards a more exact representation. Learning exact automata from trace data is known to be NP-complete (Gold 1978). For our experiments we incrementally tried different values for w, ranging within $1 < w \leq |\sigma|$, and have obtained the same automaton in all scenarios. We optimise over the hyper-parameters and choose $w = 3$ and $l = 2$ as the best fit for our setting. This ensures that the automata synthesis problem is not too complex for the synthesis framework to solve but at the same time it does not over-generalise to fit the trace.

The obtained automaton provides deep insight into the correlation of detected objects in Step 1 and shapes the intrinsic reward. The output of this stage is a DFA, from the set of succinct DFAs obtained earlier. Fig. 4 shows three step samples of the evolution of the synthesised automaton for the segmented objects in Montezuma's Revenge. Most of the deep RL approaches are able to reach to the stage of the DFA in Fig. 4.a via random exploration. However, reaching the key and further the doors as in Fig. 4.b and Fig. 4.c is challenging and is achieved by a hierarchical curiosity-driven learning method described in the next section. The automata synthesis phase is represented by the "Synth" box in Fig. 2.

Deep Temporal Neural Fitted RL (Step 3 in Fig. 2)

We propose a deep RL scheme inspired by DQN (Mnih et al. 2015) (and Neural Fitted Q-iteration (NFQ) (Riedmiller 2005) when the state is not an RGB image and already is in vector form). We show that the proposed scheme (1) is able to synthesise a policy whose traces are accepted by the DFA,

(2) encourages the agent to explore under the DFA guidance, and (3) more importantly to expand the DFA towards task satisfaction.

Given the constructed DFA at each time step during the exploration, if a new label is observed during exploration the intrinsic reward in (1) becomes positive. Namely,

$$R^i(s,a) = \left\{ \begin{array}{ll} \eta & \text{if } L(s') \notin \Sigma_{\mathfrak{a}}, \\ 0 & \text{otherwise,} \end{array} \right. \tag{2}$$

where η is an arbitrarily finite and positive reward, and $\Sigma_{\mathfrak{a}}$, as discussed in the Synthesis step, is the set of labels that the agent has observed. Further, once a new label that does not belong to $\Sigma_{\mathfrak{A}}$ is observed during exploration (Step 1) it is then passed to the automaton synthesis module (Step 2). The synthesis module then synthesises a new DFA that complies with the new label.

In the following in order to explain the core ideas underpinning the algorithm, we temporarily assume that the MDP graph and the associated transition probabilities are fully known. Later we relax these assumptions, and we stress that the algorithm can be run *model-free* over any black-box MDP environment. We relate the black-box MDP and the automaton by synchronizing them *on-the-fly* (Remark 5.1) to create a new structure that breaks down a non-Markovian task into a set of Markovian, history-independent sub-goals.

Definition 5.3 (Product MDP) *Given an MDP* $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \Sigma)$ *and a DFA* $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma_{\mathfrak{A}}, F, \delta)$*, the product MDP is defined as* $(\mathfrak{M} \otimes \mathfrak{A}) = \mathfrak{M}_{\mathfrak{A}} = (\mathcal{S}^{\otimes}, \mathcal{A}, s_0^{\otimes}, P^{\otimes}, \Sigma^{\otimes}, F^{\otimes})$*, where* $\mathcal{S}^{\otimes} = \mathcal{S} \times \mathcal{Q}$*,* $s_0^{\otimes} = (s_0, q_0)$*,* $\Sigma^{\otimes} = \mathcal{Q}$*, and* $F^{\otimes} = \mathcal{S} \times F$*. 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 given by* $s_j \sim P(\cdot|s_i, a)$ *and* $q_j = \delta(q_i, L(s_j))$*.*

By synchronising MDP states with the DFA states through the product MDP we can evaluate the satisfaction of the associated high-level task. Most importantly, as shown in (Brafman, De Giacomo, and Patrizi 2018), for any MDP $\mathfrak{M}$ with finite-horizon non-Markovian reward (e.g. Montezuma's Revenge example), there exists a Markov reward MDP $\mathfrak{M}' = (\mathcal{S}, \mathcal{A}, s_0, P, \Sigma)$ that is equivalent to $\mathfrak{M}$ such that the states of $\mathfrak{M}$ can be mapped into those of $\mathfrak{M}'$. The corresponding states yield the same transition probabilities, and also corresponding traces have same rewards. Based on this result, (De Giacomo et al. 2019) showed that the product MDP $\mathfrak{M}_{\mathfrak{A}}$ is indeed $\mathfrak{M}'$ defined above. Therefore, the non-Markovianity of the extrinsic reward is resolved by synchronising the DFA with the original MDP, where the DFA represents the history of state labels that has led to that reward.

Remark 5.1 *Note that the DFA transitions can be executed just by observing the labels of the visited states, which makes the agent aware of the automaton state without explicitly constructing the product MDP. This means that the proposed approach can run* model-free*, and as such it does not require a priori knowledge about the MDP.*

Each state of the DFA in the synchronised product MDP divides the general sequential task so that each transition between the states represents an achievable Markovian subtask. Thus, given a synthesised DFA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma_{\mathfrak{A}}, F, \delta)$, we propose a hybrid architecture of $n = |\mathcal{Q}|$ separate deep RL modules (Fig. 3 and the "Deep" box in Fig. 2). Each deep RL module is associated with a state in the DFA and together the interconnected modules act as a global *hybrid* deep RL architecture to approximate the Q-function in the product MDP. This allows the agent to jump from one subtask to another by just switching between these modules where transitions are prescribed by the DFA.

In the running example, the agent exploration scheme is ϵ-greedy with diminishing ϵ where the rate of decrease also depends on the DFA state so that each module has a chance to explore. Within the product MDP, for each automaton state $q_i \in \mathcal{Q}$ the associated deep RL module is called $B_{q_i}(s,a)$. Namely, once the agent is at state $s^{\otimes} = (s, q_i)$ the neural net B_{q_i} is active and explores the MDP. Note that the modules are not decoupled. For example, assume that by taking action a in state $s^{\otimes} = (s, q_i)$ the label $v = L(s')$ has been observed and as a result the agent is moved to state $s^{\otimes\prime} = (s', q_j)$ where $q_i \neq q_j$. By minimising the loss function $\mathfrak{L}$ the weights of B_{q_i} are updated such that $B_{q_i}(s,a)$ has minimum possible error to $R^T(s,a) + \gamma \max_{a'} B_{q_j}(s', a')$ while $B_{q_i} \neq B_{q_j}$. As such, the output of B_{q_j} directly affects B_{q_i} when the automaton state is changed. This allows the extrinsic reward to back-propagate efficiently, e.g. from modules B_{q_7} and B_{q_6} associated with q_7 and q_6 in Fig. 4.c, to the initial state.

Define $\mathcal{E}_{q_i}$ as the projection of $\mathcal{E}$ onto q_i. The size of replay buffer for each module is limited and in the case of running example $|\mathcal{E}_{q_i}| = 15000$ and it includes most recent frames that are observed during the time when the product MDP state was $s^{\otimes} = (s, q_i)$. In the running example we used RMSProp for each module with uniformly samples mini-batches of size 32. When the state is in vector form and no convolutional layer is involved we resort to NFQ deep RL modules instead of DQN modules for sample efficiency. All hyper parameters are listed in the Appendix.

6 Experimental Results

Benchmarks and Setup

We evaluate the performance of our framework on a a comprehensive set of benchmarks, given in Table 1. The Minecraft environment (`minecraft-tX`) taken from (Andreas, Klein, and Levine 2017) involves various kinds of challenging low-level control tasks, and related joint high-level goals. The two `mars-rover` benchmarks are taken from (Hasanbeig, Abate, and Kroening 2019), where the models have uncountably infinite (continuous) state spaces. Example `robot-surve` is adopted from (Sadigh et al. 2014) where the task is to visit two regions sequentially while avoiding an unsafe area. Models `slp-easy` and `slp-hard` are inspired by noisy MDPs of Chapter 6 in (Sutton and Barto 1998). The goal in `slp-easy` is to reach a particular region of the MDP and the goal in `slp-hard` is to visit four distinct regions sequentially in proper order. Both logics are also used in the `frozen-lake` benchmarks, where the first three are simple reachability and the last three are sequential visits of four regions, except that now there exist unsafe regions as well. The `frozen-lake` MDPs are stochastic and are adopted from OpenAI Gym (Brockman et al. 2016). The *task DFA* column in Table 1

Table 1: Comparison between DeepSynth and deep reinforcement learning (DQN)

experiment	$\lvert\mathbb{S}\rvert$	task DFA	synth DFA	prod. MDP	max sat. prob. at s_0	DeepSynth conv. ep.*	DQN conv. ep.*
minecraft-t1	100	3	6	600	1	25	40
minecraft-t2	100	3	6	600	1	30	45
minecraft-t3	100	5	5	500	1	40	t/o
minecraft-t4	100	3	3	300	1	30	50
minecraft-t5	100	3	6	600	1	20	35
minecraft-t6	100	4	5	500	1	40	t/o
minecraft-t7	100	5	7	800	1	70	t/o
mars-rover-1	∞	3	3	∞	n/a	40	50
mars-rover-2	∞	4	4	∞	n/a	40	t/o
robot-surve	25	3	3	75	1	10	10
slp-easy-sml	120	2	2	240	1	10	10
slp-easy-med	400	2	2	800	1	20	20
slp-easy-lrg	1600	2	2	3200	1	30	30
slp-hard-sml	120	5	5	720	1	80	t/o
slp-hard-med	400	5	5	2400	1	100	t/o
slp-hard-lrg	1600	5	5	9600	1	120	t/o
frozen-lake-1	120	3	3	360	0.9983	100	120
frozen-lake-2	400	3	3	1200	0.9982	150	150
frozen-lake-3	1600	3	3	4800	0.9720	150	150
frozen-lake-4	120	6	6	840	0.9728	300	t/o
frozen-lake-5	400	6	6	2800	0.9722	400	t/o
frozen-lake-6	1600	6	6	11200	0.9467	450	t/o

* average number of episodes to convergence over 10 runs

presents number of states in the automaton that can be generated from the high-level objective sequences of the ground-truth task. The *synth DFA* gives number of states of the synthesised automaton in DeepSynth, and *prod. MDP* shows the number of states in the resulting product MDP as per Definition 5.3. Finally, *max sat. prob. at* s_0 is the maximum probability of achieving the extrinsic reward from the initial state.

Recall that in all experiments the high-level objective sequences are initially unknown to the agent and the agent has to infer them as a DFA. Furthermore, the only source of extrinsic reward is completing the task and reaching the objectives in correct order. Details of the experiments are presented in the Appendix.

Results

The training progress for Montezuma's Revenge and Task 3 in Minecraft is illustrated in Fig. 5 and Fig. 6. In Fig. 6-left the orange line shows the very first deep net associated to the initial state of the DFA, the red and blue ones are of the intermediate states in the DFA and the green line is associated to the final state. This shows an efficient back-propagation of extrinsic reward from the final high-level state to the initial state, namely once the last deep net converges the expected reward is back-propagated to the second and so on. Each NFQ module includes 2 hidden layers and 128 ReLUs.

Note that there may be a number of ways to accomplish a particular task in the synthesised DFAs. This phenomenon however causes no harm to the learning since there is only one valid way to receive the extrinsic reward. Hence, once the extrinsic reward is back-propagated the non-optimal options automatically fall out. Further, we would like to emphasise that if the task DFA, in any of the experiments, is known or even partially known a priori, then the Synth step can build on top of this partial automaton by incrementally adding any new labels or subtask sequences discovered during exploration. As an example, if the automaton in Fig. 4.b was present initially, the agent was able to utilise the semantic correlation of objects to facilitate its explorations and find the key faster.

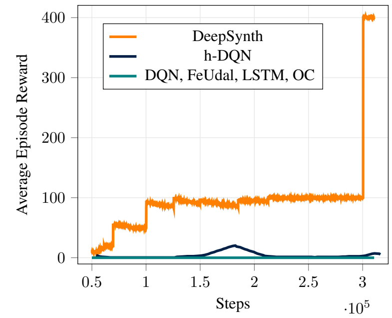


Figure 5: Average episode reward progress in Montezuma's Revenge with h-DQN (Kulkarni et al. 2016), DQN (Mnih et al. 2015), FeUdal-LSTM (Vezhnevets et al. 2017), and Option-Critic (OC) (Bacon, Harb, and Precup 2017).The reward of reaching the key is 100 and reaching the door is another 300. h-DQN (Kulkarni et al. 2016) finds the door but only after 2M steps. FeUdal and LSTM also find the door after 100M and 200M steps respectively. DQN and OC remain flat. DeepSynth finds the door in about 300k steps.

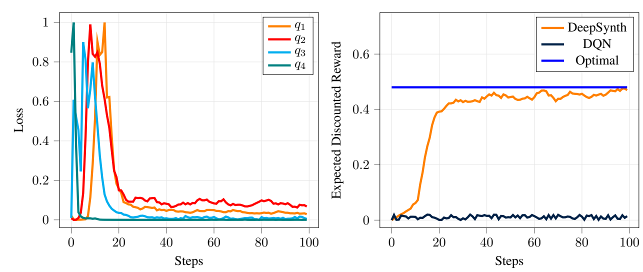


Figure 6: Minecraft Task 3 Experiment - Left: Training progress with four hybrid deep NFQ module, Right: Training progress with DeepSynth and DQN on the same training set

7 Conclusions

We have proposed a fully-unsupervised approach for training deep RL agents when the reward is extremely sparse and non-Markovian. We *automatically* infer this high-level structure from observed exploration traces using techniques from automata synthesis. The inferred automaton is a formal, ungrounded, human-interpretable representation of a complex task and its components. Thanks to the modular structure of the automaton, the overall task can be segmented into easy Markovian sub-tasks. Therefore, any segment of the proposed network that is associated with a sub-task can be used as a separate trained module in transfer learning scenarios. Another major contribution of the proposed framework is that in problems where human domain knowledge is available, it can be easily encoded as an automaton to guide learning. This enables the agent to solve complex tasks and saves the agent from an exhaustive exploration in the beginning. We showed that we are able to learn optimal policies that achieve complex high-level objectives using fewer training samples as compared to the other algorithms.

References

Abbeel, P.; Coates, A.; Quigley, M.; and Ng, A. Y. 2007. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems*, 1–8. MIT Press.

Andreas, J.; Klein, D.; and Levine, S. 2017. Modular Multitask Reinforcement Learning with Policy Sketches. In *ICML*, volume 70, 166–175.

Angluin, D. 1987. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.* 75(2): 87–106. ISSN 0890-5401. doi:10.1016/0890-5401(87)90052-6.

Bacon, P.-L.; Harb, J.; and Precup, D. 2017. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47: 253–279.

Berlyne, D. E. 1960. Conflict, arousal, and curiosity. .

Bertsekas, D. P.; and Shreve, S. 2004. *Stochastic optimal control: the discrete-time case*. Athena Scientific.

Bertsekas, D. P.; and Tsitsiklis, J. N. 1996. *Neuro-dynamic Programming*, volume 1. Athena Scientific.

Biermann, A. W.; and Feldman, J. A. 1972. On the Synthesis of Finite-State Machines from Samples of Their Behavior. *IEEE Trans. Comput.* 21(6): 592–597. ISSN 0018-9340. doi:10.1109/TC.1972.5009015.

Brafman, R. I.; De Giacomo, G.; and Patrizi, F. 2018. LTLf/LDLf Non-Markovian Rewards. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540* .

Buzhinsky, I.; and Vyatkin, V. 2017. Automatic Inference of Finite-State Plant Models From Traces and Temporal Properties. *IEEE Trans. Ind. Informat.* 13(4): 1521–1530. ISSN 1551-3203. doi:10.1109/TII.2017.2670146.

Buzhinsky, I.; and Vyatkin, V. 2017. Modular plant model synthesis from behavior traces and temporal properties. *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)* 1–7.

Cook, S.; and Mitchell, D. 1996. Finding hard instances of the satisfiability problem: A survey. In *Satisfiability Problem: Theory and Applications*.

Csikszentmihalyi, M.; and Csikzentmihaly, M. 1990. *Flow: The psychology of optimal experience*, volume 1990. Harper & Row New York.

Daniel, C.; Neumann, G.; and Peters, J. 2012. Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, 273–281.

Davis, M.; and Putnam, H. 1960. A Computing Procedure for Quantification Theory. *J. ACM* 7(3): 201–215. ISSN 0004-5411. doi:10.1145/321033.321034. URL https://doi.org/10.1145/321033.321034.

De Giacomo, G.; Iocchi, L.; Favorito, M.; and Patrizi, F. 2019. Foundations for Restraining Bolts: Reinforcement Learning with LTLf/LDLf Restraining Specifications. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 128–136.

Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K. O.; and Clune, J. 2019. Go-Explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995* .

Fu, J.; and Topcu, U. 2014. Probably Approximately Correct MDP Learning and Control With Temporal Logic Constraints. In *Robotics: Science and Systems X*.

Glover, F.; and Laguna, M. 1998. *Tabu Search*, 2093–2229. Boston, MA: Springer US. ISBN 978-1-4613-0303-9. doi:10.1007/978-1-4613-0303-9_33. URL https://doi.org/10.1007/978-1-4613-0303-9_33.

Gold, E. M. 1978. Complexity of Automaton Identification from Given Data. *Information and Control* 37: 302–320.

Gulwani, S. 2012. Synthesis from Examples. In *3rd Workshop on Advances in Model-Based Software Engineering (WAMBSE)*.

Hahn, E. M.; Perez, M.; Schewe, S.; Somenzi, F.; Trivedi, A.; and Wojtczak, D. 2019. Omega-Regular Objectives in Model-Free Reinforcement Learning. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 395–412. Springer.

Hasanbeig, M.; Abate, A.; and Kroening, D. 2018. Logically-Constrained Reinforcement Learning. *arXiv preprint arXiv:1801.08099* .

Hasanbeig, M.; Abate, A.; and Kroening, D. 2019. Logically-Constrained Neural Fitted Q-Iteration. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2012–2014. International Foundation for Autonomous Agents and Multiagent Systems.

Heule, M. J. H.; and Verwer, S. 2013. Software Model Synthesis Using Satisfiability Solvers. *Empirical Software Engineering* 18(4): 825–856. doi:10.1007/s10664-012-9222-z.

Hwang, J.-J.; Yu, S. X.; Shi, J.; Collins, M. D.; Yang, T.-J.; Zhang, X.; and Chen, L.-C. 2019. SegSort: Segmentation by Discriminative Sorting of Segments. In *Proceedings of the IEEE International Conference on Computer Vision*, 7334–7344.

Icarte, R. T.; Waldie, E.; Klassen, T.; Valenzano, R.; Castro, M.; and McIlraith, S. 2019. Learning Reward Machines for Partially Observable Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 15497–15508.

Jeppu, N. Y.; Melham, T.; Kroening, D.; and O'Leary, J. 2020. Learning Concise Models from Long Execution Traces. In *Design Automation Conference*.

Ji, X.; Henriques, J. F.; and Vedaldi, A. 2019. Invariant information clustering for unsupervised image classification and segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, 9865–9874.

Kazemi, M.; and Soudjani, S. 2020. Formal Policy Synthesis for Continuous-Space Systems via Reinforcement Learning. *arXiv preprint arXiv:2005.01319* .

Kearns, M.; and Singh, S. 2002. Near-optimal reinforcement learning in polynomial time. *Machine learning* 49(2-3): 209–232.

Kingma, D. P.; and Ba, J. 2015. Adam: A method for stochastic optimization. *ICLR* .

Koul, A.; Greydanus, S.; and Fern, A. 2018. Learning finite state representations of recurrent policy networks. *arXiv preprint arXiv:1811.12530* .

Krishnan, S. C.; Puri, A.; Brayton, R. K.; and Varaiya, P. P. 1995. The Rabin Index and Chain Automata, with Applications to Automata and Games. In Wolper, P., ed., *Computer Aided Verification*, 253–266. Springer. ISBN 978-3-540-49413-3.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, 3675–3683.

Lang, K. J.; Pearlmutter, B. A.; and Price, R. A. 1998. Results of the Abbadingo One DFA Learning Competition and a new Evidence-driven State Merging Algorithm. In Honavar, V.; and Slutzki, G., eds., *Grammatical Inference*, 1–12. Springer. ISBN 978-3-540-68707-8.

Mao, H.; Alizadeh, M.; Menache, I.; and Kandula, S. 2016. Resource management with deep reinforcement learning. In *ACM Workshop on Networks*, 50–56. ACM.

Memarian, F.; Xu, Z.; Wu, B.; Wen, M.; and Topcu, U. 2020. Active Task-Inference-Guided Deep Inverse Reinforcement Learning. *arXiv preprint arXiv:2001.09227* .

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level Control Through Deep Reinforcement Learning. *Nature* 518(7540): 529–533.

Precup, D. 2001. *Temporal abstraction in reinforcement learning*. Ph.D. thesis, University of Massachusetts Amherst.

Riedmiller, M. 2005. Neural Fitted Q iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method. In *ECML*, volume 3720, 317–328. Springer.

Ryan, R. M.; and Deci, E. L. 2000. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology* 25(1): 54–67.

Sadigh, D.; Kim, E. S.; Coogan, S.; Sastry, S. S.; and Seshia, S. A. 2014. A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In *CDC*, 1091–1096. IEEE.

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

Sutton, R. S.; and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1-2): 181–211.

Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2018. Teaching Multiple Tasks to an RL Agent using LTL. In *AAMAS*, 452–461.

Ulyantsev, V.; Buzhinsky, I.; and Shalyto, A. 2016. Exact Finite-State Machine Identification from Scenarios and Temporal Properties. *CoRR* abs/1601.06945.

Ulyantsev, V.; and Tsarev, F. 2011. Extended Finite-State Machine Induction Using SAT-Solver. In *International Conference on Machine Learning and Applications and Workshops*, 346–349. doi:10.1109/ICMLA.2011.166.

Vezhnevets, A.; Mnih, V.; Osindero, S.; Graves, A.; Vinyals, O.; Agapiou, J.; et al. 2016. Strategic attentive writer for learning macro-actions. In *Advances in neural information processing systems*, 3486–3494.

Vezhnevets, A. S.; Osindero, S.; Schaul, T.; Heess, N.; Jaderberg, M.; Silver, D.; and Kavukcuoglu, K. 2017. FeUdal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161* .

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 1–5.

Walkinshaw, N. 2018. *MINT framework Github repository*. URL https://github.com/neilwalkinshaw/mintframework.

Walkinshaw, N.; and Bogdanov, K. 2008. Inferring Finite-State Models with Temporal Constraints. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, 248–257. ISSN 1938-4300. doi:10.1109/ASE.2008.35.

Walkinshaw, N.; Bogdanov, K.; Holcombe, M.; and Salahuddin, S. 2007. Reverse Engineering State Machines by Interactive Grammar Inference. In *Proceedings of the 14th Working Conference on Reverse Engineering*, WCRE '07, 209–218. IEEE Computer Society. ISBN 0-7695-3034-6. doi:10.1109/WCRE.2007.45.

Walkinshaw, N.; Taylor, R.; and Derrick, J. 2016. Inferring extended finite state machine models from software executions. *Empirical Software Engineering* 21(3): 811–853. ISSN 1573-7616.

Watkins, C. J.; and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4): 279–292.

Zheng, Z.; and Yang, Y. 2020. Rectifying Pseudo Label Learning via Uncertainty Estimation for Domain Adaptive Semantic Segmentation. *arXiv preprint arXiv:2003.03773* .

Zhou, Z.; Li, X.; and Zare, R. N. 2017. Optimizing chemical reactions with deep reinforcement learning. *ACS Central Science* 3(12): 1337–1344.