University of California, Merced

Ph.D. Proposal

State Representation in Reinforcement Learning

Jacob Rafati

Electrical Engineering and Computer Science
Computational Cognitive Neuraocience Lab
University of California, Merced

jrafatiheravi@ucmerced.edu

supervised by
Dr. David C. Noelle

December 22, 2017

Contents

Abstract

Keywords

1 Introduction

2 Preliminaries

Most of this section is taken from "introduction to reinforcement learning" book by Sutton and Barto (1998)

2.1 Notation

In entire this document, the random variables and matrices are denote by capital letter like X and the vectors and value that random variables can take are denoted by lowercase letter like x. Pr and p is reserved for probability distribution function. For a sequence of variables, $\{X_i, i = 0, \ldots, t\}$ we use the notation of $X_{0:t}$.

2.2 Reinforcement learning problem

Agent and Environement

The reinforcement learning problem is a framework of the problem of learning with interaction with *environment* to achieve a *goal*. The learner and decision maker is called the *agent* and everything outside the agent is called the *environment*. The agent and environment interact at each of a sequenceof discrete time steps, $t = 0, 1, 2, \ldots$ $(t \in [T],\ T \in \mathbb{N})$.

Experience: State, Action, Reward, Next State

At each time step t, the agent receives a representation of the environment's *state*, $S_t \in \mathcal{S}$, where $\mathcal{S}$ is set of all possible states, and on that basis agent selects an *action*, $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}(S_t)$ is a set of all available actions for agent in state S_t. One time step later at $t + 1$, as a consequence of agent's action, the agent receives a *reward* $R_{t+1} \in \mathbb{R}$ and also an update on the new state S_{t+1} from environment. Each cycle of interaction is called an *experience*, $E_{t+1} = \{S_t, A_t, R_{t+1}, S_{t+1}\}$. Figure 1 summerizes the agent–environment interaction. Ideally state should be equivalent to the *state of dynamical sysytem*, but here we refer to state as whatever information is available to the agent.

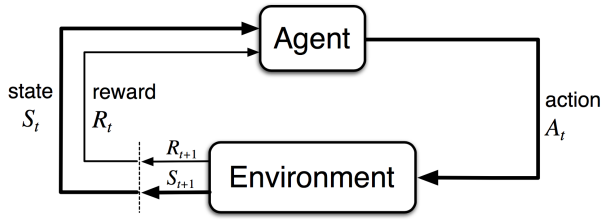

Figure 1: The agent–environment interaction in reinforcement learning.

Policy Function

At each time step agent implements a mapping from states to possible actions $\pi_t : \mathcal{S} \to \mathcal{A}$. This mapping is called agent's *policy*. The stochastic policy can be defined as the probability of taking action $A_t = a$ at state $S_t = s$

$$\pi_t(a|s) = p(A_t = a|S_t = s).$$

The deterministic policy can be obtained as

$$a = \pi_t(s) = \arg\max_{a'} \ \pi_t(a'|s)$$

Goals

The objective of reinforcement learning problem is the maximization of the expected value of *return* i.e. the cumulative sum of a received reward

$$G_t = \sum_{t'=t+1}^{T} \gamma^{t'-t-1} R_{t'}, \quad \forall t \in [T], \tag{1}$$

where $T \in \mathbb{N}$ is a final step and $\gamma \in [0, 1]$ is a discount factor. As $\gamma \to 1$, the objective takes future rewards into account more strongly (farsighted agent) and if $\gamma \to 0$ the agent is only concerend about maximizing the immediate reward (myopic agent). Having $\gamma < 1$ can bound the return $G_t < \infty$ when $T \to \infty$. The goal of reinforcement learning problem is finding an optimal policy that maximizes the expected return.

Episodes

The agentenvironment interaction can often break into subsequences, which we call *episodes*, such as plays of a game or any sort of repeated interactions. Each episode ends when time is over i.e. $t = T$ or agent reaches to an absorbing *terminal* state.

Markov Property

A state signal that succeeds in retaining all relevant information is said to be Markov, or to have the Markov property. Consider how a general environment might respond at time $t+1$ to the action taken at time t. In general the dynamics can be defined only by specifying the complete joint probability distribution:

$$\Pr\{S_{t+1} = s', R_{t+1} = r \mid S_{0:t}, A_{0:t}\} \quad \forall s', r.$$

If the state signal has the Markov property, then the environment's response at $t+1$ depends only on the state and action representations at t, in which case the environment's dynamics can be defined by

$$p(s', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}. \tag{2}$$

Markov Decision Processes

A reinforcement learning task that satisfies the Markov property is called a Markov decision process (MDP). If the state and action spaces are finite, then it is called a finite Markov decision process. A finite MDP is defined by its state and action sets and by the one-step *dynamics* of the environment specified by (2). These quantities completely specify the dynamics of a finite MDP. In this proposal, we implicitly assumes the environment is a finite MDP. Given the dynamics of environment, we can compute the *state-transition probabilities*,

$$p(s'|s, a) = \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\} = \sum_r p(s', r|s, a), \tag{3}$$

Also we can compute the expected rewards for stateaction pairs,

$$r(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_r r \sum_{s' \in \mathcal{S}} p(s', r|s, a), \tag{4}$$

and the expected rewards for stateaction-next state triples

$$r(s, a, s') = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_r r p(s', r|s, a)}{p(s'|s, a)}, \tag{5}$$

Value Function

The value of state s under a policy π, denoted $v_\pi(s)$ is the expected return when starting from s and following π thereafter. For MDPs, $v_\pi(s)$ can be defined as

$$\begin{aligned} &v_\pi : \mathcal{S} \to \mathbb{R} \\ &v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi\Big[\sum_{t'=t+1}^{T} \gamma^{t'-t-1} R_{t'} \;\Big|\; S_t = s \Big], \end{aligned} \tag{6}$$

where $\mathbb{E}_\pi[.]$ denotes the expected value of random variable given that the agent follows policy π. Value of terminal state by definition is always zero. We call the function v_π the state-value function for policy π. Similarly, we can define the state-action value function $q_\pi(s, a)$ for policy π as

$$\begin{aligned} &q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R} \\ &q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi\Big[\sum_{t'=t+1}^{T} \gamma^{t'-t-1} R_{t'} \;\Big|\; S_t = s, A_t = a \Big], \end{aligned} \tag{7}$$

Based on these definitions, the two value functions, v_π and q_π are related

$$v_\pi(s) = \sum_a \pi(a|s)\; q_\pi(s, a). \tag{8}$$

Bellman Equations

For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states s':

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma v_\pi(s')\big], \quad \forall s \tag{9}$$

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)\big[r + \gamma v_\pi(s')\big], \quad \forall s, a. \tag{10}$$

We can see that (8) holds for Bellman's equations. We can rewrite Bellman equations using expected reward and state transition probabilities in (3) and (4):

$$v_\pi(s) = \sum_a \pi(a|s)\big[r(s, a) + \gamma \sum_{s'} p(s'|s, a) v_\pi(s')\big], \quad \forall s. \tag{11}$$

$$q_\pi(s, a) = r(s, a) + \gamma \sum_{s'} p(s'|s, a) v_\pi(s'), \quad \forall s, a. \tag{12}$$

Optimal Value Functions

The optimal state-value function, denoted v_* is the objective function in reinforcement learning problem

$$v_* = \max_\pi v_\pi(s), \quad \forall s. \tag{13}$$

There is always at least one policy that is better than or equal to all other policies. This is an optimal policy

$$\pi_* = \arg\max_\pi v_\pi(s), \quad \forall s. \tag{14}$$

Optimal policies also share the same optimal action-value function, denoted q

$$q_* = \max_\pi q_\pi(s, a), \quad \forall s, a. \tag{15}$$

We can easily obtain optimal policies once we have found the optimal value functions, v or q, which satisfy the Bellman optimality equations:

$$v_*(s) = \max_a \left\{ r(s,a) + \gamma \sum_{s'} p(s'|s,a) v(s') \right\}, \quad \forall s. \tag{16}$$

$$q_*(s,a) = \left\{ r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} \{ q_*(s',a') \} \right\}, \quad \forall s, a. \tag{17}$$

Note that $v_*(s) = \max_a q_*(s,a)$. It's easy to prove that equation $\mathbb{T}[v] = v$ with

$$H[v(s)] = \max_a \left\{ r(s,a) + \gamma \sum_{s'} p(s'|s,a) v(s') \right\}, \quad \forall s.$$

has a unique fixed point v_* (function in this functional case) and such function is the optimal value function. Once we have optimal value function v_* or q_*, we can also find the optimal policy map function π_*:

$$\pi_*(s) = \arg\max_a \left\{ r(s,a) + \gamma \sum_{s'} p(s'|s,a) v(s') \right\}, \quad \forall s. \tag{18}$$

$$\pi_*(s) = \arg\max_a q_*(s,a), \quad \forall s. \tag{19}$$

2.3 Policy and Value Iteration Algorithms

Finding the Optimal Value and Policy Functions

There are two main methods in literature to find the optimal value and policy functions.

2.4 SARSA: Online Temporal Difference Method

2.5 Q-Learning

2.6 Generalization in Reinforcement Learning

The Curse of Dimentionality of The State Space

Artficial Neural Netwroks

3 Background and Related Works

4 Sparse Representation of State in Reinforcement Learning

4.1 Introduction

4.2 Background and related works

4.3 Proposed Methods

4.4 Results and Discussion

4.5 Future Works

4.6 Conclusion

5 Subgoal Discovery in Hierarchical Reinforcement Problem

5.1 Introduction

5.2 Background and Related works

5.3 Proposed Methods

5.4 Results and Discussion

5.5 Future Works

5.6 Conclusion

6 Experiments and Applications

6.1 Toy Task: The Rooms Problem

To provide an illustration of HRL in action, we applied the subgoal discovery algorithm to a toy *Rooms problem* introduced by (Sutton and Barto, 1998). Given initial state S and goal state G, the agent's task is to navigate through a set of rooms interconnected by doorways, in order to reach the goal state G (Figure 2). In each state, the agent can select any of eight deterministic primitive actions, each of which moves the agent to one of the adjacent squares (unless a wall prevents this movement). The rooms are reachable through *doors* and agent is only allowed to use these doors to reach the other room. Agent receives reward of -1 for each action, -2 for actions that results bumping to the walls and 0 when reaching to G.

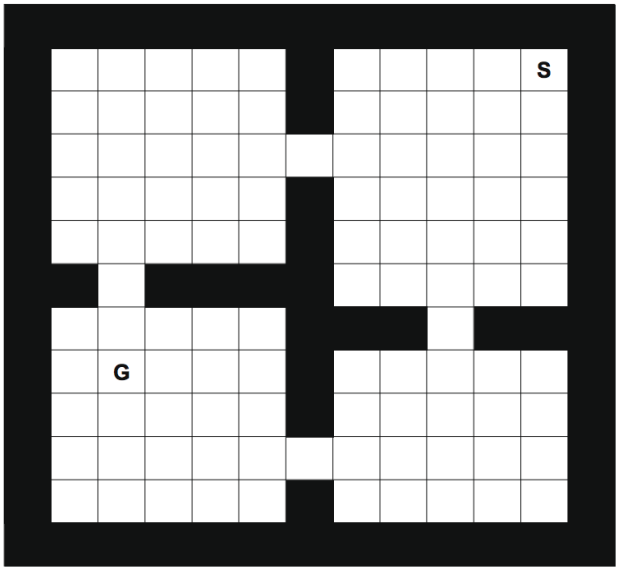

Figure 2: The rooms problem, adapted from (Botvinick et al., 2009)

6.2 Games: Montezuma's Revenge

Montezuma's revenge is one of the hardest games available through the *Arcade Learning Environment* (ALE) (Bellemare et al., 2013). The game is infamous for challenging agents with lethal traps and sparse rewards (Vezhnevets et al., 2017). The game (Figure 3) requires the player to navigate the explorer (in red) through several rooms while collecting treasures. In order to pass through doors (in the top right and top left corners of the figure), the player has to first pick up the key2 (Kulkarni et al., 2016).

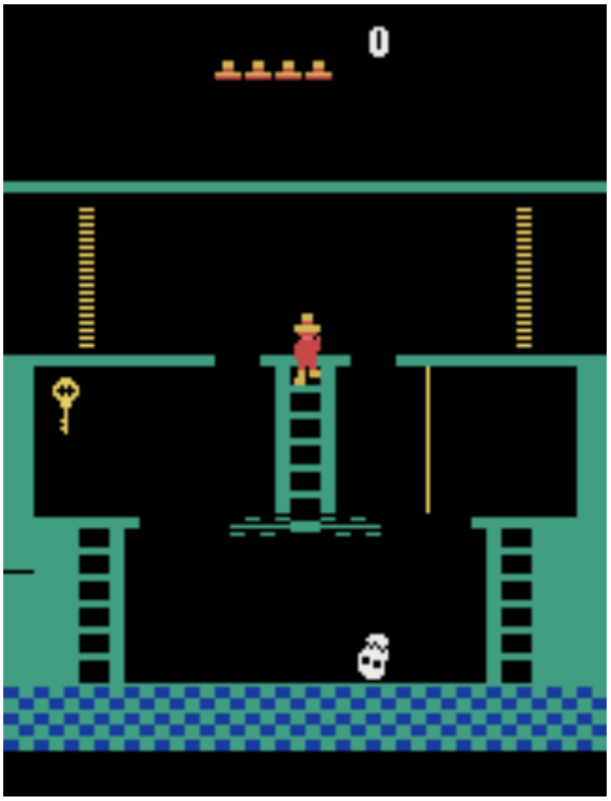

Figure 3: A sample screen from the ATARI 2600 game called ''Montezuma's Revenge" adopted from (Kulkarni et al., 2016)

6.3 Path Planning for Autonoumous Robot Navigation

Navigation is one of the most important robotic tasks. One classic approach consists of three steps: build a map of the environment through, e.g., SLAM, plan a path using the map, and finally execute the path. High-fidelity geometric maps make it easier for path planning and execution. However, building such maps is time-consuming (Gao et al., 2017). Further, even small environmental changes may render them partially invalid. Alternatively, the optical flow approach does not use maps at all and relies on the visual perception of the local environment for navigation. While this avoids the difficulty of building and maintaining accurate geometric maps, it is difficult to achieve effective goal-directed global navigation in complex geometric environments without maps. Our approach uses 2-D floor plans available for indoor environments. We use robotics simulator such as Gebezo and Robot Operating System (ROS) to simulate environments for robotcs navigation tasks.

6.4 Bipedal Locomotion and Reaching Tasks

Learning physics-based locomotion skills is a difficult problem. With this simulation we aim to learn a variety of environment-aware locomotion skills with a limited amount of prior knowledge. We investigate the task of reaching and also locomotion using two-level hierarchical control framework. Results are demonstrated on a simulated 3D biped (Peng et al., 2017; Geijtenbeek et al., 2013). For animating virtual characters, physics based simulation is used by (Geijtenbeek et al., 2013) (Figure 4). The subgoal discovery algorithm can be applied to the experience space of locomotion task and learning to reach for these animated creatures in hierarchical reinforcement learning framework.

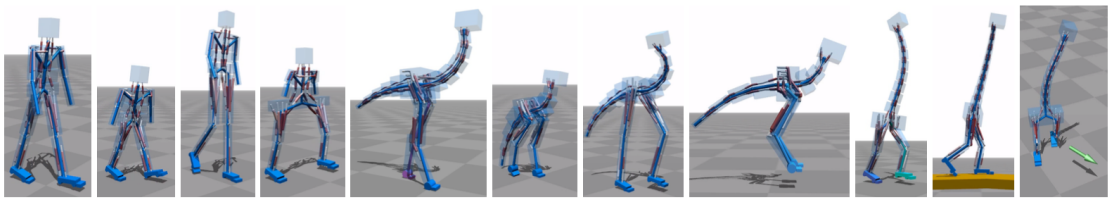

Figure 4: Physics-based simulation of locomotion for a variety of creatures driven by 3D muscle-based control adopted from (Geijtenbeek et al., 2013)

6.5 Autonoumous Driving

We use CARLA, an open-source simulator for autonomous driving research (Dosovitskiy et al., 2017). CARLA has been built for flexibility and realism in the rendering and physics simulation. It is implemented as an open-source layer over Unreal Engine 4 (UE4). CARLA has been developed from the ground up to support development, training, and validation of autonomous urban driving systems. The simulation platform supports flexible specification of sensor suites and environmental conditions. We use CARLA to study the performance of subgoal discovery algorithms for hierarchical reinforcement learning applied to autonomous driving. We use CARLA to stage controlled goal-directed navigation scenarios of increasing difficulty. We manipulate the complexity of the route that must be traversed, the presence of traffic, and the environmental conditions (Figures 5 and 6).

Figure 5: A third-person view in four weather conditions. Clock- wise from top left: clear day, daytime rain, daytime shortly after rain, and clear sunset adopted from (Dosovitskiy et al., 2017)

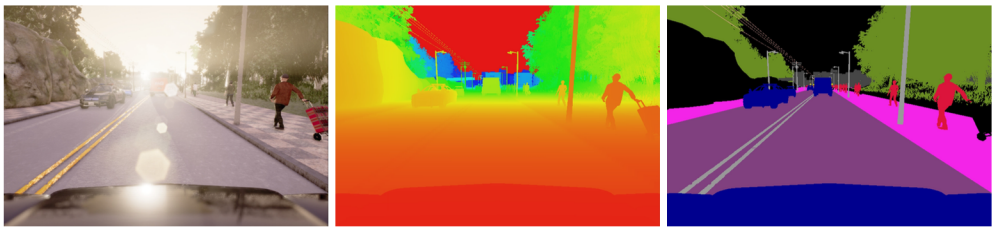

Figure 6: Three of the sensing modalities provided by CARLA. From left to right: normal vision camera, ground-truth depth, and ground-truth semantic segmentation. Depth and semantic segmen- tation are pseudo-sensors that support experiments that control for the role of perception. Additional sensor models can be plugged in via the API adopted from (Dosovitskiy et al., 2017)

7 Research and Writing Timetable

8 Conclusion

References

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.

Botvinick, M. M., Niv, Y., and Barto, A. C. (2009). Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113(3):262 – 280.

Dosovitskiy, A., Ros, G., Codevilla, F., López, A., and Koltun, V. (2017). CARLA: an open urban driving simulator. *CoRR*, abs/1711.03938.

Gao, W., Hsu, D., Lee, W. S., Shen, S., and Subramanian, K. (2017). Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation. *CoRR*, abs/1710.05627.

Geijtenbeek, T., van de Panne, M., and van der Stappen, A. F. (2013). Flexible muscle-based locomotion for bipedal creatures. *ACM Trans. Graph.*, 32(6):206:1–206:11.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. B. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *CoRR*, abs/1604.06057.

Peng, X. B., Berseth, G., Yin, K., and Van De Panne, M. (2017). Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Trans. Graph.*, 36(4):41:1–41:13.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. *CoRR*, abs/1703.01161.