Deep Reinforcement Learning and Self Instructing Database

Debjyoti Paul
University of Utah
deb@cs.utah.edu

INTRODUCTION

Reinforcement learning (RL) is the area of machine learning that deals with *sequential decision making*. Sequential decision-making is a task of deciding, the sequence of actions to perform in an uncertain environment in order to achieve some goals. Sequential decision-making tasks cover a wide range of possible applications with the potential to impact many domains, such as robotics, healthcare, smart grids, finance, self-driving cars, and many more.

Inspired by behavioral psychology (e.g. Sutton [16]) reinforcement learning (RL) proposes a formal framework to this problem. The main idea is that an artificial agent may learn by interacting with its environment, similarly to a biological agent. I will dive more into the details how a general RL problem is formally represented and what are key components involved in the framework in Section 1.

1 PART A

In reinforcement learning there is no supervisor but a *reward* signal to guide the learning process. Even the feddback is delayed and not instantenous which sets it as a different paradigm from other machine learning methods. In a sequential decision making process an/the *agent* get to pick an action at every time step which tries to optimize the future cumulative *reward* signal. Traditional Reinforcement Learning requires explicit design of state space and action space, while the mapping from state space to action space is learned [17]. That makes the problem spaces and the possible states in an environment very limited, only with fully observable state of environment for agent. Neural networks enhances RL with the capability to make the agent learn a state abstraction and a policy approximation directly from its input data in a partially observable environment. Deep RL is the fruitful outcome of this attempt of RL enhancement.

Formal RL Framework:

The general RL problem is formalized as a discrete time stochastic control process where an agent interacts with its environment in the following way: the agent starts, in a given state within its environment $s_0 \in \mathcal{S}$, by gathering an initial observation $\omega_0 \in \Omega$. At each time step t, the agent has to take an action $a_t \in \mathcal{A}$. As illustrated in Figure 1 it follows three consequences: (i) the agent obtains a reward $r_t \in \mathcal{R}$, (ii) the state transitions to $s_{t+1} \in \mathcal{S}$, and (iii) the agent obtains an observation $\omega_{t+1} \in \Omega$. This control setting was first proposed by [3] and later extended to learning by Barto [2]. The goal of RL is to *select actions to maximise total future reward.*

Definition 1. Reward: A reward r_t is a scalar feedback signal that indicates how well *agent* is doing at step t.

University of Utah, Salt Lake City, USA.

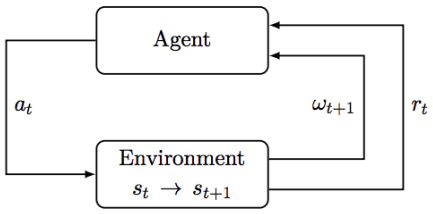

Figure 1: Agent-environment interaction in RL.

An example of *reward* in the case of flying toy robot helicopters are : (a) +ve reward for following desired trajectory, (b) fk??ve reward for crashing.

Definition 2. History: The history $\mathcal{H}_t$ is the sequence of observations, actions, rewards, i.e. all observable variables up to time t. Mathematically, $\mathcal{H}_t = \omega_1, r_1, a_1, \ldots, a_{t\text{fk}??1}, \omega_t, r_t$.

Formally, state $s_t \in \mathcal{S}$ is a function of the history $\mathcal{H}_t$.

$$s_t = f(\mathcal{H}_t)$$

Environment and agent both can have threr own state which we define as s_t^e and s_t^a respectively. The environment state is usually not visible to the agent.

In a fully observable system, the agent directly observes environment state i.e. $\omega_t = s_t^a = s_t^e$. Formally this is called Markov Decision Process (MDP). The Markov property means that the future of the process only depends on the current observation, and the agent has no interest in looking at the full history.

Markov Decision Process (MDP):
An MDP is a 5-tuple $(\mathcal{S}, \mathcal{A}, T, \mathcal{R}, \gamma)$ where

$\mathcal{S}$ is the state space,
$\mathcal{A}$ is the action space,
$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function (set of conditional transition probabilities between states),
R: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$ is the reward function, where R is a continuous set of possible rewards in a range $R_{max} \in \mathbb{R}^+$ (e.g., $[0, R_{max}]$),
$\gamma \in [0, 1)$ is the discount factor.

However, in the real world not many sytems are fully observable, an *agent* indirectly observes the environment without knowing it's internal information state. In this scenario $s_t^a \neq s_t^e$.

Formally this is a partially observable Markov decision process (POMDP).

Partially Observable Markov Decision Process (POMDP): A POMDP is a 7-tuple $(\mathcal{S}, \mathcal{A}, T, \mathcal{R}, \Omega, O, \gamma)$ where

$\mathcal{S}$ is the state space,
$\mathcal{A}$ is the action space,
$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function (set of conditional transition probabilities between states),

R: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$ is the reward function, where R is a continuous set of possible rewards in a range $R_{max} \in \mathbb{R}^+$ (e.g., $[0, R_{max}]$),
Ω is a finite set of observations $1, ..., N_\Omega$,
$O : \mathcal{S} \times \Omega \rightarrow [0, 1]$ is a set of conditional observation probabilities,
$\gamma \in [0, 1)$ is the discount factor.

In a POMDP *agent* must construct its own state representation s_t^a. For example,
(i) **Complete history:** $s_t^a = \mathcal{H}_t$.
(ii) ***Beliefs* of environment state:** $s_t^a = \mathbb{P}[s_t^e = s^1], \ldots, \mathbb{P}[s_t^e = s^n]$ where s^k represents a state in environment.
(iii) **Recurrent Neural Network (RNN):** $s_t^a = \sigma(s_{t-1}^a W_s + \omega_t W_o)$ where W_s and W_o represents weight vectors and σ some non linear function. [10, 11, 18].

This RNN approach to solving POMDPs is related to other problems using dynamical systems and state space models, where the true state can only be estimated [4].

RL Agent Components:

So far, we have introduced the key formalism used in RL, the MDP and the POMDP. Now I will talk about what is inside an RL agent and the components it uses for learning. An RL agent may include one or more of these component for learning.

Value functions: Value function attempts to measure goodness/badness of each state and/or action. In other words, value function methods are based on estimating the value (expected return) of being in a given state i.e. future reward. The state-value function $V_\pi(s)$ is the expected return when starting in state s and following π henceforth:

$$V_\pi(s) = \mathbb{E}_\pi[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots | s_t = s]$$

where $\gamma \in [0, 1]$ is discount parameter, that governs the future sightedness of the model.

Policy Search: A policy is the *agent*'s behavior. It is a map from state to action. Examples of some policies are:
(i) *Deterministic Policy:* This is the simplest policy, here an action a is mapped to a state s i.e. $a = \pi(s)$.
(ii) *Stochastic Policy:* A policy can be stochastic based on probability of state observation, i.e. $\pi(a|s) = \mathbb{P}[\mathcal{A} = a|\mathcal{S} = s]$.
(iii) *Parameterized Policy:* A parameterised policy π_θ is chosen, whose parameters are updated to maximise the expected return $E[R|\theta]$ using either gradient-based or gradient-free optimisation [6]. Neural networks that encode policies have been successfully trained using both gradient-free [5, 8, 13] and gradient based [12, 14, 19] methods.

Model: A model tries to learn the behavior of the environment. First, it tries to predict what the environment will do next, that is predicting next state and is called Transitions ($\mathcal{P}$). Secondly, the model tries to predict the expected next reward and is called Rewards model ($\mathcal{R}$).

$$\mathcal{P}_{ss'}^a = \mathbb{P}[\mathcal{S}' = s'|\mathcal{S} = s, \mathcal{A} = a]$$

where s is the state prior state and s' is the resultant state on action a.

$$\mathcal{R}_s^a = \mathbb{E}[r|\mathcal{S} = s, \mathcal{A} = a]$$

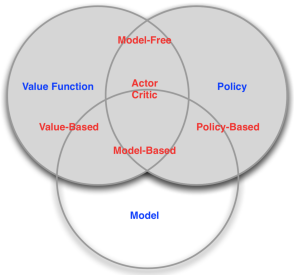

Figure 2: RL Agent Taxonomy.

where s is the state prior state and r is the reward for next step on action a.

RL Agent Taxonomy:

Model methods are optional in RL systems. In fact there are many model free based methods applied for real world problems. To understand a comprehensive taxonomies of categorizing RL Agent we present it in Figure 2.

RL focus on learning without access to the underlying model of the environment. However, interactions with the environment could be used to learn value functions, policies, and also a model. Model-free RL methods learn directly from interactions with the environment, but model-based RL methods can simulate transitions using the learned model, resulting in increased sample efficiency [1]. This is particularly important in domains where each interaction with the environment is expensive. However, learning a model introduces extra complexities, and there is always the danger of suffering from model errors, which in turn affects the learned policy; a common but partial solution in this latter scenario is to use model predictive control, where planning is repeated after small sequences of actions in the real environment [4]. Although deep neural networks can potentially produce very complex and rich models [7, 15], sometimes simpler, more data efficient methods are preferable [9].

2 PART B.

REFERENCES

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.
[2] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.
[3] R. Bellman. Dynamic programming. *Princeton, USA: Princeton University Press*, 1(2):3, 1957.
[4] D. P. Bertsekas. Dynamic programming and suboptimal control: A survey from adp to mpc. *European Journal of Control*, 11(4-5):310–334, 2005.
[5] G. Cuccu, M. Luciw, J. Schmidhuber, and F. Gomez. Intrinsically motivated neuroevolution for vision-based reinforcement learning. In *Development and Learning (ICDL), 2011 IEEE International Conference on*, volume 2, pages 1–7. IEEE, 2011.
[6] M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.
[7] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.
[8] F. Gomez and J. Schmidhuber. Evolving modular fast-weight networks for control. In *International Conference on Artificial Neural Networks*, pages 383–389. Springer, 2005.
[9] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[10] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 7(1), 2015.
[11] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.
[12] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.
[13] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1061–1068. ACM, 2013.
[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
[15] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871, 2015.
[16] R. S. Sutton. Temporal credit assignment in reinforcement learning. 1984.
[17] R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.
[18] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL*, 18(5):620–634, 2010.
[19] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.